Non-Parametric Learning for Monocular Visual Odometry

Vitor Campanholo Guizilini

A thesis submitted in fulfillment
of the requirements of the degree of
Doctor of Philosophy

Australian Centre for Field Robotics
School of Information Technologies
The University of Sydney

March 2013

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Vitor Campanholo Guizilini

12 March 2013

Abstract

Vitor Campanholo Guizilini
The University of Sydney

Doctor of Philosophy
March 2013

Non-Parametric Learning for Monocular Visual Odometry

This thesis addresses the problem of incremental localization from visual information, a scenario commonly known as visual odometry. Accurate localization is a fundamental aspect in autonomous navigation, where a vehicle should be able to position itself on the surrounding environment in order to perform tasks such as mapping, obstacle avoidance and path planning. Current visual odometry algorithms are heavily dependent on camera calibration, using a pre-established geometric model to provide the transformation between input (optical flow estimates) and output (vehicle motion estimates) information.

A novel approach to visual odometry is proposed in this thesis where the need for camera calibration, or even for a geometric model, is circumvented by the use of machine learning principles and techniques. A non-parametric Bayesian regression technique, the Gaussian Process (GP), is used to elect the most probable transformation function hypothesis from input to output, based on training data collected prior and during navigation. Other than eliminating the need for a geometric model and traditional camera calibration, this approach also allows for scale recovery even in a monocular configuration by exploring similarities in optical flow, and provides a natural treatment of uncertainties due to the probabilistic nature of GPs. Several extensions to the traditional GP framework are introduced and discussed at depth, and they constitute the core of the contributions of this thesis to the machine learning and robotics community.

Initially, the standard GP derivation is modified to address a multiple-output scenario (MOGP), and the standard MOGP derivation is further extended to allow the estimation of all outputs simultaneously, along with a full covariance matrix, in a novel technique called Coupled Gaussian Process (CGP). This allows the algorithm to exploit dependencies between different degrees of freedom in motion, caused by constraints in vehicle dynamics, to improve localization results. The usual zero mean assumption in the GP derivation is also explored and modified, and a method of incorporating well-established geometric models into the non-parametric framework is presented, creating a semi-parametric approach to visual odometry. The optimization of the GP hyperparameters and camera calibration parameters is conducted simultaneously, so there is still no need for camera calibration, and if this information is available it can be incorporated seamlessly. The issue of environment change during navigation is addressed by introducing online updates to the semi-parametric model, allowing new information to be incorporated and redundant information to be discarded as a way to maintain computational complexity within a certain boundary. Finally, an automatic dynamic object removal algorithm is presented as a way to improve the reliability of optical flow information extracted during navigation, since any relative motion observed should be caused solely by camera rotation and translation.

The proposed framework is tested in a wide variety of scenarios, ranging from urban and off-road 2D environments, using images collected from a modified vehicle equipped with a single camera, to 3D environments, using images collected from an unmanned aerial vehicle UAV) flying over a deserted area. The results show a significant improvement over traditional visual odometry algorithms, and also surpass results obtained using other sensors, such as laser scanners and IMUs. Due to the natural treatment of uncertainties and recovery of a full covariance matrix, incorporation of the visual odometry results obtained using the proposed framework is straightforward. In particular, the incorporation of these results to a SLAM scenario, using a Exact Sparse Information Filter (ESIF), is shown as a way to use loop-closure to decrease global uncertainty.

All you have to decide is what to do with the time that is given to you.

- Mithrandir

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Notation

General Symbol Format

y	Variable
$\mathbf{x}$	Vector
A	Matrix
$\mathbf{M}$	Block-Matrix

General

$p(A)$	Probability of event A
$p(A\|B)$	Probability of event A given event B
$\mathcal{L}$	Cost function
$f(\mathbf{x})$	Function over $\mathbf{x}$
$\phi(\mathbf{x})$	Basis function
Φ	Design Matrix
I	Identity Matrix
K^{-1}	Matrix inversion
K^T	Matrix transpose
$\|K\|$	Determinant of matrix K
$\|\mathbf{x}\|$	Euclidean length of vector x
x_i	i^{th} element of vector $\mathbf{x}$
K_{ij}	Element of matrix K at row i and column j
$\mathbf{w}$	Parameter vector

Data Format

X	Observation set
$\mathbf{x}_i$	i^{th} observation
D	Dimension of input space
$\Re^D$	Input space

$\mathbf{y}$	Output vector
y_i	i^{th} output
T	Number of outputs
$\Re^T$	Output space

Camera Geometry

$P = (X, Y, Z)^T$	Tridimensional point
$p = (x, y)^T$	Image point
f	Focal length
C	Calibration matrix
T	Transformation matrix
$\mathbf{t}$	Translation vector
R	Rotation matrix
F	Fundamental Matrix
E	Essential Matrix
$\mathbf{e}$	Epipolar point
$(\mathbf{x} - \mathbf{e})$	Epipolar line
α	Epipolar plane

Gaussian Processes

μ	Mean value of a distribution
σ^2	Variance value of a distribution
K	Covariance matrix
$K(X, X)$	Covariance matrix between training data
$K(X, \mathbf{x}_*) = K_*$	Covariance matrix between training data and query point
$k(\mathbf{x}, \mathbf{x}')$	Covariance function evaluated between $\mathbf{x}$ and $\mathbf{x}'$
$m(\mathbf{x})$	GP mean function
$v(\mathbf{x})$	GP variance function
σ_n^2	GP noise level
σ_f^2	GP signal variance level
θ	GP hyperparameter set

Robotics

$\mathbf{x}_t$	Vehicle pose at timestep t
μ_t,Σ_t	Mean prediction and uncertainty for vehicle pose at timestep t
$\mathbf{u}_t$	Control signal at timestep t
$\mathbf{z}_t$	Observation taken at timestep t
$\mathbf{v}_t$	Vehicle velocity at timestep t

SLAM framework

$\mathbf{m}_i$	i^{th} landmark observed
μ_i, Σ_i	Mean prediction and uncertainty for landmark i
Σ_s	Measurement uncertainty
$\mathbf{M}$	Environment map
ν_t,Λ_t	Information vector and matrix
$\boldsymbol{\zeta}_t$	State vector at timestep t
H	Jacobian matrix

Abbreviations

GP	Gaussian Process
SGP	Single Gaussian Process
MOGP	Multiple-Output Gaussian Process
CGP	Coupled Gaussian Process
SPCGP	Semi-Parametric Coupled Gaussian Process
SFM	Structure From Motion
SLAM	Simultaneous Localization and Mapping
ESIF	Exact Sparse Information Filter
RANSAC	Random Sample Consensus
LSQ	Least Squares
ML	Maximum Likelihood
MAP	Maximum a Posteriori
EM	Evidence Maximization
LOO	Leave-One-Out
LOOCV	Leave-One-Out Cross-Validation
PLSC	Probabilistic Least-Squares Classification
SSD	Sum of Squared Differences
SIFT	Scale-Invariant Feature Transform
DoG	Difference of Gaussian
DoF	Degree of Freedom
BBF	Best-Bin-Fit
SVD	Singular Value Decomposition
GPS	Global Positioning System
IMU	Inertial Measurement Unit
ICP	Iterative Closest Point
UAV	Unmanned Aerial Vehicle
SVM	Support Vector Machine
IMG	Image frames used in the visual odometry algorithm
FRT	Feature sets obtained from a image frame
MTC	Matching sets obtained from a pair of image frames
INL	Inlier features filtered from a matching set using RANSAC

Chapter 1

Introduction

1.1 Motivation

This thesis is concerned with vehicle localization, more specifically with vehicle localization based on image information. The ability to position itself on the surrounding environment, and constantly update this position during navigation, is of key importance in most high-level autonomous tasks, such as mapping, path planning and exploration. Without localization, a vehicle is limited to reactive behaviour based solely on current observations, and is incapable of performing actions beyond its sensory range.

There are several sensors capable of performing vehicle localization, with different ranges of application and degrees of accuracy. *Internal* sensors, such as wheel encoders and IMUs, work isolated from the surrounding environment, and thus are only capable of providing *incremental* estimates of vehicle velocity and/or acceleration. These individual estimates are integrated over time, according to a predetermined vehicle dynamic model, to generate a pose estimate. The main disadvantage of incremental localization, also known as *dead reckoning*, is that each individual estimate contains a component of error (drift) that accumulates as part of the integration process, increasing monotonically at each iteration. Any improvement in sensor or model accuracy will serve only to slow, but not prevent, this increase, and eventually the

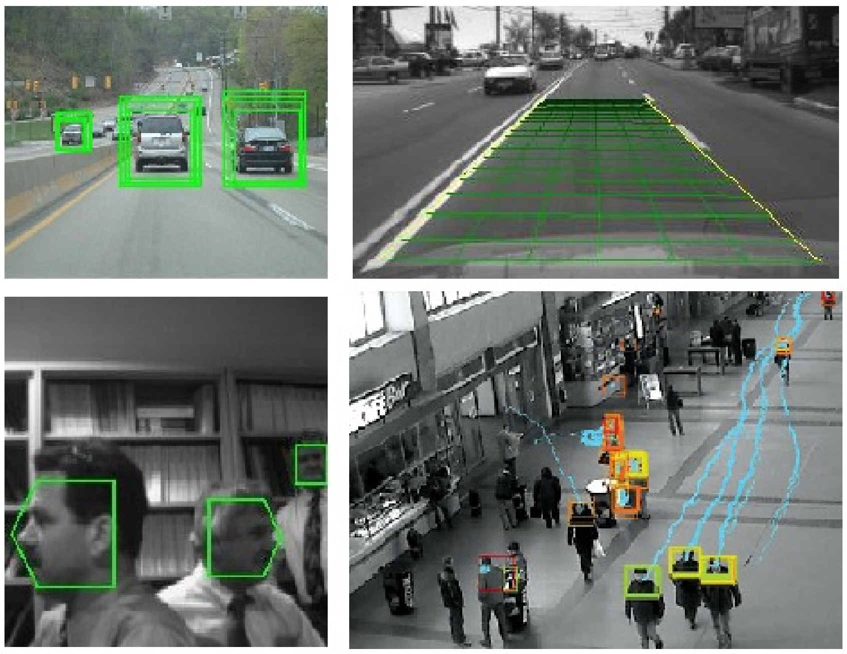


Figure 1.1 – Some common applications of visual sensors in robotics: car detection (upper left), lane segmentation (upper right), face recognition (bottom left) and people tracking (bottom right).

pose estimate becomes so uncertain that it no longer serves any meaningful purpose. For this reason, dead reckoning is insufficient for long-term localization, however it still maintains useful auxiliary information that can be exploited in conjunction with other forms of localization.

Pose estimation with upper boundaries of uncertainty is possible only through the availability of *absolute* measurements, rather than incremental. This is achieved using *external* sensors, which interact in one way or another with the environment to generate pose estimates that are independent of the robot's previous states. A GPS is the simplest form of absolute pose estimation, however it has a limited range of applications as it does not work underwater, indoors or in extra-terrestrial navigation. Another family of external sensors, which include cameras and range-finders (i.e. laser, sonar and radar scanners), work by registering information from the environment, which the vehicle can then use to perform both incremental (by estimating relative motion between measurements) and absolute (by detecting previously visited areas) localization.

Figure 1.2 – Examples of robotic platforms that use visual information for navigational purposes.

The use of cameras in lieu of other external sensors has several benefits: cameras are inexpensive, compact and low-power consumption sensors capable of producing a dense and rich representation of the environment. They possess a wide field of view both horizontally and vertically, a range that is only limited by illumination or image resolution (or solid objects), and are naturally capable of registering motion in all 6 degrees-of-freedom that constitute 3D navigation. The colour and texture information provided by a visual representation of the environment can also be used in a wide variety of other applications (see Fig. 1.1) that pose a challenge to any other sensor. Furthermore, recent advances in computational power now allow the real-time processing of this information, which further contributes to the increase in popularity of cameras in robotics.

The act of performing vehicle localization based on image information is known as *visual odometry*, a term chosen due to its similarities with wheel odometry, where motion is estimated by integrating wheel rotation over time. Likewise, visual odometry operates by incrementally estimating camera translation and rotation based on the relative motion of structures around the vehicle, assuming a static environment. For a visual odometry algorithm to work effectively, there should be sufficient illumination and sufficient texture for feature extraction and matching. In addition, consecutive frames should be captured in such a way that ensures sufficient scene overlapping between them.

1.2 Overview on Visual Odometry

The use of visual sensors in autonomous navigation is far from new, and can be traced back at least to 1976, with Gennery and Moravec using feature tracking for course correction in the Stanford AI Lab Cart [87]. The functionality of theses sensors was later extended to include egomotion estimation [86], achieved by tracking a set of stationary landmarks over a sequence of frames and calculating their relative motion. Since then, visual odometry has been extensively and successfully employed in a wide variety of applications (Fig. 1.2), such as autonomous aircrafts [56], underwater vehicles [9], space exploration [15] and indoor/outdoor ground navigation [52, 108].

Solutions to the visual odometry problem can be broadly divided into two categories: *Structure-From-Motion* (SFM), which draws from multi-view projective geometry [47]; and *Probabilistic Filtering*, which draws from state estimation methods [129, 139]. Overall, probabilistic filtering approaches produce accurate results in small environments, however they do not scale well to larger problems where a vast amount of features is available. SFM approaches can be further divided into two categories: *stereo* and *monocular* configurations. Stereo configurations [56, 146] use a multi-camera array to capture several images of the environment simultaneously, from different vantage points. In this scenario, feature depth estimates can be recovered directly from the binocular disparity between images, and this information is then used to infer camera translation and rotation. Monocular configurations [108, 128] use a single camera, which is essentially a bearing-only sensor and therefore incapable of providing feature depth estimation directly. One well-known limitation of monoc-

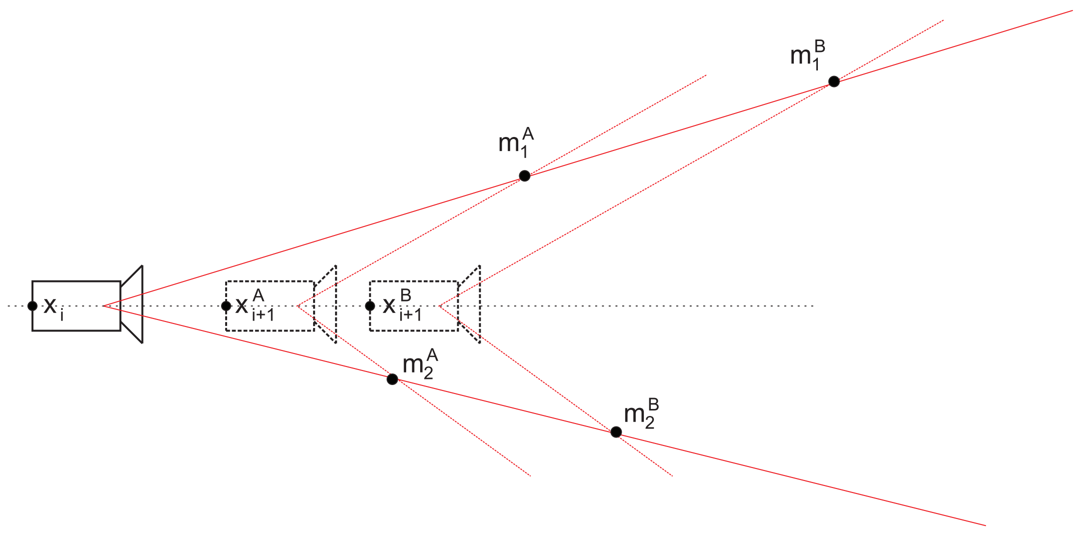


Figure 1.3 – Example of monocular visual odometry and its inherent scale ambiguity. From visual information alone, it is impossible to distinguish between hypothesis *A* and *B* for vehicle pose and landmark location.

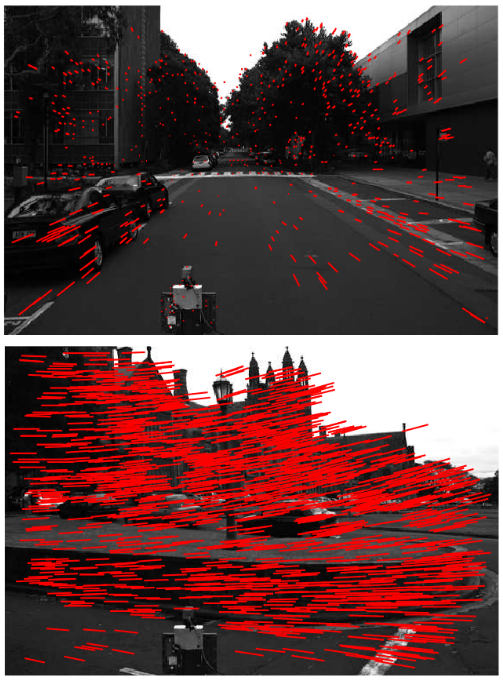
ular visual odometry is *scale ambiguity* (Fig. 1.3), caused by the parallax effect (an observed object could be close and moving slowly or far away and moving fast, it is impossible to know from a single image). This means that, unless special circumstances are considered, such as a ground-plane assumption [61] or a particular vehicle dynamics [109], visual odometry algorithms are only capable of recovering vehicle translation up to a scaling factor.

All SFM-based approaches to visual odometry, however, are calibration-dependent, in the sense that the transformation between image information and vehicle motion is calculated using a *geometric model* of the visual system [47]. This geometric model is governed by the camera's *intrinsic parameters*, which are obtained via calibration prior to the beginning of navigation. This means that any localization estimate provided by such approaches will only be, at most, as accurate as the geometric model and calibration parameters used. Over the years several methods for camera calibration have been proposed, however there is no guarantee that the resulting parameters will not change over time, due to vibration, mechanical shocks or changes in temperature. The use of self-calibration algorithms [17, 34] is attractive because it allows the tracking and auto-correction of changes in calibration parameters, but they still assume a known fixed geometric model, thus limiting the flexibility of the solution.

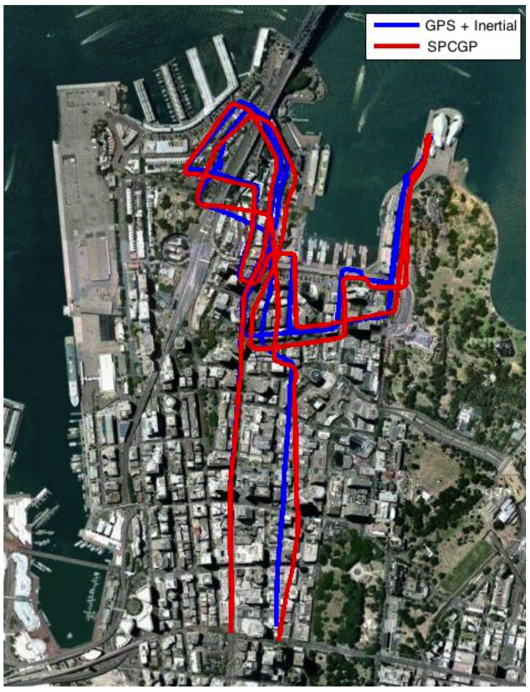
An alternative to explicitly defining a geometric model is the use of *machine learning* techniques, a field of research that is concerned with the development of algorithms that take as input empirical data and attempt to learn the underlying function from which this data was generated. By introducing a *training dataset* containing examples of input-output pairs, it is possible to estimate a transformation function that directly maps image information to vehicle motion, without any prior knowledge of the visual system or environment structure. Although intuitive, this approach has been scarcely used in visual odometry, most notably in [102], where the authors use a KNN-Learner voting method to estimate changes in pose, with each learner taking as input the average of the sparse optical flow in a grid-divided image. A similar idea is explored in [103], where a constant pixel depth is assumed and the Expectation-Maximization (EM) algorithm [26], in conjunction with an extension to PPCA [131], is used to perform a linear mapping between optical flow and incremental motion.

1.3 Problem Statement

This thesis introduces the use of *Gaussian Processes* (GPs), a non-parametric Bayesian regression technique, as a valid and attractive approach to address the problem of visual odometry from a machine learning perspective. A GP is a non-parametric technique in the sense that it does not explicitly define a model between inputs and outputs, maintaining instead a probabilistic distribution over an infinite number of possible functions. Overly complex solutions are penalized and data fit is rewarded, based on a cost function that quantifies the relationship between training inputs, to elect a continuous approximation of the underlying function that generalizes well over new inputs. Furthermore, each inference also produces a corresponding uncertainty estimate, that can be used to gauge the resulting model's confidence in its own predictions. The proposed algorithm is tested in a wide variety of scenarios, ranging from highly dynamic urban and unstructured off-road 2D environments, using images



(a)



(b)

Figure 1.4 – Results obtained with the proposed method. (a) Examples of visual information used as input (translational and rotational optical flow). (b) Localization results obtained in an urban environment using a modified vehicle equipped with a single uncalibrated camera.

collected from a ground vehicle (Fig. 1.4), to 3D unconstrained navigation, using images collected from an unmanned aerial vehicle (UAV), with results that consistently outperform traditional, purely geometric, visual odometry techniques.

The objective of this thesis is to create a solution to the problem of visual odometry that reduces the need for any prior knowledge of the visual system and/or environment, allowing the system to learn the transformation between image information and vehicle motion directly from training data, in an online fashion. If such knowledge (such as the geometric model or the camera calibration parameters) is available, it can be seamlessly incorporated into this framework and then further refined during navigation, improving results on any particular visual odometry algorithm currently available.

1.4 Contributions

The main contributions of this thesis[1] are enumerated as follows:

1. **Monocular visual odometry algorithm based on machine learning principles**. A Multiple-Output Gaussian Process (MOGP) is used to learn the transformation function from image information directly into vehicle motion, eliminating the need for a geometric model or even traditional camera calibration. The benefits of this approach are three-fold:

    - **Full covariance matrix recovery.** The traditional MOGP derivation is extended to allow the simultaneous estimation of all outputs (linear and angular velocities). This new methodology exploits dependencies between these outputs, generated by constraints in vehicle dynamics, to improve accuracy on each individual estimate. These dependencies are quantified as the cross-terms in the resulting covariance matrix.
    - **Scale recovery in monocular configuration.** The proposed framework uses training data obtained from a different and independent sensor as ground-truth. If this sensor is capable of scale estimation (i.e. range sensors), this information is encoded into the resulting non-parametric model and can be recovered by exploiting structure similarities between training and testing images.
    - **Natural treatment of uncertainties.** Because Gaussian Processes are a Bayesian probabilistic technique, all inferences conducted using the proposed framework will naturally provide a measurement of uncertainty. This is of key importance in further treatments of the results, such as data fusion or incorporation into a SLAM scenario.

[1]The majority of the contributions proposed in this thesis (Chapters 3 and 4) was accepted for publication at the International Journal of Robotics Research (IJRR) 2013, under the title *Semi-Parametric Learning for Visual Odometry.*

2. **Temporal dependencies in the Gaussian Process framework**. The standard spatial correlation between features in the same frame used by the Gaussian Process framework is extended to include temporal dependencies between features in subsequent frames. This approach works under the assumption that vehicle velocity will change smoothly during navigation, and significantly increases the amount of information available for inference.

3. **Semi-parametric Gaussian Processes.** The traditional zero-mean assumption in Gaussian Processes is modified and a standard Structure-From-Motion geometric model is incorporated into the proposed framework, as the new mean function. The camera calibration parameters are learned simultaneously with the GP hyperparameters, and if they are available they can be incorporated seamlessly and further refined during navigation.

4. **Online learning of visual odometry estimators.** The semi-parametric model obtained during training is constantly updated during navigation, incorporating new information as it becomes available and discarding redundant or old information, in order to maintain computational time roughly constant. The online update of the semi-parametric model allows the algorithm to gradually adapt to new environments, decreasing the sensitivity to similar training and testing conditions. All new information incorporated is obtained directly from the GP inference process, so there is still no need for a different sensor during navigation.

5. **Vectorized representation of optical flow information.** A novel method for representing optical flow information is proposed, where it is stored as a single vector of fixed dimension. This representation also retains the spatial structure of the image, a valuable characteristic since optical flow patterns tend to vary radically and consistently throughout each frame.

6. **Automatic segmentation of dynamic objects.** A novel technique for the self-supervised segmentation of dynamic objects from a static background is proposed. The RANSAC algorithm is used to provide an initial classification

between classes, and these results are used to iteratively train a Gaussian Process during navigation. The dynamic objects are then removed before the optical flow vectorization stage, to ensure that only a static background is used for visual odometry estimation. This technique is shown to radically improve results obtained during city driving, reducing the influence of pedestrians, cars and buses in the final localization estimates.

1.5 Thesis Outline

Chapter 2 presents the theoretical background that will form the basis to the remainder of the thesis. It starts by providing an overview on regression techniques, starting with parametric techniques, both deterministic and Bayesian, and later moves on to non-parametric models, in specific a non-parametric Bayesian regression technique known as Gaussian processes. The Gaussian process model is presented, along with the various covariance functions available in the literature, techniques for hyperparameter optimization and extension to multiple outputs. The chapter then switches to computer vision, introducing the various aspects in which a camera can be an attractive sensor in robotic applications. Algorithms for feature extraction and matching are presented and discussed, and the RANSAC algorithm is described as a valuable tool for outlier removal. Finally, the camera model used in this thesis is presented, and the chapter concludes by describing how vehicle motion can be recovered from image information using this camera model, which forms the basis for visual odometry estimation.

Chapter 3 introduces the proposed algorithm for visual odometry estimation, in which the geometric model is substituted by a non-parametric Gaussian process model, thus eliminating the need for conventional camera calibration. An overview of a simplified version of the algorithm is presented, and the remainder of the chapter is dedicated to describing each of its particular stages. Initially it is shown how to convert image information into a vector that can be used as the input vector for the GP framework, in such a way that spatial structure is maintained to allow a direct comparison between different inputs. The GP framework is then introduced, describing the two different vehicle models used in the experiments (a 2D constrained ground vehicle model and an unconstrained 3D aerial vehicle model), the process of covariance function selection and techniques for hyperparameter optimization that address the high-dimensionality of the problem at hand. Lastly, experimental results obtained using the proposed algorithm, both in 2D and 3D environments, are presented and discussed, along with comparisons to a standard structure from motion algorithm and possible shortcomings of the proposed algorithm that should be addressed before it becomes a viable solution to the visual odometry problem.

Chapter 4 extends the visual odometry algorithm introduced in the previous section to address its various shortcomings. Five extensions are proposed: 1) the modelling of cross-dependencies between different outputs, that allows the recovery of a full covariance matrix; 2) the introduction of temporal dependencies between outputs of subsequent frames, that increases the amount of information available for inference; 3) the incremental update of the covariance matrix, that allows the non-parametric model to gradually adapt to new environments; 4) the incorporation of a geometric model as the mean function for the GP framework, that provides an initial estimate that is then further refined by the non-parametric model; 5) and finally an extension to the SLAM framework, in which all vehicle poses are tracked over time and a loop-closure algorithm is used to detect when an area is revisited, with this information being used to globally decrease uncertainty. Further experiments are conducted to evaluate the effectiveness of such extensions, and also to test the proposed method's ability to generalize over different training and testing conditions.

Chapter 5 introduces a novel technique for the segmentation of dynamic objects, also based solely on visual information and using the GP framework described in previous chapters. This is done in order to detect and remove dynamic objects during navigation, which could compromise results by incorporating optical flow information that is not generated by camera translation and rotation. An unsupervised method for obtaining ground-truth data, based on the RANSAC algorithm, is presented, alongside a new descriptor used to generate the vector that serves as input for the

GP framework. It produces a dense binary classification of the entire image, where each pixel is classified either as a static or a dynamic object, and a measurement of confidence in such classification is also provided. Once this classification is available, it is possible to filter out all objects that are deemed dynamic and use solely static structures to generate the optical flow information. Experiments are conducted to validate the proposed algorithm's ability to correctly segment dynamic objects using different camera configurations, and also to show the improvement it produces on visual odometry applications.

Chapter 6 concludes the thesis and provides a brief summary of the contributions, alongside potential future research directions that can be taken based on the work conducted here.

Chapter 2

Theoretical Background

This chapter is a review on the main tools and techniques used to develop the visual odometry algorithm proposed in this thesis. It starts with an overview of parametric regression techniques, including the least-squares model, maximum likelihood and Bayesian inference. Afterwards, it moves on to Gaussian processes as a non-parametric Bayesian regression technique, including different covariance functions, techniques for hyperparameter optimization and extension to multiple-outputs. Finally, it concludes by providing an overview of different computer vision techniques, highlighting different methods for feature extraction and matching, outlier removal techniques, and the geometric camera model that will be used during experiments.

2.1 Parametric Regression

Over the years, regression techniques have become increasingly popular in robotics, as a way to learn a system's characteristics without having to explicitly define a model. If a comprehensive set of observations $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, $\mathbf{x}_n \in \Re^D$ and their corresponding outcomes $\mathbf{y} = \{y_1, \dots, y_N\}$, $y_n \in \Re$ is available, statistical modelling is able to provide an estimation of the *underlying function* $y = f(\mathbf{x})$ and predict the outcome of new observations. This is especially valuable in situations where the phenomenon that generated the data is too complex or unpredictable to be properly modelled using classical approaches.

This section addresses one form of regression known as *parametric* regression, in which the underlying function is modelled by a finite number of quantifiable characteristics, or parameters. In other words, the underlying function is defined as $f(\mathbf{x}, \mathbf{w})$, where $\mathbf{w}$ is the parameter vector and $f(\mathbf{x}, .)$ is the *functional mapping.* Naturally, some parameters will be better than others at explaining the observed outputs, and the challenge now is to find the ones that provide the "best" explanation.

2.1.1 Least-Squares Regression

One way of defining the "best" model is by finding the parameters that minimize a certain *cost function* $\mathcal{L}(\mathbf{w})$. By this definition, better models will be the ones with lower costs. A common cost function is the sum of squared errors:

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} (y_n - f(\mathbf{x}_n, \mathbf{w}))^2, \tag{2.1}$$

that favours models in which the transformed inputs are closer to the outputs in the Euclidean space. The solution $\mathbf{w}_{LSQ}$ obtained by minimizing $\mathcal{L}(\mathbf{w})$ with respect to $\mathbf{w}$ is known as the *least-squares* regression model. Applications of this model include back-propagation neural network training [104], where the weights are optimized according to the gradient of Eq. 2.1, and polynomial regression [38], where the functional mapping is a polynomial and the parameters are the polynomial coefficients.

One problem with least-squares regression is the lack of probabilistic treatment of uncertainties. The resulting model is capable of providing a scalar prediction at any point in the input space, however there is no corresponding measure of confidence in that prediction. Another problem is that of *overfitting* (Fig. 2.1), in which the resulting model becomes unable to discern between random noise and the underlying function it is trying to explain. This is usually caused by an excessively complex

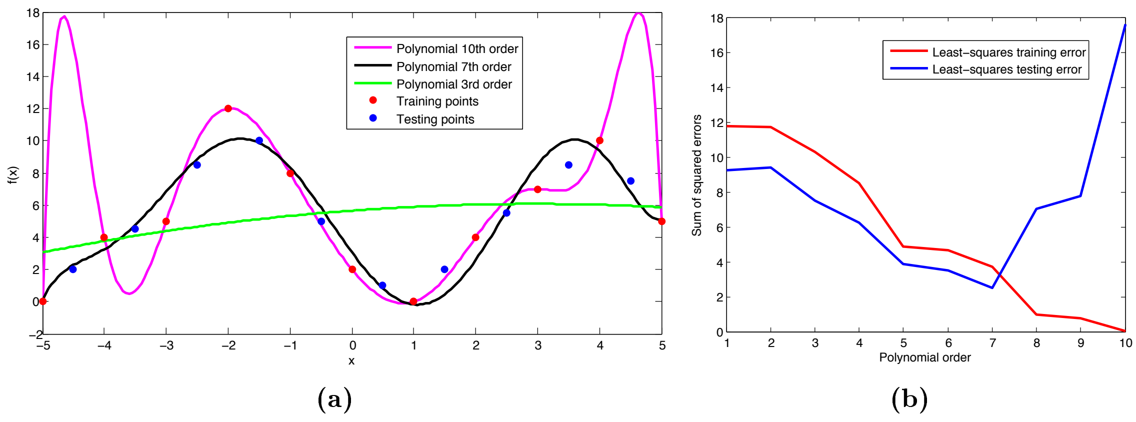


Figure 2.1 – Example of overfitting in a least-squares regression problem using polynomials. In (a) we can see how the resulting model changes as polynomial order increases, and (b) shows the corresponding training and testing errors. As expected, both training and testing errors initially decrease as polynomial order increases, until overfitting takes over (at the 8th polynomial order mark) and the testing error starts to increase, while the training error continues to decrease.

functional mapping, that interpolates all available inputs to a high degree of precision but has a poor predictive performance with new inputs (generalization ability), as it interprets minor fluctuations as trends.

A straightforward solution to overfitting is the use of a simpler functional mapping, with fewer parameters. However, if the chosen functional mapping is too simple it will also have a poor predictive performance, as it is unable to correctly follow the hidden patterns. Other approaches include cross-validation, regularization and early stopping, which can indicate when further optimization is not resulting in better generalization. This is achieved by either explicitly penalizing overly complex models or testing the model's ability to generalize, by evaluating its performance on a subset of the training data that is withheld during optimization.

2.1.2 Maximum Likelihood

A popular alternative to the problem of fitting a function to input data is to assume a probabilistic distribution over X, with a *density function* $p(X|\mathbf{w})$ that is again completely characterized by the parameter vector $\mathbf{w}$. Assuming that the observations $\mathbf{x}_n$ in X are independent and identically distributed (i.i.d), the *joint probability distribution* is given by:

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N|\mathbf{w}) = p(\mathbf{x}_1|\mathbf{w}) \times p(\mathbf{x}_2|\mathbf{w}) \times \ldots \times p(\mathbf{x}_N|\mathbf{w}). \tag{2.2}$$

Alternatively, it is possible to see this problem from a different perspective by considering the observation set X as fixed and the parameter vector $\mathbf{w}$ as variable. This new distribution is the *likelihood function*, and it is given by:

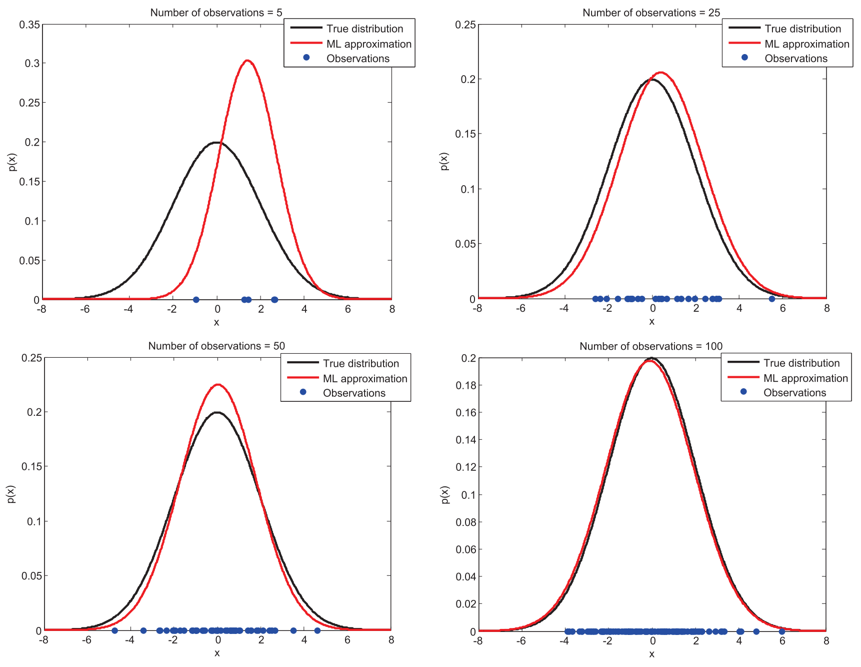
$$\mathcal{L}(\mathbf{w}|X) = p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N|\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{w}). \tag{2.3}$$

For a variety of reasons, the likelihood function is often expressed in terms of its natural logarithm, or as the average *log-likelihood* (Eq. 2.4). The logarithm is a monotonically increasing function, so it achieves its maximum values at the same points as the original function. Computationally, the product of many small probabilities may cause instability and loss of precision, and the logarithm function transforms these multiplications into a more manageable summation. Finding the maximum of a function usually involves taking the derivative of a function, and this is often easier when the function being maximized is a log-likelihood rather than the original likelihood function.

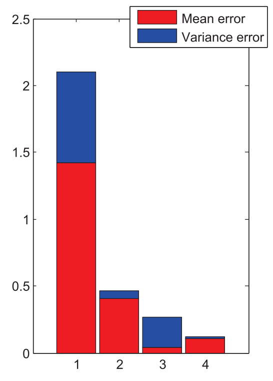
$$\hat{l} = \frac{1}{N} \ln \mathcal{L}(\mathbf{w}|X) = \frac{1}{N} \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathbf{w}) \tag{2.4}$$

The parameters $\mathbf{w}_{ML}$ obtained by maximizing $\hat{l}$ with respect to $\mathbf{w}$ are known as the *maximum likelihood* (ML) estimates. The procedure for maximizing the log-likelihood is dependent on the distribution chosen to represent the underlying function. A common approach is to assume a Gaussian distribution (Fig. 2.2), which is completely defined by its mean μ and variance σ^2. The log-likelihood of such a distribution is of the form:

$$\ln \mathcal{L}(\mu, \sigma|X) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} (\mathbf{x}_n - \mu)^2 - \frac{N}{2} \ln(\sigma^2) - \frac{N}{2} \ln(2\pi), \tag{2.5}$$



(a)



(b)

Figure 2.2 – Maximum likelihood estimates for a Gaussian distribution $\mathcal{N}(0, 2)$. (a) Resulting distributions with different number of observations. (b) Corresponding mean and variance errors for each distribution, as the number of observations increase.

that gives rise to a simple set of equations (Eq. 2.6) to determine $\mathbf{w}_{ML}$. In fact, it can be shown [45] that under this assumption the maximum likelihood model is equivalent to the least-squares model:

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \qquad \sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \mu_{ML})^2. \tag{2.6}$$

The use of maximum likelihood as a regression tool will be addressed here for the particular case of *linear regression*, where the underlying function can be defined as a linear combination of the input variables and some noise ϵ, here assumed to be of a Gaussian distribution $\mathcal{N}(0, \sigma^2)$:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \ldots + w_Dx_D = \mathbf{x}^T\mathbf{w}, \qquad y = f(\mathbf{x}, \mathbf{w}) + \epsilon. \tag{2.7}$$

Under these circumstances, the resulting model will be a linear combination on both the parameters and the inputs, which limits its expressiveness to simple lines. Consequently, it is common to use basis functions $\phi(\mathbf{x})$ to project the D-dimensional inputs into a higher dimensional space M, thus allowing for more flexible solutions. The likelihood function (as presented in Eq. 2.3) is now obtained by maintaining both X and $\mathbf{w}$ fixed and varying the outputs $\mathbf{y}$:

$$\mathcal{L}(\mathbf{y}|X, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} \exp\left(-\frac{(y_n - \phi(x_n)^T\mathbf{w})^2}{2\sigma^2}\right) = \sum_{n=1}^{N} \mathcal{N}(\phi(\mathbf{x}_n)^T\mathbf{w}, \sigma^2 I). \tag{2.8}$$

Similarly, converting the above likelihood function to its average logarithmic form (as shown in Eq. 2.4) provides:

$$\hat{l}_{MLR} = \frac{1}{2}\ln(\sigma^2) - \frac{1}{2}\ln(2\pi) - \frac{1}{2N\sigma^2}\sum_{n=1}^{N}(y_n - \phi(\mathbf{x}_n)^T\mathbf{w})^2. \tag{2.9}$$

The solution $\mathbf{w}_{MLR}$ obtained by maximizing $\hat{l}_{MLR}$ with respect to $\mathbf{w}$ is known as the *maximum likelihood regression model*, and is of the form $\mathbf{w}_{MLR} = (\Phi\Phi^T)^{-1}\Phi\mathbf{y}$, where Φ is the design matrix obtained by projecting the N input observations into the M-dimensional basis function space:

$$\Phi = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \dots & \phi_1(\mathbf{x}_N) \\ \phi_2(\mathbf{x}_1) & \phi_2(\mathbf{x}_2) & \dots & \phi_2(\mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_M(\mathbf{x}_1) & \phi_M(\mathbf{x}_2) & \dots & \phi_M(\mathbf{x}_N) \end{bmatrix}. \tag{2.10}$$

2.1.3 Bayesian Inference

Up to this point, the focus was on finding a set of deterministic parameters that maximize the probability of the outputs given the inputs. A natural progression to the maximum likelihood regression model, where a probabilistic distribution was applied to the observation set, is to place a distribution probability on the parameters themselves, resulting on a model that is capable of estimating the confidence of its own predictions.

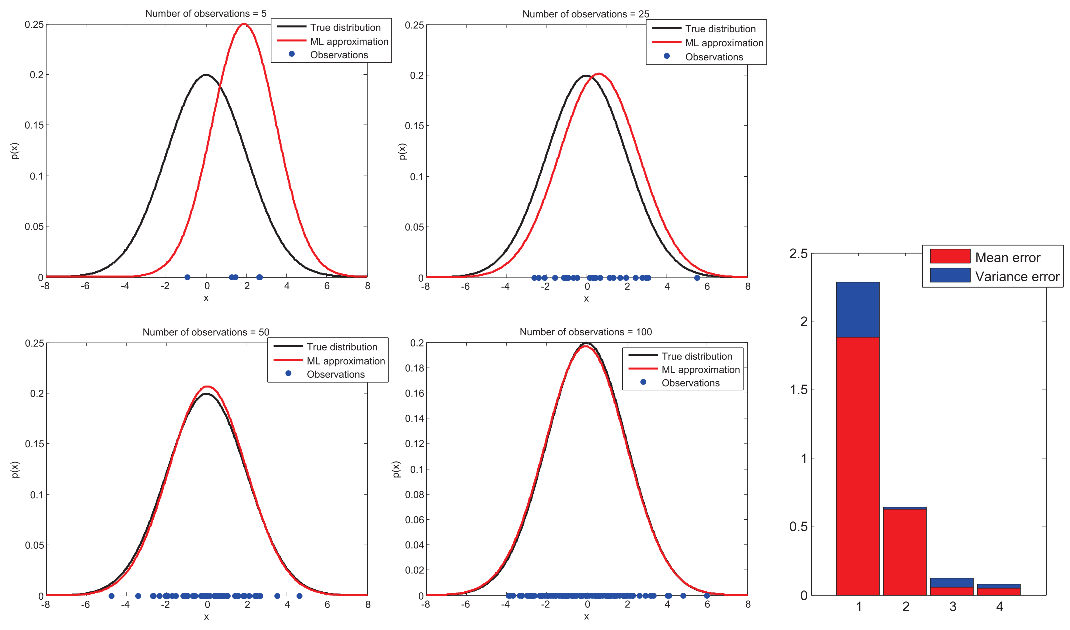
The Bayesian probability theory allows us to do exactly that, by treating the parameters as stochastic latent variables. The core of Bayesian probability lies in the Bayes' Rule, that is used to find the *posterior* distribution over the parameters conditioned on the observations:

$$\overbrace{p(\mathbf{w}|X, \mathbf{y}, \sigma^2)}^{\text{posterior}} = \frac{\overbrace{\mathcal{L}(\mathbf{y}|X, \mathbf{w}, \sigma^2)}^{\text{likelihood}} \overbrace{p(\mathbf{w})}^{\text{prior}}}{\underbrace{\mathcal{L}(\mathbf{y}|X, \sigma^2)}_{\text{marginal likelihood}}}. \tag{2.11}$$

The *prior* distribution is set according to our prior belief about the distribution of the parameters. Since Bayesian inference is usually conducted iteratively, as new data becomes available, the prior distribution in one step is the posterior distribution of the previous one, and at the beginning of the process the prior is manually determined by our knowledge of the parameters (or lack thereof) before any data has been presented. The likelihood function (Eq. 2.8) represents the odds of observing the available data given a specific set of parameters, and the *marginal likelihood* is found by integrating over the likelihood-prior product:

$$\mathcal{L}(\mathbf{y}|X, \sigma^2) = \int \mathcal{L}(\mathbf{y}|X, \mathbf{w}, \sigma^2) p(\mathbf{w}) d\mathbf{w}. \tag{2.12}$$

As the name suggests, the marginal likelihood is independent of any specific parameter, and serves as a normalization constant so that the resulting posterior is a proper probabilistic distribution. Within the Bayesian framework, assuming a 0-1 indicator cost function $L(y_i, y_j) = I(y_i \neq y_j)$, the maximum likelihood estimation method can be combined with the prior distribution to create the parameter estimation technique known as *maximum a posteriori* (MAP). Assuming the Gaussian likelihood function in Eq. 2.8 and a Gaussian distribution $\mathcal{N}(\mu_p, \Sigma_p)$ for the prior, the mean μ_{MAP} becomes:



(a) **(b)**

Figure 2.3 – Maximum a posteriori estimates for a Gaussian distribution $\mathcal{N}(0,2)$ (same observation set as Fig. 2.2, for comparison). (a) Resulting distributions with different number of observations. (b) Corresponding mean and variance errors for each distribution, as the number of observations increase.

$$\mu_{MAP} = \frac{N\Sigma_p^2}{N\Sigma_p^2 + \sigma^2}\mu_{ML} + \frac{\sigma^2}{N\Sigma_p^2 + \sigma^2}\mu_p, \tag{2.13}$$

where μ_{ML} is obtained according to Eq. 2.6. In fact, μ_{MAP} is a linear interpolation between μ_p and μ_{ML} weighted by their respective covariances. This introduces a regularization term to the likelihood function that penalizes larger parameter values, which are indicative of overfitting, and thus tend to produce simpler models. For the special case of $\sigma_w \to \infty$, we have a non-informative prior that leads to $\mu_{MAP} \to \mu_{ML}$. A common criticism of MAP estimation is that they are not very representative of Bayesian methods in general. This is because MAP estimates are punctual, whereas Bayesian methods are known to use distributions to characterize data and draw inferences. Also, unlike ML estimates, the MAP estimate is not invariant under reparametrization, which means that the use of a Jacobian to switch from one parametrization to another has an impact on the location of the maximum.

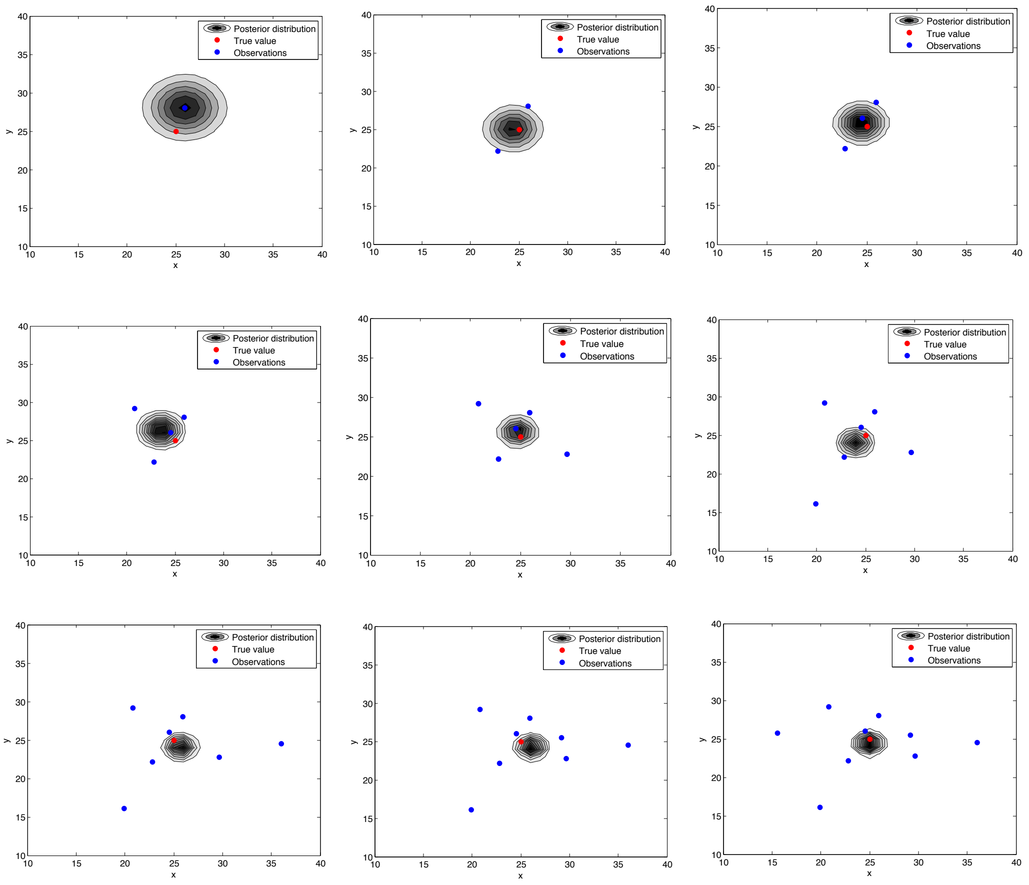


Figure 2.4 – Example of Bayesian inference. The red dot represents the true variable value, and observations are conducted based on a Gaussian distribution with parameters $\mathcal{N}(0, 10)$, that serves as the likelihood function. As the number of observations increase, it is possible to see how the posterior distribution (grey circles, representing a Gaussian distribution within two standard deviations) becomes both more precise and accurate.

Within this framework, inference over a single test point $\mathbf{x}_*$ is a matter of finding the predictive distribution:

$$p(y_*|\mathbf{x}_*, X, \mathbf{y}, \sigma^2) = \int \mathcal{L}(y_*|\mathbf{x}_*, \mathbf{w}, \sigma^2) p(\mathbf{w}|X, \mathbf{y}, \sigma^2) d\mathbf{w} \tag{2.14}$$

$$= \mathcal{N}\left(\frac{1}{\sigma^2}\phi(\mathbf{x}_*)^T A^{-1} \Phi \mathbf{y}, \phi(\mathbf{x}_*)^T A^{-1} \phi(\mathbf{x}_*)\right), \tag{2.15}$$

where $A = \sigma^2 \Phi \Phi^T + \Sigma_p^{-1}$. So, using a single set of parameters to make predictions, the entire posterior density is integrated over. This means that it is not just a single set of parameters that contribute to the predictions, but all parameters, and the predictive contribution from each particular parameter is given by its corresponding posterior probability. The consequence of doing so is a predictive model that, while powerful enough to model a wide variety of underlying functions, is less prone to overfitting.

Another benefit of using Bayesian probability theory is that it provides a full predictive distribution, rather than just a scalar value at each test point (as shown in Fig. 2.4). This is very useful as a measure of the resulting model's confidence on its own predictions: if the predictive distribution is tightly packed around a single value, we can be confident of its ability to correctly model the underlying function (assuming that the chosen parametric form of $f(\mathbf{x}, \mathbf{w})$ is appropriate). On the other hand, if the predictive distribution is spread over a wide range of values, there is a high uncertainty given that particular test point.

2.2 Non-Parametric Bayesian Regression

The previous section addressed the problem of regression from a parametric standpoint, in which the functional mapping is assumed known and its coefficients are optimized according to a certain function in order to generate the best possible model. Conversely, *non-parametric* techniques [138] eschew the need to explicitly define a functional mapping by using the observations themselves to generate the resulting model. As the number of observations increase, so does the model's complexity, and principles such as the Occam's Razor [77] are used to counter the effects of overfitting. While this approach allows for much more powerful and flexible solutions, it also suffers from computational storage and memory requirements, as all data collected must be kept and processed in order to perform inference, whereas parametric models discard current observations after the parameters have been optimized.

This section focuses on *Gaussian processes* (GPs) as a non-parametric Bayesian regression technique [101]. A Gaussian process is a particular case of a *stochastic process*, which is a collection of random variables often used to represent the evolution of a non-deterministic system over time [98]. It maintains a probability density

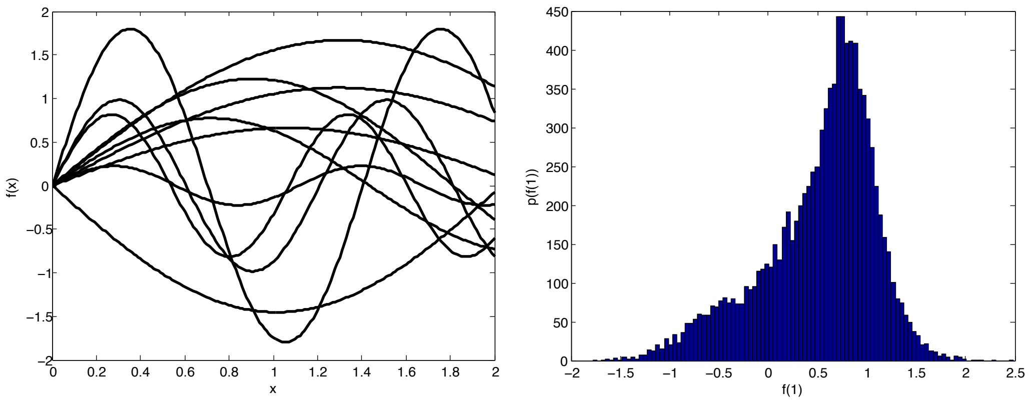


Figure 2.5 – Example of stochastic process sampling. (a) Independent sampling paths obtained based on a function $f(x) = \exp(a)\sin(bx)$, with $a \sim \mathcal{N}(0, \frac{1}{4})$ and $b \sim \mathcal{N}(1, 1)$. (b) Probability density of 10000 sample paths evaluated at $f(1)$.

function $p(f, \theta)$ defined over a function space F, and different hypotheses for the underlying function are sampled directly from this function space (Fig. 2.5). Because the parameters in θ now govern functions, that in turn have their own parameters, they are commonly referred to as *hyperparameters*, and these are optimized based on a positive-definite function that quantifies the relationship between points in the input space.

2.2.1 History of Gaussian Processes

The use of Gaussian processes as a tool for prediction can be traced back to the 1940's, in works such as the Wiener-Kolmogorov prediction theory and time analysis [76, 77]. More recent (1960's) is the introduction of *kriging* [79] as a method for the interpolation of geostatistical data [20], based on the Gauss-Markov theorem. Kriging, named after the mining engineer Danie G. Krige, is identical to Gaussian process regression, but derived and interpreted in a different manner. Furthermore, as a geostatistical method, it is mainly concerned with low-dimensional problems and ignores any probabilistic interpretations [77]. In the statistical community, the use of Gaussian processes to define prior distributions has its origins in 1978, where the theory was applied in the problem of one-dimensional curve fitting [94].

In the machine learning community, the use of Gaussian processes for supervised learning dates back to the 1980's, with the introduction of back-propagation learning in neural networks [104]. The original non-probabilistic derivation was later enhanced by Buntine [14], MacKay [72] and Neal [89], which introduced a Bayesian interpretation that provided a consistent method for handling network complexity [8, 74]. It was shown later [90] that under certain conditions these Bayesian neural networks converge to a Gaussian process with an infinite number of hidden units. This resulted in the introduction of Gaussian processes for regression in the machine learning context [77, 113], among other techniques such as least-squares [142], support vector machine [21] and decision trees [12].

2.2.2 The Kernel Trick

In Section 2.1.2, it was shown how projecting the raw data into a high-dimensional space generates more expressive models, capable of addressing a wider variety of underlying functions. However, this incurs a computational power cost that increases cubically with the dimensionality of the input space, caused by the inversion of matrix A (Eq. 2.15). As the modelling function becomes more complex, the cost of this inversion quickly becomes infeasible. Kernel methods are able to circumvent this limitation by using a function that evaluates the relationship between data points directly, as if they had already been projected into a potentially infinite dimensional space [110]. Eq. 2.15 can then be rewritten as such:

$$\begin{aligned} p(y_*|\mathbf{x}_*, X, \mathbf{y}, \sigma^2) \sim \mathcal{N}(&\phi(\mathbf{x}_*)^T \Sigma_p \Phi (\Phi^T \Sigma \Phi + \sigma^2 I)^{-1} \mathbf{y}, \\ &\phi(\mathbf{x}_*)^T \Sigma_p \phi(\mathbf{x}_*) - \phi(\mathbf{x}_*)^T \Sigma_p \Phi (\Phi^T \Sigma_p \Phi + \sigma^2)^{-1} \Phi^T \Sigma_p \phi(\mathbf{x}_*)), \end{aligned} \tag{2.16}$$

where it is possible to notice a reoccurring pattern on the different ways the high-dimensional terms appear: $\Phi^T \Sigma_p \Phi$, $\phi(\mathbf{x}_*)^T \Sigma_p \Phi$ or $\phi(\mathbf{x}_*)^T \Sigma_p \phi(\mathbf{x}_*)$. A *kernel* can now be defined as a function $k(\mathbf{x}, \mathbf{x})' = \phi(\mathbf{x}) \Sigma_p \phi(\mathbf{x}')$ in the Hilbert space that describes the relationship between two input vectors $\mathbf{x}$ and $\mathbf{x}'$. If this kernel is chosen correctly (a more thorough discussion about different kernel functions is given in Section 2.2.4), it allows the calculation of the high-dimensional dot products directly on the original space. This process of mapping observations from a general set into an inner product space, without ever having to compute the mapping explicitly, is known as the *kernel trick* [3]. Now, the computational cost is dependent on the number N of training points, rather than on the dimensionality M of the basis functions. Consequently, it is possible to use a kernel that represents basis functions with an infinite number of terms, allowing the estimator to model very complex datasets.

2.2.3 The Gaussian Process Model

A Gaussian process is a particular type of stochastic process in which all random variables are Gaussian distributed. Moreover, every finite linear combination of such random variables have a multivariate Gaussian distribution. Assuming two test points $\mathbf{x}$ and $\mathbf{x}'$, and using the Bayesian linear regression model described in Section 2.1.3, with $f(\mathbf{x}) = \phi(\mathbf{x})^T w$ and $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$, the resulting model will have the following *mean* and *covariance functions*:

$$\mathbb{E}[f(\mathbf{x})] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = \mathbf{0} \tag{2.17}$$

$$\mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}\mathbf{w}^T]\phi(\mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}'). \tag{2.18}$$

These two parameters completely describe a Gaussian process, in the sense that any finite collection of samples, $\{f(\mathbf{x}_{*1}), f(\mathbf{x}_{*2}), \ldots, f(\mathbf{x}_{*i})\}$, will have a joint Gaussian distribution. The zero mean assumption in Eq. 2.17 can be made without any loss of generality by correctly normalizing the input data (i.e. subtracting the mean training values), and will be further explored in Section 4.1.4. Furthermore, the covariance function is numerically equivalent to the kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})\Sigma_p\phi(\mathbf{x}')$ described in the previous section. This kernel representation means that inference will now be performed over a function space governed by the covariance function, which in turn is governed by a collection of parameters θ commonly referred to as the *hyperparameter* set (because they govern an entire family of functions, rather than a single one). In other words:

$$\mathbf{f}\,|X, \theta \sim \mathcal{N}(\mathbf{0}, K), \tag{2.19}$$

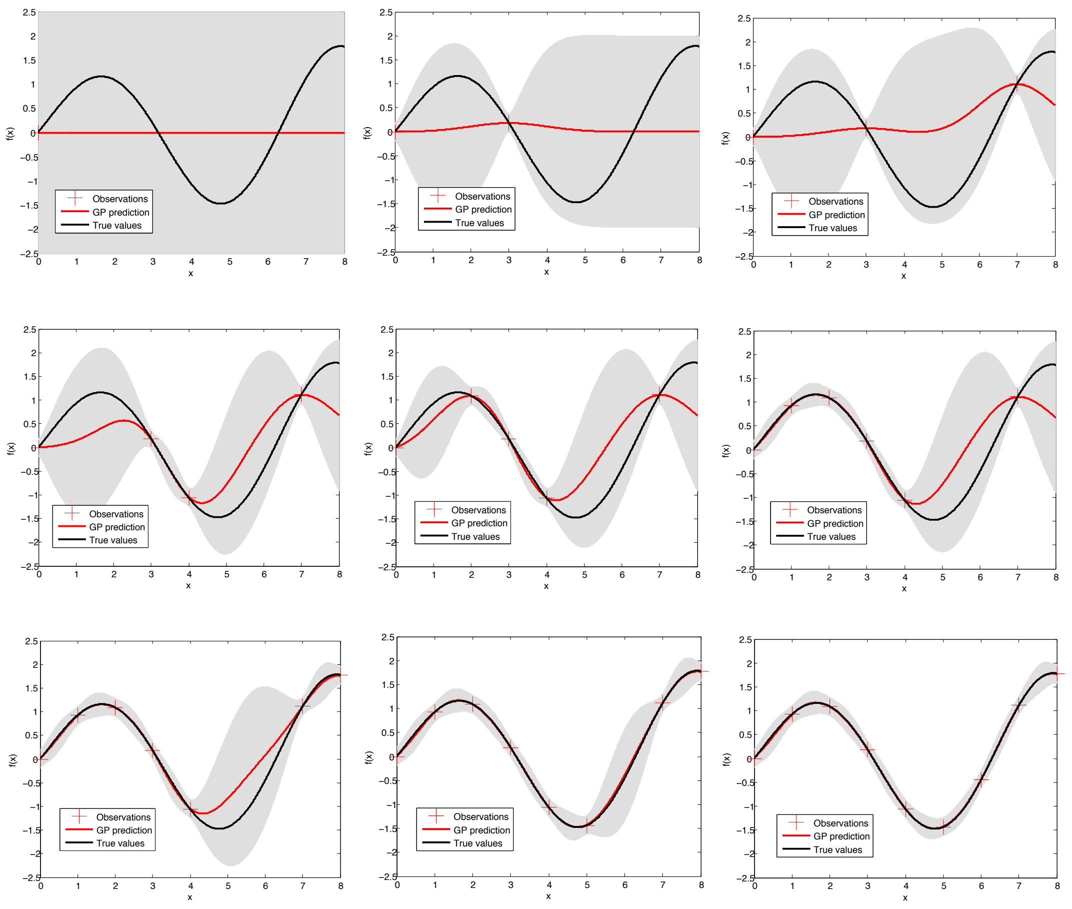


Figure 2.6 – Example of Gaussian process inference. At the beginning (upper left figure) there are no observations, so the predictive model has zero mean and high variance (grey area) throughout the entire input space. As more data is collected, the predictive model becomes more accurate around the observations and local uncertainty decreases.

where K is a $N \times N$ covariance matrix in which the $(i,j)^{th}$ elements are equal to $k(\mathbf{x}_i, \mathbf{x}_j)$. Given a test point $\mathbf{x}_*$, and its associated latent variable f_*, under the Gaussian process framework (with zero mean) the joint distribution of $\mathbf{f}$ and f_* is a multivariate Gaussian, and is found by augmenting Eq. 2.19 with this new information:

$$\left[\begin{array}{c} \mathbf{f} \\ f_* \end{array}\right] \Bigg| X, \theta \sim \mathcal{N}\left(\mathbf{0}, \left[\begin{array}{cc} K & \mathbf{k} \\ \mathbf{k}^T & \kappa \end{array}\right]\right), \tag{2.20}$$

where $\mathbf{k} = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_N)]^T$ is the $N \times 1$ vector formed from the covariance between $\mathbf{x}_*$ and X, and $\kappa = k(\mathbf{x}_*, \mathbf{x}_*)$. Using the Gaussian noise assumption introduced in Eq. 2.7, the joint distribution over the observed outputs and the unobserved target y_* is given by:

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \Bigg| X, \theta \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma^2 I & \mathbf{k} \\ \mathbf{k}^T & \kappa + \sigma^2 \end{bmatrix}\right). \tag{2.21}$$

Finally, given that the joint distribution is Gaussian and conditioning on $\mathbf{y}$, the predictive mean $\bar{f}_*$ and variance $\mathcal{V}(\bar{f}_*)$ are:

$$\begin{aligned} p(y_* | \mathbf{y}, X, \theta, \sigma^2) &= \mathcal{N}(\bar{f}_*, \mathcal{V}(\bar{f}_*)) & (2.22)\\ \bar{f}_* &= \mathbf{k}^T (K + \sigma^2 I)^{-1} \mathbf{y} & (2.23)\\ \mathcal{V}(\bar{f}_*) &= \kappa + \sigma^2 - \mathbf{k}^T (K + \sigma^2 I)^{-1} \mathbf{k}. & (2.24) \end{aligned}$$

This set of equations allows the calculation of a Gaussian predictive distribution for any test point $\mathbf{x}_*$, given a covariance function $k(.,.)$ and its corresponding hyperparameter set θ (as depicted in Fig. 2.6). The marginal likelihood of a Gaussian process is obtained according to Eq. 2.12, but now integrating over the latent variables $\mathbf{f}$:

$$\begin{aligned} \mathcal{L}(\mathbf{y} | X, \theta, \sigma^2) &= \int \mathcal{L}(\mathbf{y} | \mathbf{f}, X, \theta, \sigma^2) p(\mathbf{f} | X, \theta) d\mathbf{f} & (2.25)\\ &= \int \mathcal{N}(\mathbf{f}, \sigma^2 I) \mathcal{N}(\mathbf{0}, K) d\mathbf{f} & (2.26)\\ &= \frac{1}{(2\pi)^{\frac{N}{2}} |K + \sigma^2 I|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \mathbf{y}^T (K + \sigma^2 I)^{-1} \mathbf{y}\right), & (2.27) \end{aligned}$$

which, if converted to the logarithmic form, gives rise to the following *marginal log-likelihood*:

$$\ln \mathcal{L}(\mathbf{y} | X, \theta, \sigma^2) = -\frac{N}{2} \ln(2\pi) - \frac{1}{2} \ln(|K + \sigma^2 I|) - \frac{1}{2} \mathbf{y}^T (K + \sigma^2 I)^{-1} \mathbf{y}. \tag{2.28}$$

In its original derivation, the GP framework has a computational complexity of $\mathcal{O}(n^3)$ in time and $\mathcal{O}(n^2)$ in memory, mostly due to the cost of maintaining and inverting the covariance matrix K. Sparse implementations [22] are able to further decrease the computational cost, by approximating the full covariance matrix using a subset of points. However, this was not explored in this work because the datasets used during experiments were not big enough to warrant such approximations.

2.2.4 Covariance Functions

As described in the previous section, the covariance function plays a pivotal role in the Gaussian process framework, because it encodes the assumptions about the functions the model is trying to learn. Specifically, it quantifies how *similar* each point in the input space is to each other, a concept that is of great importance in supervised learning techniques where inference on new data is based on previous examples of the same underlying function. For example, it is a common assumption that two inputs close to each other will have similar outputs.

Over the years several kernels have been proposed as covariance functions, with different properties that allow the modelling of a wide variety of underlying functions. Furthermore, operations such as summing, multiplying and convolution have been defined as valid operations to be performed on kernels [101], and thus arbitrarily complex covariance functions can be obtained by combining simple kernels and their corresponding hyperparameter sets. In order to qualify as a covariance function, a kernel must be *positive-definite*, or in other words it must satisfy the following equation:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} w_i k(\mathbf{x}_i, \mathbf{x}_j) w_j \geq 0, \tag{2.29}$$

for any $\mathbf{x}_n \in \mathcal{R}^D$ and $\mathbf{w} \in \mathcal{R}$. Broadly, covariance functions can be divided into two categories: *stationary* and *non-stationary*, according to how they react to translation over the coordinate system.The remainder of this section is dedicated to providing an overview of these two categories, along with examples of the most commonly used covariance functions and possible applications.

Stationary Covariance Functions

Informally, a system is deemed stationary if its properties do not change when there is a shift in time or space. Similarly, a stationary covariance function is derived from a kernel in which the relationship between two inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ does not depend on their absolute position on the coordinate system, but rather on their relative position $\mathbf{r}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$ to each other. Or, in other words:

$$k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i + h, \mathbf{x}_j + h) = k(\mathbf{x}_i - \mathbf{x}_j) = k(\mathbf{r}_{ij}). \tag{2.30}$$

Stationary covariance functions are commonly used to model patterns that occur regularly on the observed data, however they lack the ability to treat different portions of the input space differently. For this reason, more complex underlying functions often require the use of the more generic non-stationary covariance functions, even though these may still be used in conjunction to model a broader spectrum of behaviours.

Exponential Covariance Function. Considered the quintessential covariance function, the squared exponential is a subgroup of the broader family of *exponential* covariance functions, which are governed by the following equation:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-d(\mathbf{x}_i, \mathbf{x}_j)}{l}\right), \tag{2.31}$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ is a measure of distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ and l is a scaling factor, commonly known as the *length-scale*, that indicates how far two points have to be from each other for the output to change significantly. The squared exponential covariance function for a D-dimensional input space can now be defined as:

$$k_{SQ}(\mathbf{r}_{ij}) = \sigma_f^2 \exp\left(-\frac{1}{2}\mathbf{r}_{ij}^T L^{-1} \mathbf{r}_{ij}\right), \tag{2.32}$$

which is essentially an infinite number of Gaussian basis functions placed over the input space. The term σ_f is a signal variance parameter that scales the entire covariance function, and is equivalent to the variance Σ_p placed on the Gaussian prior.

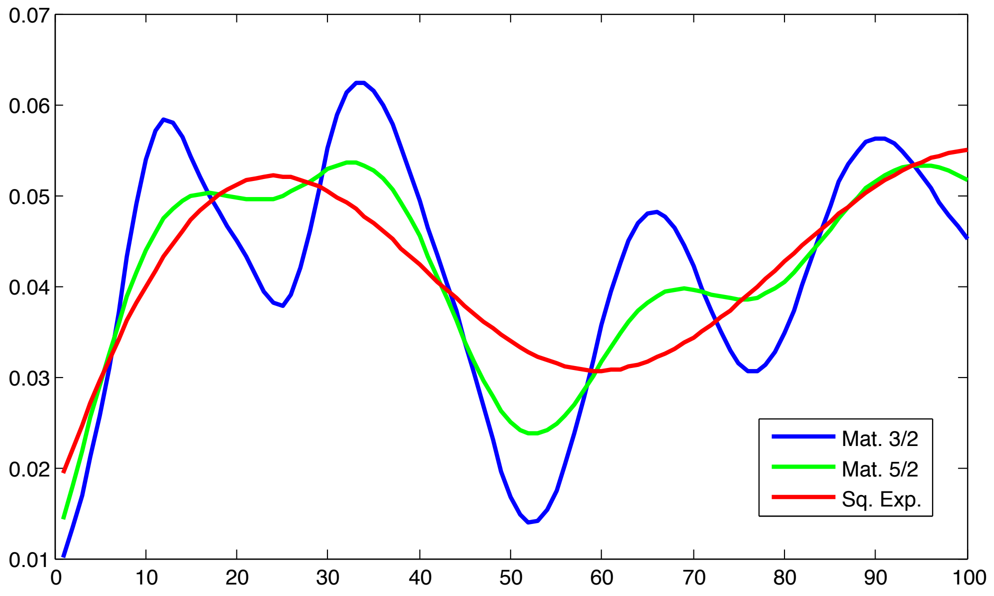


Figure 2.7 – Example of Gaussian prediction using different stationary covariance functions, showing how smoothness increases as the kernel changes from Matérn 3/2 to Matérn 5/2 to squared exponential (which is essentially Matérn ∞).

The matrix L is a $D \times D$ *length-scale matrix*, that models the influence of each input dimension on the output and can be used as a method of automatic relevance determination [90]. The squared exponential covariance function is infinitely divisible, and therefore appropriate for modelling smooth functions. Unfortunately, very few practical datasets are smooth [123], which limits the applicability of this covariance function in most real scenarios.

Matérn Covariance Function. The Matérn may be considered a generalization of the squared exponential covariance function, that addresses the problem of smoothness (Fig. 2.7) discussed previously. It is defined by:

$$k_\nu(\mathbf{r}_{ij}) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu \mathbf{r}_{ij}}}{L} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu \mathbf{r}_{ij}}}{L} \right), \tag{2.33}$$

where ν is a positive parameter, L is the length-scale matrix and K_ν is a modified Bessel function [1]. The Matérn covariance function is $\nu - 1$ times differentiable, allowing for a better control over the smoothness, and as $\nu \to \infty$ it converges to the squared exponential covariance function (Eq. 2.32). Other values for ν that are particularly attractive for modelling purposes, and produce elegant simplifications of the original formulation, are:

$$k_{\nu=3/2}(\mathbf{r}_{ij}) = \left(1 + \frac{\sqrt{3\mathbf{r}_{ij}}}{L}\right) \exp\left(-\frac{\sqrt{3\mathbf{r}_{ij}}}{L}\right), \tag{2.34}$$

$$k_{\nu=5/2}(\mathbf{r}_{ij}) = \left(1 + \frac{\sqrt{3\mathbf{r}_{ij}}}{L} + \frac{5\mathbf{r}_{ij}^2}{3L^2}\right) \exp\left(-\frac{\sqrt{5\mathbf{r}_{ij}}}{L}\right). \tag{2.35}$$

When $\nu = 1/2$ the resulting covariance function becomes too rough, and for $\nu \geq 7/2$ it is already difficult to distinguish between the Matérn and the squared exponential covariance function.

Non-Stationary Covariance Functions

A non-stationary covariance function allows for more expressive models by employing relationships between input points which are dependent on their absolute location on the coordinate system. Now, different portions of the underlying function may behave differently, a common feature in most practical datasets.

Linear Covariance Function. Also known as the *dot-product* covariance function, the linear covariance function $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ is numerically equivalent to a Bayesian linear regression using a polynomial basis function of degree 1 (see Eq. 2.7). A more generic formulation of this covariance function is:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 + \mathbf{x}_i^T \Sigma_l \mathbf{x}_j, \tag{2.36}$$

where σ_0 is a bias term and Σ_l is a general covariance matrix on the components of $\mathbf{x}$. Although not very powerful, this formulation is commonly used as part of a more complex covariance function, that models the linear aspects of the underlying function while other covariance functions tackle the more complex trends.

Neural Network Covariance Function. The derivation of a neural network covariance function [90] is done by assuming a neural network which takes an input $\mathbf{x}$, has one hidden layer with N_H units and linearly combines the outputs of the hidden units with a bias b to obtain $f(\mathbf{x})$. This mapping can be written as:

$$f(\mathbf{x}) = b + \sum_{k=1}^{N_H} v_k h(\mathbf{x}, \mathbf{u}_k), \tag{2.37}$$

where $\mathbf{v}$ is a vector containing the hidden-to-output weights and $h(\mathbf{x}, \mathbf{u})$ is the hidden unit transfer function, which depends on the input-to-hidden weights $\mathbf{u}$. It has been show [51] that as $N_H \to \infty$ networks with one hidden layer become universal approximators for a wide variety of transfer functions. Assuming that b and $\mathbf{v}$ have independent zero-mean distributions of variance σ_b^2 and σ_v^2, respectively, and the weights $\mathbf{u}_k$ for each hidden unit are independent and identically distributed, it can be shown [90] for weights $\mathbf{w}$ that:

$$\mathbb{E}_{\mathbf{w}}\left[f(\mathbf{x})\right] = 0 \tag{2.38}$$

$$\mathbb{E}_{\mathbf{w}}\left[f(\mathbf{x}_i)f(\mathbf{x}_j)\right] = \sigma_b^2 + N_H \sigma_v^2 \mathbb{E}_{\mathbf{u}}\left[h(\mathbf{x}_i, \mathbf{u})h(\mathbf{x}_j, \mathbf{u})\right]. \tag{2.39}$$

Given that the transfer function $h(\mathbf{x}, \mathbf{u})$ is bounded, all moments of the distribution will also be bounded, and hence the central theory can be applied, showing that this stochastic process will converge to a Gaussian process when $N_H \to \infty$. By evaluating $\mathbb{E}_{\mathbf{u}}\left[h(\mathbf{x}_i, \mathbf{u})h(\mathbf{x}_j, \mathbf{u})\right]$, the covariance function of the neural network can be obtained. For example (as deduced in [141]), if the error function $erf(z) = 2/\sqrt{\pi}\int_0^z e^{-t^2}dt$ is chosen as the transfer function, resulting in $h(\mathbf{x}, \mathbf{u}) = erf(u_0 + \sum i = 1^D u_i x_i)$, and the hidden weights have a Gaussian distribution $\mathbf{u} \sim \mathcal{N}(0, \Sigma)$, then the neural network covariance function is of the form:

$$k_{NN}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \arcsin\left(\frac{2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_j}{\sqrt{(1 + 2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_i)(1 + 2\tilde{\mathbf{x}}_j^T \Sigma \tilde{\mathbf{x}}_j)}}\right), \tag{2.40}$$

where $\tilde{\mathbf{x}} = \{1, x_1, \ldots, x_D\}$ is an augmented vector and σ_f is the signal variance parameter.

Sparse Covariance Functions

Another interesting and useful family of covariance functions are the *sparse* covariance functions, also known as covariance functions with compact (or local) support. This means that the covariance between any two given points become exactly zero when their distance exceed a certain threshold, generating a sparse covariance matrix by design. This property may lead to computational advantages, such as less memory usage (not all entries on the covariance matrix have to be stored) and faster inference (sparse matrix multiplication is a much more efficient computational task). Sparse covariance functions may be both stationary and non-stationary, and here we will be presenting one that is stationary [81], with non-stationarity being achieved by multiplying this sparse covariance function with a non-stationary one. For the 1-dimensional case, assuming the basis function $g(x) = cos^2(\pi x)H(1/2 - |x|)$, where H is the Heaviside unit step function, and the transfer function $h(x, u) = g(x - u)$, the following covariance function is obtained:

$$k_1(x_i, x_j) = \sigma_0 \int_{\infty}^{\infty} h\left(\frac{x_i}{l}, u\right) h\left(\frac{x_j}{l}, u\right) du. \tag{2.41}$$

Due to the chosen form of the basis function, the integral of the equation above can be evaluated analytically and is of the form:

$$k_{SP}^1(x_i, x_j, l, \sigma_0) = \begin{cases} \sigma_0 \left[\frac{2+cos\left(2\pi\frac{d}{l}\right)}{3}\left(1 - \frac{d}{l}\right) + \frac{1}{2\pi}\sin\left(2\pi\frac{d}{l}\right)\right] & \text{if } d < l \\ 0 & \text{if } d \geq l \end{cases}, \tag{2.42}$$

where σ_0 is a constant coefficient, $l > 0$ is the length-scale, and $d = |x_i - x_j|$ is the distance between points. Extension to multiple dimensions is done by calculating the direct products for all D dimensions, with $\mathbf{l} = (l_1, l_2, \ldots, l_D)$ being now the length-scale vector:

$$k_{SP}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{l}, \sigma_0) = \sigma_0 \prod_{k=1}^{D} k_{SP}^1(x_{i,k}, x_{j,k}, l_i, 1). \tag{2.43}$$

2.2.5 Hyperparameter Optimization

Once the non-parametric model for the underlying function is defined, it is necessary to find a consistent way to determine the hyperparameters θ. These hyperparameters are dependent on the covariance function $k(.,.)$ chosen, usually including a signal variance σ_f and a length-scale matrix Σ_l (for the sake of simplicity, the noise value σ_n will from now on be included on the hyperparameter vector). Ideally, we would like to integrate over all the hyperparameters in order to make our predictions, or in other words, we would like to find:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X,\theta)p(\theta|X)d\theta. \tag{2.44}$$

Again, the hyperparameter set is dependent on the covariance function chosen. For an arbitrary $k(.,.)$ that is analytically intractable, however, it is possible to use several methods to obtain approximations for the posterior distribution. This section provides an overview of three different methods: *Evidence Maximization* [73], *Cross-Validation* and the *Monte Carlo* [101] approach.

Evidence Maximization

Evidence Maximization (EM), also known as *Marginal Likelihood Maximization* or *Empirical Bayes*, uses an approximation to the integral in Eq. 2.44, based on the most probable set of hyperparameters θ_{EM}:

$$\int p(\mathbf{y}|X,\theta)p(\theta|X)d\theta \approx p(\mathbf{y}|X,\theta_{EM}). \tag{2.45}$$

This approximation is based on the assumption that the posterior distribution $p(\theta|X)$ is sharply peaked around θ_{EM} relative to the variation in $p(\mathbf{y}|X,\theta)$. This approximation is generally valid and the resulting EM estimates are often close to those found using the true predictive distribution [75]. The challenge now becomes one of finding θ_{EM}, which is done by calculating the derivatives of the posterior distribution.

Rewriting the Bayes' Rule (Eq. 2.11) under this framework, we obtain:

$$p(\theta|X,\mathbf{y}) = \frac{\mathcal{L}(\mathbf{y}|X,\theta)p(\theta)}{\mathcal{L}(\mathbf{y}|X,\sigma^2)}, \tag{2.46}$$

which is the same process used to generate the MAP estimates (Eq. 2.13). The marginal likelihood (denominator) is independent of θ, and therefore can be safely ignored in the derivation process. The two remaining terms, the likelihood of θ and the prior on θ, will be considered in their logarithmic form, for reasons discussed previously. The marginal log-likelihood of a Gaussian process is given by Eq. 2.12, and its derivative with respect to θ is:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\frac{1}{2}\left(C^{-1}\frac{\partial C}{\partial \theta}\right) + \frac{1}{2}\mathbf{y}^T C^{-1}\frac{\partial C}{\partial \theta}C^{-1}\mathbf{y}, \tag{2.47}$$

where $C = |K + \sigma^2 I|$. Interestingly, we can see that this derivative has two terms that "balance" each other out [101]. The first term penalizes model complexity, and the second terms rewards data fitting, resulting in a solution that naturally avoids overfitting (the Occam's Razor principle [77]). The derivative over the prior is usually ignored in calculations, however this could generate solutions that are obviously wrong, since the model is now ignoring our assumptions about the underlying function. Since the hyperparameters are usually constrained to be positive, [38] proposes the use of the Gamma (Eq. 2.48) and Inverse Gamma (Eq. 2.49) distributions as possible distributions for the prior:

$$Ga(\theta|m,\alpha) = \frac{(\alpha/2m)^{\alpha/2}}{\Gamma(\alpha/2)}\theta^{\alpha/2-1}\exp(-\alpha\theta/2m) \tag{2.48}$$

$$Ig(\theta|m,\alpha) = \frac{(\alpha m/2)^{\alpha/2+1}}{\Gamma(\alpha/2+1)}\theta^{(-\alpha/2+2)}\exp(-\alpha m/2\theta), \tag{2.49}$$

where m is the mean and α controls the shape of the distribution (larger values create vaguer priors). From this set of equations it is possible to find θ_{EM} using standard gradient optimization techniques, such as conjugate gradients. However, this leads into the first major disadvantage of the Evidence Maximization approach: the posterior distribution over θ may be non-convex (Fig. 2.8), and gradient optimization

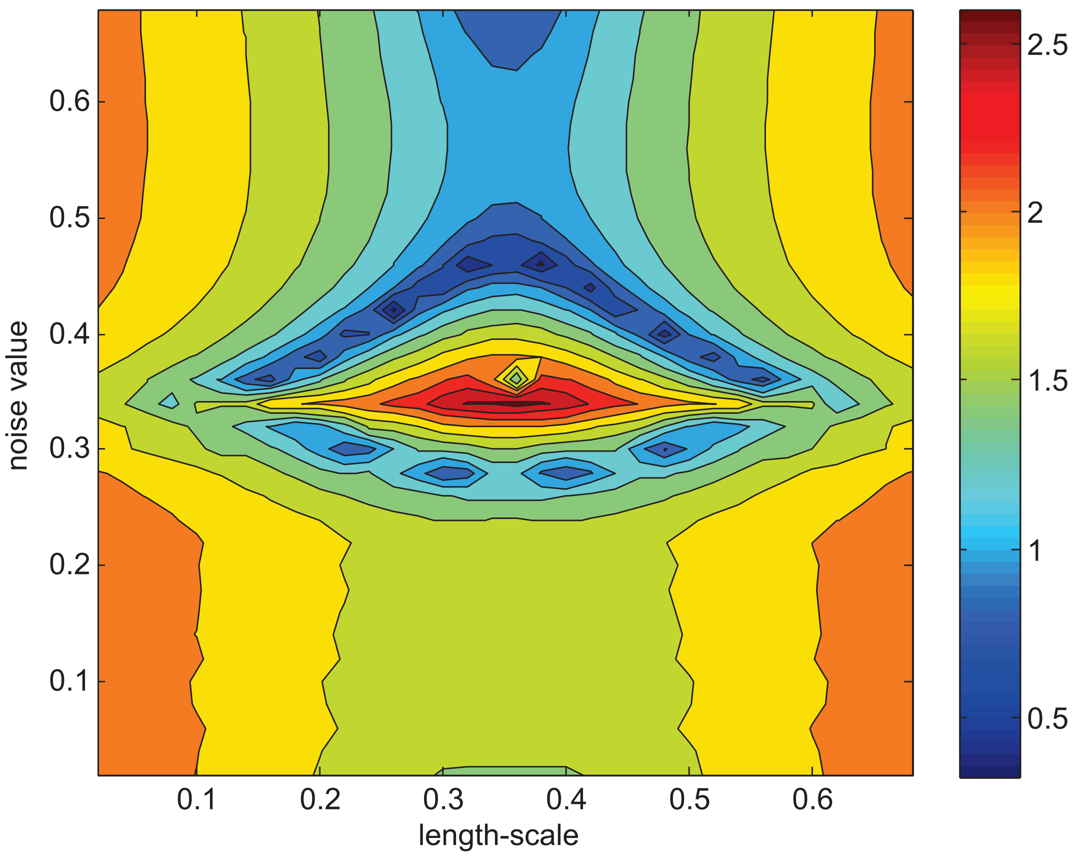


Figure 2.8 – Example of multi-modal log-likelihood for a non-stationary covariance function (neural network). The x-axis represents different values for the length-scale hyperparameter, and the y-axis represents different values for the noise value.

is only able to find local solutions. Therefore, different initial conditions could lead to different results, and multiple random starting points should be used to avoid the selection of a bad local maximum. The second major disadvantage of this approach, and perhaps the most important, is that each evaluation of the gradient requires the evaluation of C^{-1}, generating a computational cost of $\mathcal{O}(N^3)$ that becomes very time consuming for large training datasets.

Cross-Validation

Another method for hyperparameter selection is the *Leave-One-Out Cross-Validation* (LOOCV), which attempts to learn a model that generalizes well to an observation that was withheld from the training dataset (hence the name). A measure of error is chosen to estimate the disparity between estimated and true values, and the process is then repeated for each data point, with their respective errors being combined to produce a final score for the model. A common measure for error is the log-likelihood itself, which for this particular case is of the form:

$$\ln \mathcal{L}(y_i|X, \mathbf{y}_{-i}, \theta) = -\frac{1}{2}\ln(\nu_i^2) - \frac{(y_i - \mu_i)^2}{2\nu_i^2} - \frac{1}{2}\ln(2\pi), \tag{2.50}$$

where μ and ν are the respective mean and variance of $\mathbf{x}_i$ and **-i** is the output vector excluding the i^{th} observation. Once all individual log-likelihoods are calculated, they are added to determine the likelihood of the entire dataset:

$$\mathcal{L}_{LOO}(X, \mathbf{y}, \theta) = \sum_{i=1}^{N} \ln \mathcal{L}(y_i|X, \mathbf{y}_{-i}, \theta). \tag{2.51}$$

In practice, there is often little difference between the optimal hyperparameters produced using the LOO and EM approaches in most datasets. It is argued in [135] that EM optimizes better under the correct assumptions of the model, whereas LOO is independent of those and therefore more robust to model specification errors.

Monte Carlo Approach

The Monte Carlo approach uses sampling methods to calculate an approximation to the predictive distribution. By approximating the integral in Eq. 2.44 using a Markov chain, we get:

$$p(y_*|\mathbf{x}_*, X, \mathbf{y}) \simeq \frac{1}{T}\sum_{t=1}^{T} p(y_*|\mathbf{x}_*, X, \theta_t), \tag{2.52}$$

where θ_t are samples drawn from $p(\theta|X)$, the posterior distribution over θ. Each term on the above equation is a Gaussian, and so the Monte Carlo approximation to the predictive distribution is a mixture of Gaussians, with an accuracy that increases as more samples are drawn. It is important to notice that, since we are sampling from the posterior over θ, we need priors on the hyperparameters $p(\theta)$ as stated in the EM approach. The method used to sample from the posterior distribution will heavily influence the efficiency of this technique, because we need samples that represent the underlying distribution from which they are being taken. If a particular region of θ with a high associated probability is not sampled correctly, then the final approximation to Eq. 2.44 will be poor.

2.2.6 Multiple-Output Gaussian Processes

So far, the Gaussian process regression models considered only deal with one single scalar output y_n mapped to each input vector $\mathbf{x}_n$. Attempts to handle multiple outputs generally involve the use of an independent model for each output, in a method commonly referred to as *multi-kriging* [101], but such models are unable to capture the covariance between outputs. Fig. 2.9 shows an example of two dependent outputs in which one output is simply a shifted version of the other. If both outputs are treated independently, each estimate cannot exploit their obvious similarity to improve results. Intuitively, a proper model should be able to use information from output 1 to improve predictions on output 2 and vice-versa.

Joint predictions are possible, using methods such as *co-kriging* [20], however they are problematic in the sense that it is not clear how covariance functions should be defined [39]. Although several different positive-definite auto-covariance functions (see Section 2.2.4) have already been proposed, it is difficult to define cross-covariance functions that result in positive-definite covariance matrices. For example, consider the following covariance matrix between two Gaussian processes $g_1(x)$ and $g_2(x)$:

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}. \tag{2.53}$$

The definition of positive-definite auto-covariance functions to build the blocks K_{11} and K_{22} is straightforward, however it is not clear how to specify the cross-covariance functions that compose K_{12} and K_{21} such that K is still positive-definite. Returning to Eq. 2.29, a 2×2 block-matrix $\mathbf{K}$ is a positive-definite matrix if and only if $\mathbf{z}^T\mathbf{K}\mathbf{z} > 0$

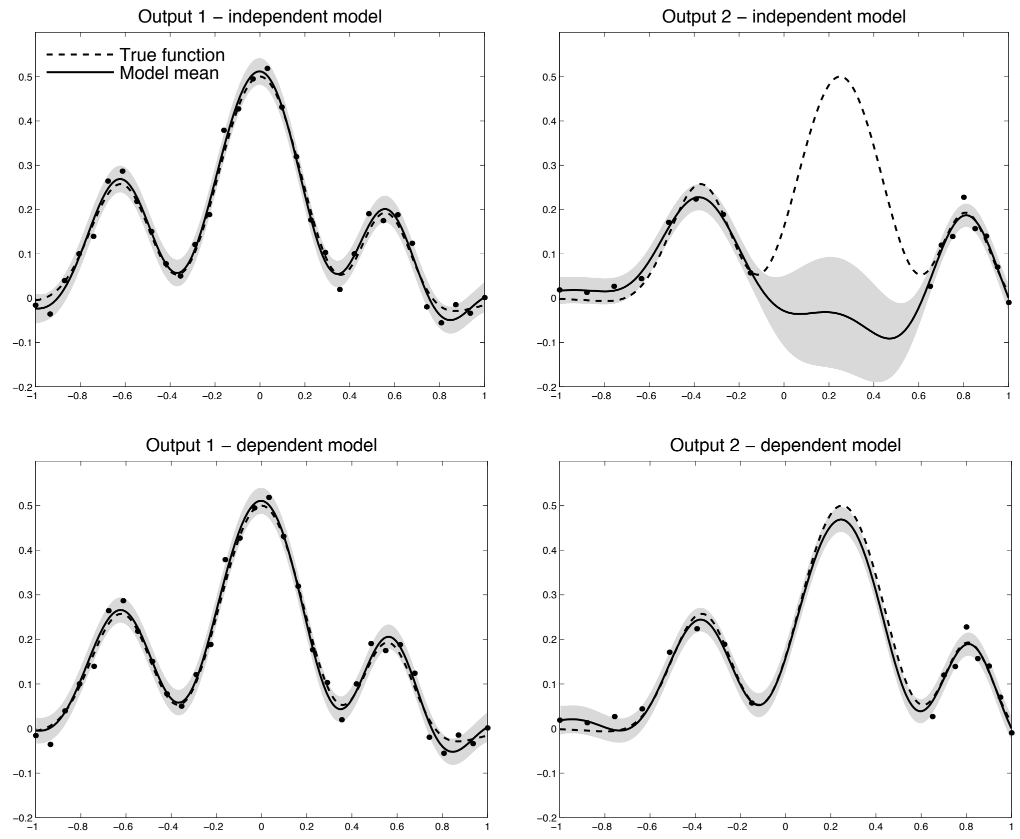


Figure 2.9 – Example [10] of dependent outputs in which output 2 is a translated version of output 1, with independent Gaussian noise of variance 0.025. The solid lines represent the model, the dotted lines are the true function, and the dots are samples. The shaded regions represent 1σ error bars for the model prediction. *(top)* Independent model of the two outputs. *(bottom)* Dependent model.

for any non-zero vector $\mathbf{z}^T = [\mathbf{z}_1^T \mathbf{z}_2^T]$. So, if K_{11} and K_{22} are positive-definite, then for $\mathbf{K}$ to be positive-definite it must satisfy the following condition [38]:

$$\mathbf{z}^T \mathbf{K} \mathbf{z} > 0$$

$$\begin{aligned} \mathbf{z}_1^T \mathbf{K}_{11} \mathbf{z}_1 + \mathbf{z}_1^T \mathbf{K}_{12} \mathbf{z}_2 + \mathbf{z}_2^T \mathbf{K}_{21} \mathbf{z}_1 + \mathbf{z}_2^T \mathbf{K}_{22} \mathbf{z}_2 > 0 \\ \mathbf{z}_1^T \mathbf{K}_{12} \mathbf{z}_2 > -\frac{1}{2} \left\{ \mathbf{z}_1^T \mathbf{K}_{11} \mathbf{z}_1 + \mathbf{z}_2^T \mathbf{K}_{22} \mathbf{z}_2 \right\}. \end{aligned} \tag{2.54}$$

The cross-covariance matrix K_{12} is built from a cross-covariance function $k_{12}(.,.)$ by setting the $(i,j)^{th}$ matrix element equal to $k(\mathbf{x}_{1,i}, \mathbf{x}_{2,j})$, where $\mathbf{x}_{1,i}$ is the i^{th} training input for output 1 and $\mathbf{x}_{2,j}$ is the j^{th} training input for output 2. It is not clear how to define a non-zero $k_{12}(.,.)$ such that Eq. 2.54 holds true, as simply setting it to some positive-definite function will not always work. One possible way of constructing Gaussian processes capable of modelling multiple outputs is to use kernel convolutions, as described in the works of Boyle and Frean [10, 11]. Under this framework, a Gaussian process $V(\mathbf{x})$ is obtained by convolving a continuous white noise process $X(\mathbf{x})$ with a smoothing kernel $h(\mathbf{x})$:

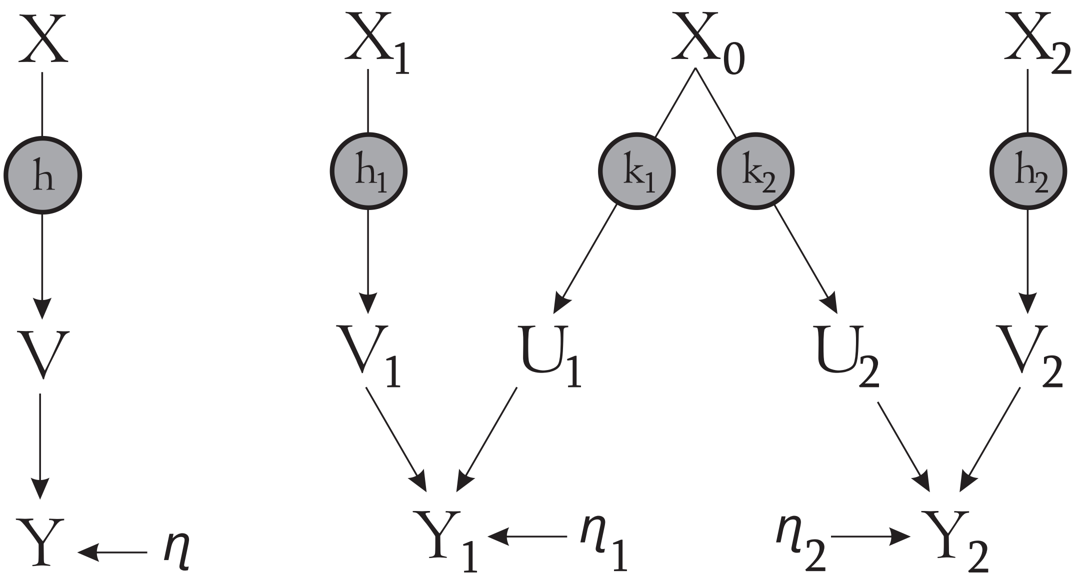
$$V(\mathbf{x}) = h(\mathbf{x}) \star X(\mathbf{x}). \tag{2.55}$$

A second white noise source, representing measurement uncertainty, can be added to the resulting Gaussian process (Fig. 2.10a). This convolution approach has been used to formulate flexible non-stationary covariance functions [97], and can also be extended to address cross-covariance functions. The remainder of this section presents and discusses a technique for multiple-output Gaussian process regression using linear filters as function generations, as described in [10]. Initially the particular case of two dependent outputs is considered, and later it is extended to address multiple outputs.

Two Outputs

For the simpler case of two outputs, we assume N_1 observations of the first output and N_2 observations of the second output, generating the training datasets $\Lambda_1 = \{\mathbf{x}_{1,n}, y_{1,n}\}_{n=1}^{N_1}$ and $\Lambda_2 = \{\mathbf{x}_{2,n}, y_{2,n}\}_{n=1}^{N_2}$. The combined training dataset is defined as $\Lambda = \{\Lambda_1, \Lambda_2\}$, and the goal is to learn a model that predicts both $y_{1,*}$ and $y_{2,*}$ from a test point $\mathbf{x}_*$. As shown in Fig. 2.10b, each output $Y_{i=\{1,2\}}$ can be modelled as a linear sum of three stationary Gaussian processes. One of these (V_i) arises from a noise source X_i unique to that output, convolved with a kernel h_i. The second (U_i) is similar, but arises from a separate noise source X_0 that influences both outputs, although via different kernels k_1 and k_2. The third is additive noise η_i as before, with a variance of σ_i.

Thus, we have $Y_i(\mathbf{x}) = U_i(\mathbf{x}) + V_i(\mathbf{x}) + \eta_i(\mathbf{x})$, and the intermediate processes are



(a) **(b)**

Figure 2.10 – (a) Gaussian process prior for a single output Y. (b) Gaussian process prior for two outputs, Y_1 and Y_2. For the special case in which X_0 is forced to be zero, Y_1 and Y_2 become independent processes as shown in (a).

defined as $U_i(\mathbf{x}) = k_i(\mathbf{x}) \star X_0(\mathbf{x})$ and $V_i(\mathbf{x}) = h_i(\mathbf{x}) \star X_i(\mathbf{x})$. For the specific purposes of this example, the kernels k_i and h_i are squared exponential covariance functions:

$$k_1(\mathbf{x}) = v_1 \exp\left(-\frac{1}{2}\mathbf{x}^T A_1 \mathbf{x}\right) \tag{2.56}$$

$$k_2(\mathbf{x}) = v_2 \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T A_2 (\mathbf{x}-\boldsymbol{\mu})\right) \tag{2.57}$$

$$h_i(\mathbf{x}) = w_i \exp\left(-\frac{1}{2}\mathbf{x}^T B_i \mathbf{x}\right), \tag{2.58}$$

where A_i and B_i are positive-definite length-scale matrices. Note that in Eq. 2.57 the vector $\mathbf{x}$ is offset from zero by $\boldsymbol{\mu}$ to allow modelling of outputs that are coupled and translated relative to one another. The next step is to calculate the functions cov_{ij} that define the auto-covariance ($i = j$) and cross-covariance ($i \neq j$) between any two given inputs $\mathbf{x}_a$ and $\mathbf{x}_b$, separated by a distance vector $\mathbf{d} = \mathbf{x}_a - \mathbf{x}_b$. By performing a convolution integral, the function $cov_{ij}^Y(\mathbf{d})$ can be expressed in closed form and is fully determined by the parameters of the squared exponential covariance functions and the noise variances σ_i^2:

$$cov_{12}^Y(\mathbf{d}) = cov_{11}^U(\mathbf{d}) + cov_{11}^V(\mathbf{d}) + \delta_{ab}\sigma_1^2 \tag{2.59}$$

$$cov_{12}^Y(\mathbf{d}) = cov_{12}^U(\mathbf{d}) \quad , \quad cov_{21}^Y(\mathbf{d}) = cov_{21}^U(\mathbf{d}) \tag{2.60}$$

$$cov_{22}^Y(\mathbf{d}) = cov_{22}^U(\mathbf{d}) + cov_{22}^V(\mathbf{d}) + \delta_{ab}\sigma_2^2, \tag{2.61}$$

where

$$cov_{ii}^U(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} v_i^2}{\sqrt{|A_i|}} \exp\left(-\frac{1}{4}\mathbf{d}^T A_i \mathbf{d}\right) \tag{2.62}$$

$$cov_{12}^U(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} v_1 v_2}{\sqrt{|A_1 + A_2|}} \exp\left(-\frac{1}{2}(\mathbf{d} - \boldsymbol{\mu})^T A_{12} (\mathbf{d} - \boldsymbol{\mu})\right) \tag{2.63}$$

$$cov_{21}^U(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} v_1 v_2}{\sqrt{|A_1 + A_2|}} \exp\left(-\frac{1}{2}(\mathbf{d} - \boldsymbol{\mu})^T A_{21} (\mathbf{d} - \boldsymbol{\mu})\right) = cov_{12}^U(\text{-}\mathbf{d}) \tag{2.64}$$

$$cov_{ii}^V(\mathbf{d}) = \frac{\pi^{\frac{D}{2}} w_i^2}{\sqrt{|A_i|}} \exp\left(-\frac{1}{4}\mathbf{d}^T A_i \mathbf{d}\right), \tag{2.65}$$

with $A_{12} = A_{21} = A_1(A_1 + A_2)^{-1}A_2 = A_2(A_1 + A_2)^{-1}A_1$. Given the formulations for $cov_{ij}^Y(\mathbf{d})$, it is possible to construct the covariance matrices K_{ij} that together compose Eq. 2.53 as follows:

$$K_{ij} = \begin{bmatrix} cov_{ij}^Y(\mathbf{x}_{i,1} - \mathbf{x}_{j,1}) & \dots & cov_{ij}^Y(\mathbf{x}_{i,1} - \mathbf{x}_{j,N_j}) \\ \vdots & \ddots & \vdots \\ cov_{ij}^Y(\mathbf{x}_{i,N_i} - \mathbf{x}_{j,1}) & \dots & cov_{ij}^Y(\mathbf{x}_{i,N_i} - \mathbf{x}_{j,N_j}) \end{bmatrix}. \tag{2.66}$$

The hyperparameter set $\theta = \{v_1, v_2, w_1, w_2, A_1, A_2, B_1, B_2, \boldsymbol{\mu}, \sigma_1, \sigma_2\}$ completely parametrize the above equations. The log-likelihood can now be calculated as:

$$\mathcal{L}(\mathbf{y}|\Lambda, \theta) = -\frac{1}{2}\ln|K| - \frac{1}{2}\mathbf{y}^T K^{-1}\mathbf{y} - \frac{N_1 + N_2}{2}\ln(2\pi), \tag{2.67}$$

where $\mathbf{y} = [y_{1,1}, \ldots, y_{1,N_1} y_{2,1}, \ldots, y_{2,N_2}]^T$. Learning a model now becomes a matter of maximizing the log-likelihood $\mathcal{L}(\mathbf{y}|\Lambda, \theta)$, which is equivalent to the posterior probability. The predictive distribution for a test point $\mathbf{x}_*$ on the i^{th} output is Gaussian with mean $\bar{\mathbf{f}}_*$ and variance $\mathcal{V}(\bar{\mathbf{f}}_*)$ is given by:

$$\bar{\mathbf{f}}_* = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{y} \tag{2.68}$$

$$\mathcal{V}(\bar{\mathbf{f}}_*) = \kappa - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}, \tag{2.69}$$

where $\kappa = cov_{ii}^Y(\mathbf{0}) = v_i^2 + w_1^2 + \sigma_i^2$ and

$$\mathbf{k} = \begin{bmatrix} cov_{i,1}^Y(\mathbf{x}_* - \mathbf{x}_{1,1}) \\ \vdots \\ cov_{i,1}^Y(\mathbf{x}_* - \mathbf{x}_{1,N_1}) \\ cov_{i,2}^Y(\mathbf{x}_* - \mathbf{x}_{2,1}) \\ \vdots \\ cov_{i,2}^Y(\mathbf{x}_* - \mathbf{x}_{2,N_2}) \end{bmatrix}. \tag{2.70}$$

Multiple Outputs

The convolution methodology described above can be extended to build models capable of addressing T outputs, each defined over a D-dimensional input space. In general, we assume M independent Gaussian white noise processes X_m, T outputs Y_t and $M \times T$ kernels. The kernel $k_{mt}(\mathbf{x})$ defines the correlation from input m to output t. The auto-covariance ($i = j$) and cross-covariance ($i \neq j$) functions between processes i and j become:

$$cov_{ij}^Y(\mathbf{d}) = \sum_{m=1}^{M} \int k_{mi}(\mathbf{x}) k_{mj}(\mathbf{x} + \mathbf{d}) d\mathbf{x}, \tag{2.71}$$

and the block matrix $\mathbf{K}$ defined in Eq. 2.53 becomes:

$$\mathbf{K} = \begin{bmatrix} K_{11} & \dots & K_{1T} \\ \vdots & \ddots & \vdots \\ K_{T1} & \dots & K_{TT} \end{bmatrix}. \tag{2.72}$$

The combined training dataset is now $\Lambda = \{\Lambda_1, \dots, \Lambda_T\}$, composed of T individual training datasets $\Lambda_t = \{(\mathbf{x}_{t,1}, y_{t,1}), \dots, (\mathbf{x}_{t,N}, y_{t,N_t})\}$, each containing N_t observations. The total number of observations is therefore $N = \sum_{t=1}^{T} N_t$, and the log-likelihood function becomes:

$$\mathcal{L}(\mathbf{y}|\Lambda, \theta) = -\frac{1}{2}\ln|K| - \frac{1}{2}\mathbf{y}^T K^{-1}\mathbf{y} - \frac{N}{2}\ln(2\pi), \tag{2.73}$$

where $\mathbf{y} = [(y_{1,1} \dots y_{1,N_1}) \dots (y_{t,1} \dots y_{t,N_t}) \dots (y_{T,1} \dots y_{T,N_T})]^T$. Finally, the predictive distribution for a test point $\mathbf{x}_*$ on the i^{th} output is Gaussian with mean $\bar{\mathbf{f}}_*$ and variance $\mathcal{V}(\bar{\mathbf{f}}_*)$:

$$\bar{\mathbf{f}}_* = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{y} \tag{2.74}$$

$$\mathcal{V}(\bar{\mathbf{f}}_*) = \kappa - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}, \tag{2.75}$$

where $\kappa = cov_{ii}^Y(\mathbf{0}) = v_i^2 + w_1^2 + \sigma_i^2$ and

$$\mathbf{k} = [\mathbf{k}_1^T, \dots, \mathbf{k}_j^T, \dots, \mathbf{k}_T^T] \tag{2.76}$$

$$\mathbf{k}_j^T = [cov_{ij}^Y(\mathbf{x}_* - \mathbf{x}_{j1}), \dots, cov_{ij}^Y(\mathbf{x}_* - \mathbf{x}_{jN_j})]. \tag{2.77}$$

Under this new framework, both computational time and memory requirements are increased by a factor of t (the number of outputs), becoming respectively $\mathcal{O}(n^3t^3)$ and $\mathcal{O}(n^2t^2)$. This is due to the now block covariance matrix K, that has to simultaneously maintain all available training data for each individual output, in order to correctly model their correlations during the learning and inference stages. This greatly decreases the number of points the GP framework can maintain for each specific task, making sparse approximations even more attractive.

2.3 Computer Vision for Motion Estimation

Computer vision is a field that includes methods for acquiring, processing, analysing and understanding images in order to produce numerical or symbolic information. A common goal in this field is to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentanglement of symbolic information encoded on its data, using models constructed with the aid of geometry, physics, statistics and learning theory. Typical applications of computer vision in robotics include: object detection/recognition, content-based image retrieval, pose estimation, optical character recognition, motion analysis (egomotion, tracking and optical flow estimation), scene reconstruction and image restoration.

The design of a computer vision algorithm is highly dependent on the task at hand. Some systems are stand-alone applications that aim to solve a specific measurement or detection problem, while others constitute a sub-system of a larger design. The specific implementation of a computer vision algorithm also depends on its functionality, if it is pre-determined or if some part of it can be learned or modified during operation. There are, however, some typical functions that are considered part of virtually every computer vision application:

- **Image Acquisition.** A digital image is produced by the use of one or multiple *image sensors*, that include, other than the typical light-sensitive cameras, range sensors, tomography devices, radar, ultra-sonic cameras, etc. Depending on the type of sensor, the resulting image may be an ordinary 2D matrix, a 3D volume or a sequence of frames. The pixel values usually correspond to light intensity in one or several spectral bands (monochromatic/colour), but may also be related to various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.
- **Pre-Processing.** Before a computer vision method is applied, it is usually necessary to process the raw data in order to assure that it satisfies any particular set of assumptions implied. Examples of pre-processing techniques include

re-sampling (to assure that the image coordinate system is correct), noise reduction (to minimize the amount of false information provided by the sensor), undistortion (especially if the images are used for measurement), contrast enhancement (to assure the detection of relevant information) and scale-space (to enhance image structures at specific scales).

- **Feature Extraction.** Image features at various levels of complexity are extracted from the pre-processed data. These features are obtained either on a pixel level, by maximizing a certain potential function that allows the detection of structures such as corners, edges or ridges, or as a sub-portion of the image (blob detection), using differential or local extrema methods.

- **Detection/Segmentation.** At some point during the feature extraction process a decision is made in regards to which image points or regions are relevant for further processing. This step is highly dependent on the task at hand, and usually falls in one of these two categories: detection of a specific set of interest points, or segmentation of one or multiple image regions which contain a specific object of interest.

- **High Level Processing.** At this point all information deemed available from the image has already been obtained, and it can be applied to a wide variety of regression/classification/validation algorithms. Most of these algorithms fall into one of these categories: estimation of a specific set of parameters (regression), clustering a specific set of features into different categories (classification) and verification that the data satisfies a specific model-based assumption (validation).

- **Decision Making.** After all available data has been obtained and processed, the computer vision algorithm makes a final decision that is passed on to the main system.

In the particular case of visual odometry, the challenge is to obtain camera motion from visual information, more specifically optical flow information extracted from a pair of frames. The remainder of this section is dedicated to the various steps necessary to perform this transformation from a purely geometric stand-point. This framework is extended in the later chapters to include a non-parametric portion, creating the semi-parametric solution to visual odometry that comprises the main contribution of this thesis. It starts by providing an overview of feature extraction and matching techniques, along with methods for outlier removal and optical flow parametrization, both sparse and dense. Afterwards, it introduces the camera model used in the experiments and describes the projective equations used to estimate camera motion based on optical flow information.

2.3.1 Feature Extraction

Feature extraction can be considered a form of dimensionality reduction. When the input data for an algorithm is too large to be processed, and possibly highly redundant, one possible way to shrink the available data to a manageable size without losing important information is by transforming it into a particular set of *features*, also known as feature vector. The act of transforming raw data into a feature vector is called *feature extraction*. If the features extracted are carefully chosen, it is expected that they will encode the relevant information necessary to perform the desired task.

In the context of visual information, features usually represent portions of the image that contain a particular set of characteristics that are robust to a wide variety of transformations (i.e. translation, rotation, scale, affine). This repeatability is important for the later stage of *feature matching*, in which features are compared to determine whether or not they are representative of the same environment structure. This section provides an overview of two different methods for local feature extraction: the *Shi-Tomasi* (ST) corner detection and the *Scale-Invariance Feature Transform* (SIFT) algorithm.

Shi-Tomasi Corner Detection

A corner can be defined as the intersection of two edges, or as a point for which there are two dominant and different edge directions in its local neighbourhood. One of the earliest attempts at corner detection can be found in [86], where each pixel in the image is tested to see if it is a corner by considering how similar a patch centered on the pixel is to nearby, largely overlapping patches. One of the main problems of this approach is that it is not isotropic, in the sense that if an edge is not in the direction of its neighbours it will not be considered as a potential feature. This approach was later improved by Stephens and Harris [46], by considering the differential of the corner score with respect to direction directly, instead of using shifted patches. Considering (without loss of generality) a greyscale 2-dimensional image I, with a patch over the area (u, v) that is shifted by (x, y), the weighted *sum of squared differences* (SSD) between these two patches, denoted as S, is given by:

$$S(x,y) = \sum_u \sum_v w(u,v) \left(I(u+x, v+y) - I(u,v) \right)^2 . \tag{2.78}$$

The term $I(u + x, v + y)$ can be approximated by a Taylor expansion. Let I_x and I_y be the partial derivatives of I, such that:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y. \tag{2.79}$$

This produces the approximation:

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2, \tag{2.80}$$

which can be written in the following matrix form:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}, \tag{2.81}$$

where A is the structure tensor:

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}. \tag{2.82}$$

This matrix is a *Harris* matrix, and the angle brackets denote averaging (i.e. a summation over (u, v)). If a circular window (or a circular weighted window, such as a Gaussian) is used, then the response will be isotropic. The covariance matrix for the corner position is A^{-1}, calculated as follows:

$$A^{-1} = \frac{1}{\langle I_x^2 \rangle \langle I_y^2 \rangle - \langle I_x^2 I_y^2 \rangle} \begin{bmatrix} \langle I_y^2 \rangle & -\langle I_x I_y \rangle \\ -\langle I_x I_y \rangle & \langle I_x^2 \rangle \end{bmatrix}. \tag{2.83}$$

A corner is characterized by a large variation of S in all directions of the vector (x, y). By analysing the eigenvalues λ_1 and λ_2 of A, this characterization can be expressed in the following way: A should have at least one "large" eigenvalue to be a feature. Based on the magnitudes of eigenvalues, the following inferences can be made:

- If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then the pixel (x, y) has no features of interest.
- If $\lambda_1 \approx 0$ and λ_2 has some large positive value, or vice-versa, then the pixel (x, y) is considered an edge.
- If λ_2 and λ_1 have both some large positive value, then the pixel (x, y) is considered a corner.

Because the exact calculation of the eigenvalues is computationally expensive, requiring the computation of a square root, it is suggested that the following function M_c be used, where κ is a tunable sensitivity parameter:

$$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(A) - \kappa \ \text{trace}^2(A). \tag{2.84}$$

A different method for calculating M_c is proposed by Shi and Tomasi in [117], in which $M_c = min(\lambda_1, \lambda_2)$. It has been shown that this method produces features that are more stable for tracking. A corner detector algorithm that uses this method is commonly referred to as the *Shi-Tomasi* corner detector.

Scale-Invariant Feature Transform

SIFT features, as proposed in [68], possess several invariant properties that make them especially attractive in a wide range of robotic applications. SIFT features are invariant to translation, scale and rotation, partially invariant to illumination changes and robust to local geometric distortion. These properties, combined with the high-dimensional space in which they are projected during the descriptor generating process, ensures robustness in tasks such as robot localization and mapping [112], image stitching [13] and 3D scene recognition and tracking [40], although computational complexity is an issue. Over the years several techniques have been proposed as an attempt to generalize and/or improve the applicability of SIFT features in different situations, such as RIFT [63], G-RIF [59], SURF [7] and PCA-SIFT [55].

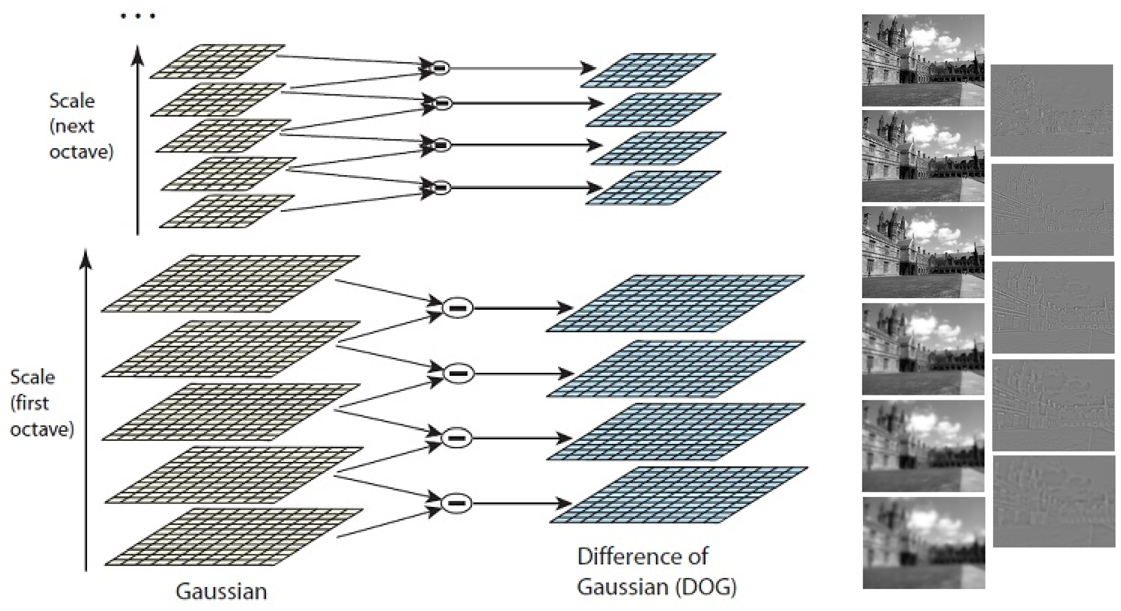
Within the SIFT framework, features are obtained by localizing the maximum and minimum values of a *Difference-of-Gaussian* (DoG) function applied in the scale-space to a series of smoothed and resampled images. The use of different scales ensures the invariant property in regards to changes in the size of the structures observed. Specifically, a DoG image $D(x, y, \sigma)$ is given by:

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma), \tag{2.85}$$

where $L(x, y, k\sigma)$ is the convolution of the original image $I(x, y)$ with the Gaussian blur $G(x, y, k\sigma)$ at a scale $k\sigma$, or:

$$L(x, y, k\sigma) = G(x, y, k\sigma) \star I(x, y). \tag{2.86}$$

Therefore, a DoG image between scales $k_i\sigma$ and $k_j\sigma$ is just the difference of the Gaussian-blurred images at such scales. For the scale-space extrema detection in the SIFT algorithm, the image is first convolved with a series of Gaussian blurs at



(a) **(b)**

Figure 2.11 – Image courtesy of [68], depicting the feature selection process. (a) Construction of the scale-space for the SIFT algorithm. (b) Example of DoG images obtained from Gaussian blurs of different $k\sigma$.

different and increasing scales. The convolved images are grouped by octave (an octave represents the doubling of σ), and the value of k_i is selected so that we obtain a fixed number of convolved images per octave. The DoG images are then obtained (Fig. 2.11) by subtracting adjacent Gaussian-blurred images in each octave. To increase speed, at each octave the image is downsampled by half (which is effectively equal to doubling σ), generating a structure that is similar to a pyramid.

Once the DoG images have been obtained, features are identified as the local minima/maxima of the DoG images across scales. This is done by comparing each pixel in the DoG images to its eight neighbours at the same scale and the nine neighbouring pixels in each of the adjacent scales. If the pixel value is a minimum or a maximum in comparison to all these pixels, then it is selected as a potential candidate. This is a variation of one of the blob detection methods developed by Lindeberg [66], where features are obtained by detecting scale-space extrema in the scale normalized Laplacian. The Difference-of-Gaussian operator can be seen as an approximation to the Difference-of-Laplacian, with the implicit normalization in the pyramid also constituting a discrete approximation to the scale normalized Laplacian.

The scale-space extrema detection usually generates a large amount of candidate features, some of which are unstable. The next step in the algorithm consists of a detailed fit to the nearby data for accurate location, scale and ratio of principal curvatures. This process allows the rejection of candidates with low contrast (and therefore sensitive to noise) or that are poorly localized along an edge. The interpolated location of each extremum is done using the quadratic Taylor expansion of the Difference-of-Gaussian scale-space function, $D(x, y, \sigma)$, with the candidate feature as the origin. This Taylor expansion is given by:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}, \tag{2.87}$$

where D and its derivatives are evaluated at the candidate feature location $\mathbf{y} = (u, v)$ location, and $\mathbf{x} = (x, y, \sigma)$ is the offset from this point. The location of the extremum, $\hat{\mathbf{x}}$, is determined by taking the derivative of this function with respect to $\mathbf{x}$ and setting it to zero. If the offset is larger than 0.5 in any dimension, this means that the extremum lies closer to another candidate feature. In this case, the candidate feature is changed and the interpolation is performed instead about that point. To discard features with low contrast, the value of $D(\mathbf{x})$ is computed at the offset $\hat{\mathbf{x}}$, and if this value is less than a given threshold the candidate point is discarded. Otherwise, it is kept with a final location $\mathbf{y} + \hat{\mathbf{x}}$ and scale σ.

The DoG function will also have strong responses along edges, even if the candidate feature is not robust to small amounts of noise. Therefore, features with poorly determined locations but high edge responses should be eliminated. For poorly defined peaks in the DoG function, the principal curvature across the edge should be much larger than the principal curvature along it. Finding these principal curvatures amounts to solving for the eigenvalues of the second order Hessian matrix:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}. \tag{2.88}$$

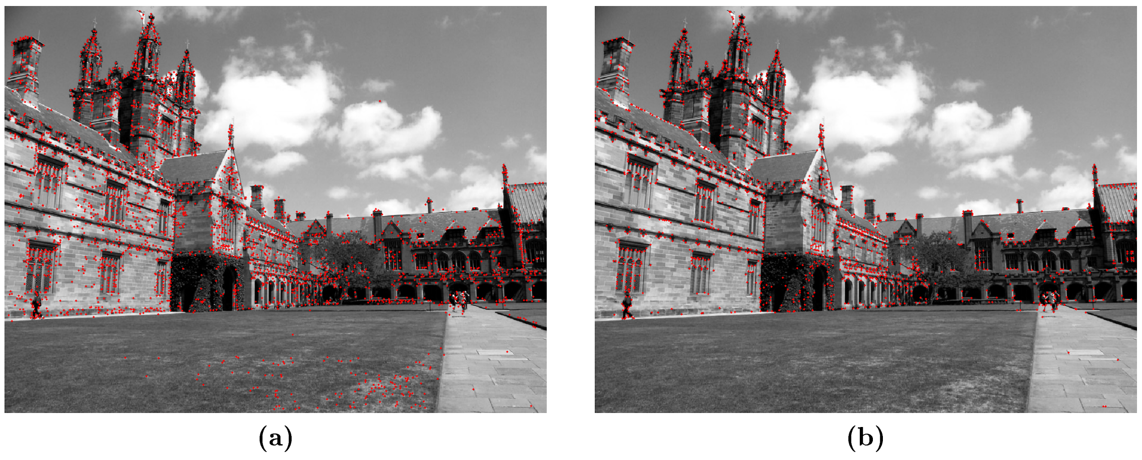


Figure 2.12 – Results using different feature extraction methods. (a) SIFT features. (b) Shi-Tomasi features.

The eigenvalues α and β of H are proportional to the principal curvatures of D, and the ratio of these two eigenvalues is defined as $r = \alpha/\beta$, with $\alpha > \beta$. Fortunately, this ratio can be obtained without calculating the eigenvalues explicitly, using instead the trace $D_{xx} + D_{yy}$ and the determinant $D_{xx}D_{yy} - D_{xy}^2$. The ratio $R = tr(H)^2/det(H)$ is shown to be equal to $(r+1)^2/r$, which depends only on the ratio of eigenvalues instead of their individual values. Therefore, the higher the value of R the higher the absolute difference between α and β will be, and if R is greater than a certain threshold the feature is considered poorly localized and discarded. This technique is a transfer from a corresponding approach in the Harris operator for corner detection, where the Harris matrix (Eq. 2.82) is used instead of the Hessian (Eq. 2.88).

2.3.2 Feature Matching

As stated previously, a good feature should be robust to a series of transformations, to allow for posterior matching with features obtained in different images. The feature matching process is the basis for most computer vision algorithms, where the goal is to detect and recognize structures on new data based on available information. This is done by comparing the *descriptor* of each feature, which is a (usually high-dimensional) vector $\mathbf{d} \in \Re^S$ that represents the information encoded in that portion of the image. A distance metric is defined to quantify the difference between $\mathbf{d}_i$ and $\mathbf{d}_j$ in this S-dimensional space, usually the L_1 (Eq. 2.89) or L_2 (Eq. 2.90) norms, and techniques such as *k-d trees* [85] are used to find the nearest neighbour. If the dimensionality is too high, k-d trees are inefficient and approximate techniques [105] are commonly used for speed purposes.

$$L_1(\mathbf{d}_i, \mathbf{d}_j) = \sum_{s=1}^{S} |d_{i,s} - d_{j,s}| \tag{2.89}$$

$$L_2(\mathbf{d}_i, \mathbf{d}_j) = \sqrt{\sum_{s=1}^{S} (d_{i,s} - d_{j,s})^2} \tag{2.90}$$

Visual descriptors can be divided into two categories [78]: *General Information Descriptors* and *Specific Domain Information Descriptors.* The first category comprises low-level descriptors, which encode raw information such as colour, shape, texture and motion. The second category comprises more semantic descriptors that usually cannot be obtained directly, such as object classification and scene recognition results. We will focus here on the first category, which can be further divided into the following sub-groups [78]:

- **Colour.** The most basic quality of visual information. It can be organized into five different parameters: *Dominant Colour Descriptor* (DCD), *Scalable Colour Descriptor* (SCD), *Colour Structure Descriptor* (CSD), *Colour Layout Descriptor* (CLD) and *Group of Frames* (GoF).

- **Texture.** Another important aspect in image description, texture information characterizes a region's homogeneity and is usually obtained using histograms. It is composed of three different parameters: *Homogeneous Texture Descriptor* (HTD), *Texture Browsing Descriptor* (TBD) and *Edge Histogram Descriptor* (EHD).

- **Shape.** Comprises segmented information similar to that a human would use to recognize objects. Even though current algorithms are still not able to provide such information, there are several methods that are considered a good approximation. It is also composed of three different parameters: *Region-based Shape Descriptor* (RSD), *Contour-based Shape Descriptor* and *3-D Shape Descriptor* (3DSD).

- **Motion.** Describes movement in a video sequence. This movement may be due to the presence of dynamic objects or to the camera's own motion. This descriptor set is composed of four different parameters: *Motion Activity Descriptor* (MAD), *Camera Motion Descriptor* (CMD), *Motion Trajectory Descriptor* (MTD) and *Warping and Parametric Motion Descriptor* (WPMD).

Two different descriptors commonly found in the literature are discussed below: *Colour Patches* and *Gradient Orientation Histogram*, also known as the SIFT descriptor [68]. Initially the most basic version of the Colour Patch descriptor is presented, and then gradually improved until it approaches the SIFT descriptor.

Colour Patch Descriptor

The naive approach to the Colour Patch descriptor consists in placing a $N \times N$ window (Fig. 2.13) centered at the pixel coordinate (u, v). The pixel intensity of each window cell is then stored as the descriptor, generating in the case of a monochromatic image a N^2-dimensional descriptor vector. This method is invariant to translation, but will behave poorly when faced with rotation, blur and illumination changes (or any transformation that changes the colour space). A weight mask can be used to decrease the impact of pixels far from the center point in the calculations, thus increasing the descriptor's robustness to small changes in orientation. To counter illumination changes in the scene, and other transformations in the colour space, the patch can be normalized by enforcing a unit length constraint. Now, global changes in pixel intensity will not have any effect on the final descriptor. If a RGB image is available, the descriptor can be extended to include patches in each colour channel, resulting in a $3N^2$-dimensional vector.

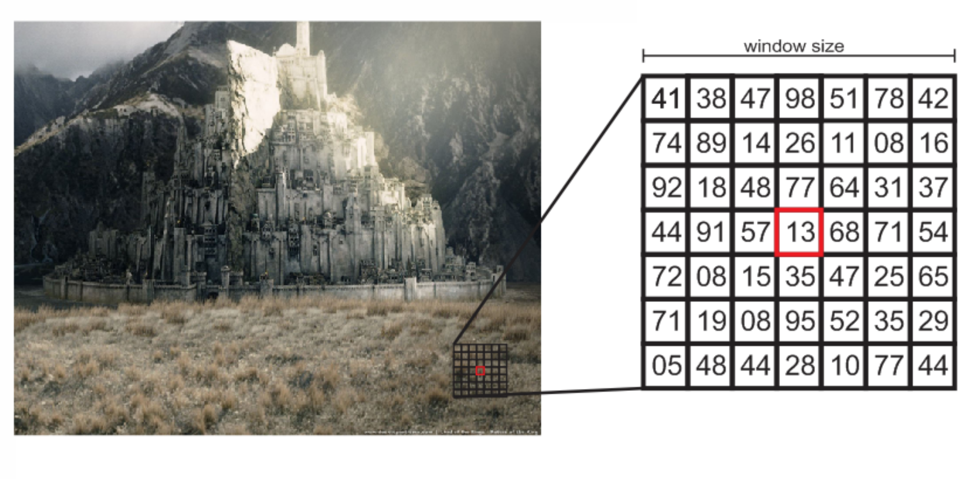


Figure 2.13 – Example of a 7×7 window being used to create a pixel intensity matrix, centered around the red square.

A descriptor vector created directly from a sequential patch will be sensitive to its orientation, and also to minor changes in local pixel intensity. In this scenario, doing a value-by-value comparison of descriptors could lead to poor matching performance, especially if both images are taken from a significantly different vantage point. An alternative approach to directly storing individual pixel information consists in using histograms, which eliminate small positional errors by clustering similar values together. Assuming a histogram with M bins, the resulting descriptor will be a M-dimensional vector. The same normalization process described previously can also be applied in this scenario. If the window size is sufficiently large, multiple histograms may also be applied to different portions of the patch, resulting in a PN-dimensional descriptor vector, where P is the number of histograms.

Finally, instead of pixel intensities, the intensity gradients could be used to encode colour information. The most common method of obtaining these intensity gradients is to apply the 1-D derivative mask $[-1, 0, 1]$ for horizontal gradients and $[-1, 0, 1]^T$ for vertical gradients. Other derivative masks can also be used, such as the 3×3 Sobel masks, however it has been shown [23] that these usually exhibited poorer performance, along with the use of Gaussian smoothing before applying the masks. The colour histograms now become orientation histograms, storing the individual orientation of each pixel according to its neighbours, and the most frequent becomes the *dominant orientation.* By aligning the dominant orientation of different descriptors, it is possible to achieve rotational invariance.

SIFT Descriptor

The SIFT descriptor combines all techniques described previously, resulting in a 128-dimensional vector that is invariant to translation, scale and rotation, partially invariant to illumination changes and robust to local geometric distortion. Assuming a Gaussian-smoothed image $L(x, y, \sigma)$, the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are precomputed using pixel differences:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.91}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right). \tag{2.92}$$

This magnitude and orientation calculation for the gradient is done for every pixel in a neighbouring region around the feature. An orientation histogram of 36 bins is formed, with each bin covering 10 degrees. Each sample in the neighbouring region added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times higher than that of the feature scale. The peaks in this histogram correspond to the dominant orientations. Once the histogram is filled, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peak are assigned to the feature. In the case of multiple orientations being assigned, an additional feature with the same location and scale as the original feature is created for each additional orientation.

Up to this point the orientation assigned ensures invariance to translation, rotation and scale. The next step is to convert this information into a vector that is both highly distinctive and partially invariant to other transformations, such as illumination and affine. First, a set of orientation histograms is created using values of samples in a 16×16 region around the feature, such that each histogram contains samples from a 4×4 subregion of the original neighbourhood region (Fig. 2.14). The magnitudes

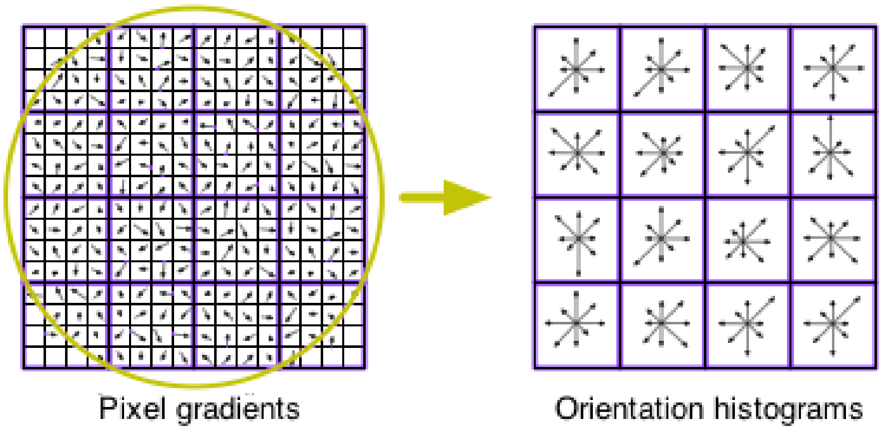


Figure 2.14 – Example of SIFT descriptor. The 16×16 square window around the feature is divided into 4×4 histograms, each one composed of 8 bins (the yellow circle is a Gaussian mask weighting the contribution of each pixel gradient). The resulting descriptor is therefore a 128-dimensional vector.

are further weighted by a Gaussian function with σ equal to half the width of the descriptor window. The descriptor then becomes a vector of all the values of these histograms, and since there are $4 \times 4 = 16$ histograms, each with 8 bins, the vector has 128 elements. This vector is then normalized to a unit length in order to enhance invariance to affine changes in illumination.

Even though 128 dimensions may seem a high number, descriptors with lower dimensionality do not perform as well across the range of matching tasks [69], and

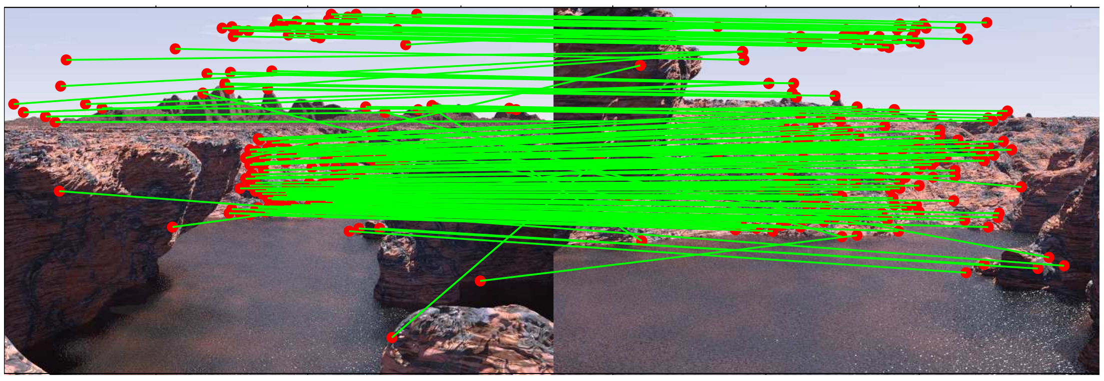


Figure 2.15 – Example of matching using the SIFT algorithm, and the impact of outliers on the final result.

the computational cost remains low by using the Best-Bin-Fit (BBF) approach to find the nearest neighbour. Longer descriptors continue to perform better but not by much, and there is an additional chance of increasing sensitivity to distortion and occlusion. It is also shown that feature matching accuracy is above 50% for viewpoint changes of up to 50 degrees, thus rendering the descriptor partially invariant to affine transformations.

2.3.3 Outlier Removal

Regardless of how robust a descriptor might be, there will always be a chance of obtaining *false matches*, which are instances of matching in which features corresponding to one structure in the image will be falsely correlated to features belonging to another structure. This is caused by a variety of reasons, such as structure similarity, radical changes in viewpoint or luminosity, textureless regions or occlusion. Fig. 2.15 is an example of a matching set obtained from two images of the same structure taken from a different perspective, using the SIFT descriptor and the least-squares metric for distance. It is possible to see that the matching algorithm was able to correctly correlate most of the repeating structures, however there are several errors caused mostly by similar patterns in the sky and rocky terrain.

A false match can be seen as a particular case of an *outlier*, which is an observation that is numerically distant from the rest of the data, deviating significantly from the other members of the sample in which it occurs. Assuming a static environment (as is the case in Fig. 2.15), it is obvious that all motion experienced by the features will be caused by the camera's own translation and rotation between frames. Therefore, there is a constraint that all pairs of matched features must share, and any pair that does not comply to this constraint must be an outlier and should be discarded.

A popular method of outlier detection and removal is the RANSAC (RANdom SAmple Consensus) [35], an iterative algorithm used to estimate the parameters of a mathematical model from a set of observed data which contains outliers (Fig. 2.16). It is a non-deterministic algorithm in the sense that is produces a reasonable result only within a certain probability, with its probability increasing as more iterations are allowed. In broad terms, it draws a random subset from all available observations, from which a hypothetical model is generated, and the remaining observations are then tested against this model, producing a hypothetical inlier set. This process is repeated many times, and the model with the highest number of hypothetical inliers is assumed to be the correct one. A pseudo-algorithm describing the RANSAC steps in more details is shown in Algorithm 2.1. It is possible to avoid the explicit determination of k as a fixed number of iterations for the algorithm, enforcing instead a percentage p that all points used to generate the model are in fact inliers. In this case, the value of k becomes:

$$k = \frac{\log(1-p)}{\log(1-w^n)}, \tag{2.93}$$

where w is the ratio of inliers points to the total number of points (usually obtained from a rough estimation) and n is the number of points necessary to define the model.

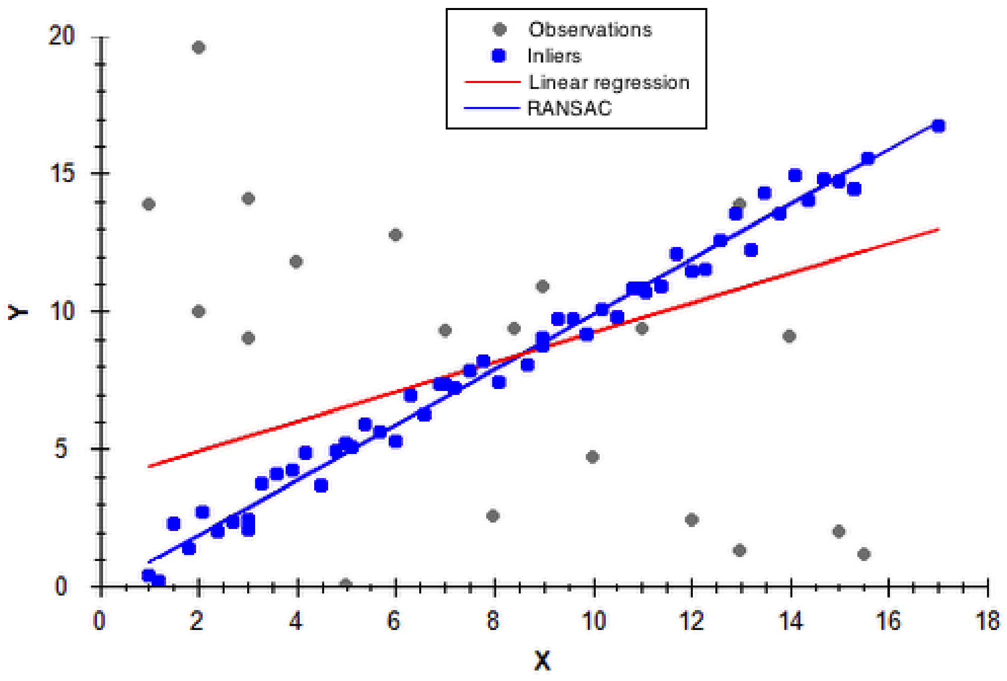


Figure 2.16 – Effect of RANSAC in linear regression. The red line indicates the resulting linear model that fits all observed data, and the blue line indicates the resulting linear model that fits the remaining observations after all outliers have been removed (95% probability).

Algorithm 2.1: The RANSAC algorithm

```
Input :  data        - set of observations
         model       - model used to fit the data
         n           - minimum number of points required to fit the model
         k           - number of iterations performed by the algorithm
         t           - threshold for determining whether a point fits a model
         d           - minimum number of data values required to fit a model
Output:  best_model  - model parameters that best fit the data
         best_set    - inlier set for the best_model
         best_error  - the error of best_model relative to the data
iterations = 0
best_model = best_set = null
best_error = $\infty$
while iterations $< k$ do
    maybe_inliers = n randomly selected points from data
    maybe_model = model parameters fitted to maybe_inliers
    current_set = maybe_inliers
    foreach point not in maybe_inliers do
        if point fits maybe_model with error $< t$ then
            add point to current_set
        end
    end
    if size of current_set $> d$ then
        this_model = model parameters fitted to current_set
        this_error = measure of how well this_model fits current_set
        if this_error $<$ best_error then
            best_model = this_model
            best_set = current_set
            best_error = this_error
        end
    end
    iterations = iterations + 1
end
return best_model,best_set,best_error
```

In the particular case of visual outliers, this model is the camera model itself, which is dependent on the type of camera used. The next section will describe the *pinhole* camera model, which is used during the experiments conducted in this thesis.

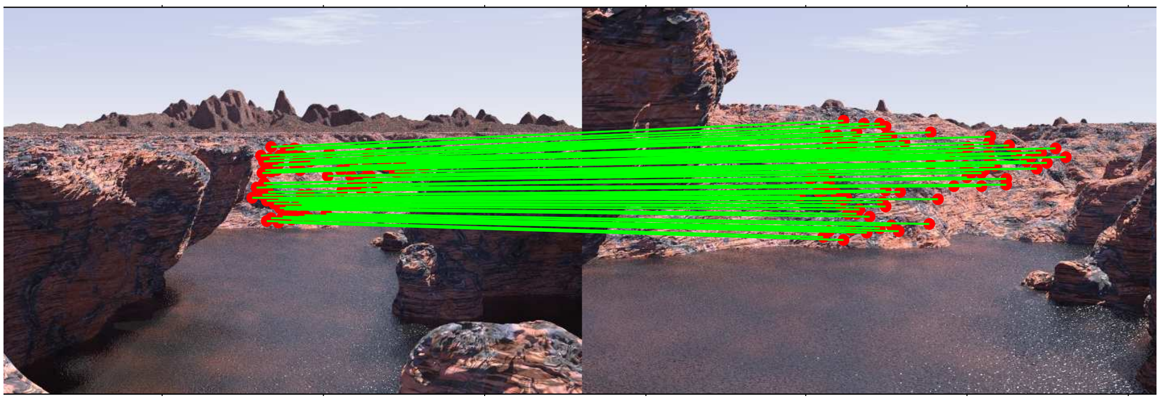


Figure 2.17 – Matching set after the application of RANSAC (compare to Fig. 2.15).

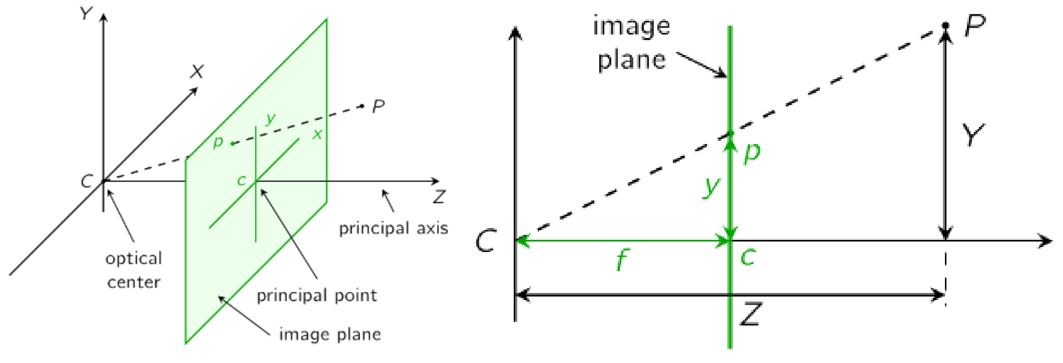
2.3.4 Pinhole Camera Model

The pinhole camera model describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane of an ideal *pinhole camera*, where the camera aperture is described as a point and no lenses are used to focus light. Consider $P = (X, Y, Z)^T \in \Re^3$ to be a point in the tridimensional space (a *world point*) that is mapped to the *feature point* $p = (x, y)^T \in \Re^2$ in the 2D image plane (Fig. 2.18a). Initially, we assume that the image plane is mirrored along the Z-axis in front of the camera between the *optical center* and point P. There are two coordinate systems, the *camera coordinate system* XYZ and the *image coordinate system* xy, and the points P and p are fixed in relation to each of them respectively.

Looking at Fig. 2.18b, we see that the distance between the optical center and the image plane is the *focal length* f. The following relations can then be obtained using the Interception Theorem: $x = fX/Z$ and $y = fY/Z$, which if combined into a vector become:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix}. \tag{2.94}$$

The next step is to project P and p into their projective coordinates, also known as *homogeneous coordinates* [47]. These coordinates have the advantage of allowing the representation of points at infinity using finite values, and formulas involving



(a) (b)

Figure 2.18 – Pinhole Camera model.

homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. The new coordinate values for P and p are now:

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \Rightarrow \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \tilde{P} \qquad p = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{p}. \tag{2.95}$$

Using homogeneous coordinates we can rewrite Eq. 2.94 as such:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}, \tag{2.96}$$

which, if put in matrix form, becomes:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.97}$$

From this basic formulation other camera parameters, other than the focal length, can be added to increase the model's ability to deal with real situations. For example, we can remove the assumption that the origin of the image coordinate system is located in the image's center by adding an offset $\hat{c}_x$ and $\hat{c}_y$ to each axis. Also, in digital images the vertical and horizontal pixel size may be different, and therefore the focal lengths f_x and f_y will also differ in each axis. The angle between the x and y pixel axes may also be defined as α, modelling lens distortions. Taking all these parameters into consideration, Eq. 2.97 becomes:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} k_x f & \alpha & k_x \hat{c}_x & 0 \\ 0 & k_y f & k_y \hat{c}_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \alpha & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \quad (2.98)$$

where k_x and k_y are the number of pixels per unit of length (resolution) in each camera axis. Finally, the pinhole camera model should also consider differences in rotation and translation between the two coordinate systems. This is done by the introduction of a *transformation matrix* T, such that:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{C} \underbrace{\begin{pmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{pmatrix}}_{T} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \quad (2.99)$$

The matrix C is the *calibration matrix*, and governs the camera's *intrinsic parameters*, whereas the transformation matrix T governs the camera's *extrinsic parameters.* The intrinsic parameters are internal to the camera and are obtained using calibration methods, while the extrinsic parameters define the position and orientation of the camera in relation to a global coordinate system. The next section will delve deeper into the problem of estimating the extrinsic parameters of one camera in relation to another, a problem commonly known as the *structure from motion* problem.

2.3.5 Structure from Motion

Structure from motion (SFM) refers to the process of finding the three-dimensional structure of an object by analysing the changes in an observing sensor over time. Basically, it attempts to recover 3D structure from 2D data, in this case the projection of the environment on an image. Conversely, if the same point (assumed static) is observed from a different perspective, it is possible to use this information to recover the sensor distance between images. In the particular case of a single camera, this distance is equivalent to the translation and rotation between frames. The set of equations that governs these homogeneous projections from the 3D space to 2D images, assuming a pinhole camera model, is known as *epipolar geometry* [47].

An example of the epipolar geometry between two cameras is shown in Fig. 2.19, where they are both looking at point **X**. In reality, the image plane is actually behind

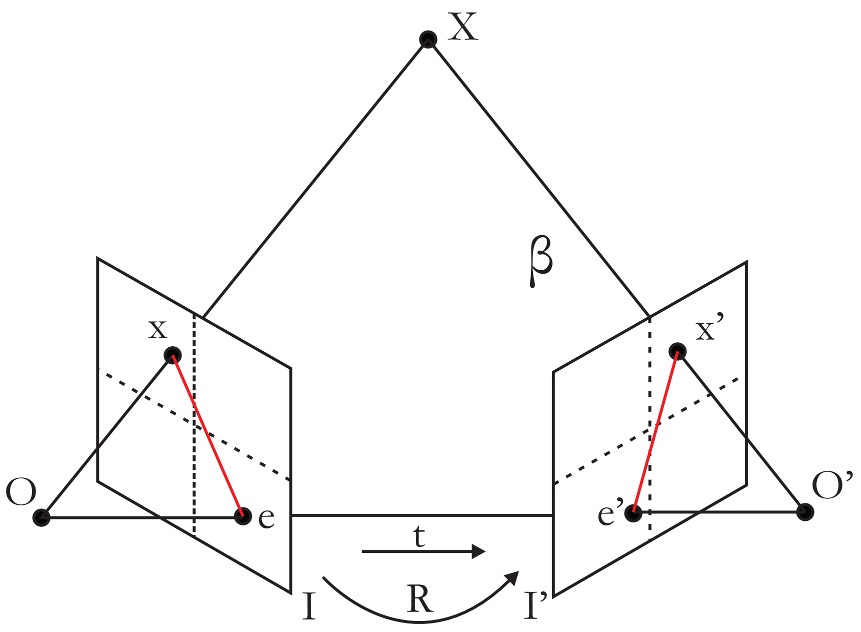


Figure 2.19 – Epipolar geometry between two cameras. The same point P in the 3D space is observed by images I and I', generating the respective projections p and p'. The red lines represent the epipolar lines in each image, with $\mathbf{e}$ and $\mathbf{e}'$ being the epipoles. The transformation between the two cameras is given by the translation vector $\mathbf{t}$ and the rotation matrix R.

the center of projection and produces a rotated image, however here the projection problem is simplified by placing a virtual image plane I in front of the center of projection $\mathbf{O}$ of each camera. The projection of $\mathbf{X}$ in image I creates $\mathbf{x}$, and $\mathbf{e}$ is the *epipolar point*, representing the projection of $\mathbf{O}'$ also in image I. The line $(\mathbf{x} - \mathbf{e})$ is the *epipolar line*, and it represents the projection of $(\mathbf{X} - \mathbf{O}')$ in image I. Because the center of projection $\mathbf{O}$ is fixed for this particular configuration, this means that all epipolar lines in I must intersect on $\mathbf{e}$. In fact, any line which intersects $\mathbf{e}$ is an epipolar line, since it can be derived from some 3D point. The same principles can also be applied to the other camera, generating the image point $\mathbf{x}'$, the epipolar point $\mathbf{e}'$ and the epipolar line representing the projection of $(\mathbf{X}-\mathbf{O})$ in image I'. The points $\mathbf{X}$, $\mathbf{O}$ and $\mathbf{O}'$ form a plane β called the *epipolar plane*, and all epipolar planes and epipolar lines must intersect the epipolar points regardless of where $\mathbf{X}$ is located.

The epipolar constraints created by this particular geometry can be codified in a 3×3 matrix F that describes the correlation between points $\mathbf{x}$ and $\mathbf{x}'$, called the *fundamental matrix*. If $\mathbf{x}$ is observed, then its matching point $\mathbf{x}'$ in the other image must lie in the epipolar line created by $(\mathbf{X} - \mathbf{O})$, as depicted in Fig. 2.20. In other words, F must satisfy:

$$\mathbf{x}'^T F \mathbf{x} = 0. \tag{2.100}$$

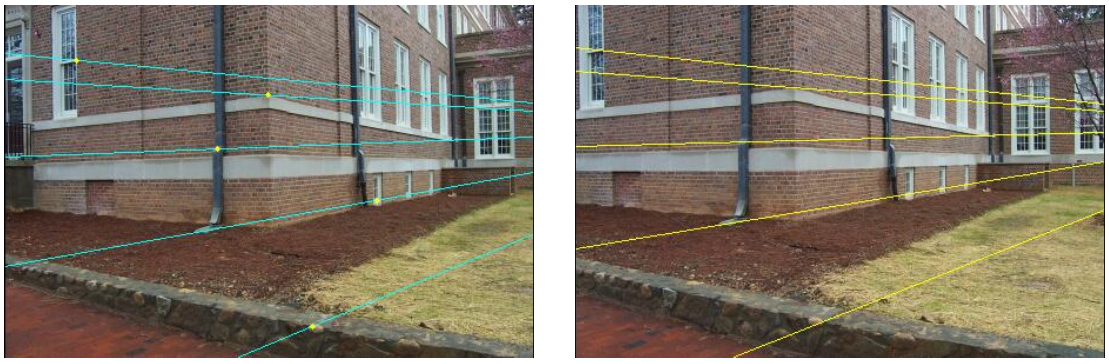

(a) (b)

Figure 2.20 – Example of epipolar lines. For each point in (a) there is an epipolar line defined in (b), and it is possible to see that each of these lines pass on their corresponding points in the other image (these points were selected manually, without a corresponding match).

Being of rank two and determined only up to scale ($\mathbf{x}'$ may lie in any point of the epipolar line defined by $\mathbf{x}$), the fundamental matrix can be estimated given at least seven point correspondences, $\mathbf{x}_n = (x_n, y_n, 1)^T$ and $\mathbf{x}'_n = (x'_n, y'_n, 1)^T$. If eight points are available, the solution can be obtained by solving a linear system (Eq. 2.101), and if more than eight points are available the solution can be obtained using least-squares error. Since the matching set will often contain outliers, techniques such as RANSAC (Algorithm 2.1) are used to elect the most probable hypothesis.

$$\begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0 \tag{2.101}$$

If the calibration matrix C, as defined in Eq. 2.99, is known, the *essential matrix* can be obtained (Eq. 2.102). The essential matrix further explores the correlation between two matching points $\mathbf{x}$ and $\mathbf{x}'$ by using the camera's intrinsic parameters to provide an estimation of relative translation $\mathbf{t}$ (up to a scale factor) and rotation R between frames. To be an essential matrix a 3×3 matrix must have two singular values, which are equal to each other, and another which is zero. An essential matrix has only five degrees of freedom: three from the translation vector $\mathbf{t}$ and three from the rotation matrix R, but there is an element of scalar ambiguity that must be subtracted from this total, resulting in five degrees of freedom. This means that the essential matrix is an element of a *projective space*, that is, two essential matrices are considered equivalent if one is a non-zero scalar multiplication of the other.

$$E = C'^T F C = R[\mathbf{t}] \tag{2.102}$$

The following method to determine R and $\mathbf{t}$ is derived from [47], and it is based on performing a singular value decomposition (SVD) of the essential matrix E. It is also possible to obtain R and $\mathbf{t}$ directly, without performing SVD, as shown in [67]. The singular value decomposition of E gives rise to:

$$E = U\Sigma V^T, \tag{2.103}$$

where U and V are orthogonal and Σ is a 3×3 diagonal matrix defined as:

$$\Sigma = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.104}$$

The diagonal entries of Σ are the singular values of E, which according to the internal constraints of the essential matrix must consist of two identical values s and one zero value. If E is approximated from available data, this constraint is usually enforced by finding the average of the two highest values and setting the third one to zero. An auxiliary skew-symmetric matrix W is then defined as:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{with} \quad W^{-1} = W^T = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.105}$$

From the matrices defined above four different configurations are possible, depending on the sign of $\mathbf{t}$ (which direction the camera is pointing) and the orientation of R (if the camera rotates clockwise or counter-clockwise). These four possible solutions are:

$$[R|\mathbf{t}]_1 = [UWV^T| - VW\Sigma V^T] \tag{2.106}$$

$$[R|\mathbf{t}]_2 = [UWV^T| + VW\Sigma V^T] \tag{2.107}$$

$$[R|\mathbf{t}]_3 = [UW^TV^T| - VW\Sigma V^T] \tag{2.108}$$

$$[R|\mathbf{t}]_4 = [UW^TV^T| + VW\Sigma V^T]. \tag{2.109}$$

It turns out that only one of these solutions are physically possible (Fig. 2.21), given the camera configuration. Given a pair of corresponding image coordinates, three of the solutions will always produce 3D points that are behind at least one of the cameras, and therefore cannot be seen by it. Only one of the four solutions will consistently produce 3D points that are in front of both cameras, and obviously this will be the right one, albeit an undetermined scaling factor.

The resulting translation and rotation matrices are then used as the motion estimates between frames, providing a full 6 degree of freedom velocity vector. Further processing, such as bundle adjustment [126], could in principle be used to further refine these estimates, however this was not explored in this thesis since the same techniques could also be applied to the proposed machine learning based visual odometry algorithm. The same could be said about any further improvements on this basic geometric model [17, 92, 109], since in Section 4.1.4 it is shown how to incorporate these estimates into the GP framework. Better estimates would only lead to a better starting value for the non-parametric model to improve using training information. In addition, other techniques were tested, such as 1-Point RANSAC [106] and Mono-SLAM [24], as a way to show how state-of-the-art traditional visual odometry algorithms compare to the proposed approach.

It is also important to note that, since this is a monocular configuration, the translation vector is recovered only up to a scale factor, because of the parallax effect

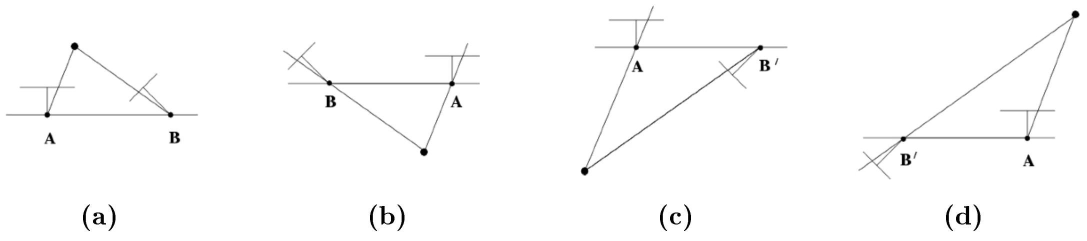


Figure 2.21 – Four possible solutions obtained from the singular value decomposition of the essential matrix. Note that only in (a) the 3D point is in front of both cameras, and therefore is the correct solution. In (b) the direction of the translation vector is reversed, and (c) and (d) are the "twisted pairs", related to each other by a 180^o rotation about the baseline.

(an object could be close and moving slowly or far away and moving fast). Several approaches have been proposed to deal with this limitation, while keeping a monocular configuration [107, 109, 137], however they usually require either an extra sensor capable of recovering absolute distance or very specific circumstances (i.e. a ground planar assumption or pure rotation). Because the focus of this thesis is the development of a visual odometry solution capable of addressing situations in which there is no knowledge of the visual system or environment around the vehicle, using only visual sensors during the inference stage, these techniques were not explored here. The geometric model discussed previously was also able to generate reasonable estimates even in tridimensional unconstrained outdoor navigation (see 4.3.2), a scenario in which most state-of-the-art visual odometry algorithms struggle with, because they are mostly focused on ground applications or low-velocity flights.

2.4 Summary

This chapter explored two important areas of research in robotics: regression, in particular Bayesian regression and Gaussian processes; and computer vision, in particular the structure from motion problem. The literature in both these areas is vast and the material presented here is only meant to provide an overview of the techniques used throughout the remainder of this thesis.

Bayesian inference offers a powerful framework capable of addressing numerous issues present in robotics. It allows the modelling of complex systems within a realistic probabilistic setting, and this has led to their increase in popularity over the years, especially as computational power now makes online Bayesian inference possible. Gaussian process regression has already been responsible for several advancements in state-of-the-art algorithms in areas such as localization, mapping, control theory, reinforcement learning, manipulation, among many others. Again, recent increases in computational power and storage make the use of non-parametric models more and more attractive, since vast amounts of data can now be maintained and processed in a timely manner.

Visual sensors are now common-place in robotics, for a variety of reasons: cameras are cheap, compact, relatively inexpensive, have a wide field of view both vertically and horizontally, and are able to provide a rich representation of the environment. The colour and texture information encoded in an image allow for tasks that no other sensor can accomplish, such as traffic signs and optical character recognition, and also substantially improve results on other tasks such as object recognition, tracking, scene reconstruction, and many others. In the particular case of motion estimation problems, visual sensors are attractive because they can provide estimations in the full six degrees of freedom that constitute 3D navigation, and are not dependent on any particular method of locomotion.

Chapter 3

Learning Visual Odometry Estimators

The aim of this chapter is to combine the tools and techniques introduced in the previous chapter to create a unified framework for visual odometry, using information extracted from a pair of frames to estimate camera translation and rotation[1]. As mentioned previously (Problem Statement, Section 1.2), most current visual odometry algorithms rely on a geometric model to provide translation and rotation estimates, and are therefore limited to the complexity of this model and the correct estimation of its corresponding calibration parameters. In fact, this approach is present in virtually all visual odometry literature, ranging from monocular [9, 25, 61] to stereo [52, 56, 86, 146] configurations, including the use of omnidirectional cameras [18, 108, 109, 128] and data fusion with other sensors, such as a IMU [57] or a low-cost GPS [2]. Several successful techniques for calibration have been proposed, however there is no guarantee that the resulting parameters will not change over time, due to vibration, mechanical shocks or changes in temperature. Self-calibration algorithms [17, 34] are able to track and auto-correct changes in calibration parameters, however they still assume a known pre-determined geometric model, which limits flexibility.

Alternative approaches to visual odometry, that rely on machine learning techniques instead of the standard geometric models, are still scarce and experimental, and

[1]The contents of this chapter were presented at the International Symposium on Experimental Robotics (ISER) 2010, under the title *Multi-Task Learning of Visual Odometry Estimators* [41].

insofar as this thesis is concerned can be limited to works of Roberts et al. In [102] the authors use a KNN-learner voting method to estimate a vehicle's velocity and turn rate, with each learner taking as input the average of the sparse optical flow in a grid-divided image. Subsets of the sparse optical flow distribution are used to cope with a variable dimensionality, and a study is conducted to verify the impact of such dimensionality reduction in the results. A similar idea is explored in [103], where a constant pixel depth is assumed and the Expectation-Maximization (EM) algorithm [26], in conjunction with an extension to PPCA [131], is used to perform a linear mapping between sparse optical flow and incremental motion. A generative model is used to estimate each optical flow subspace using only the observed measurements, and a per-pixel Gaussian mixture outlier process is used to deal with image regions that violate the constant pixel depth assumption.

The most interesting and attractive aspect of using machine learning techniques, in particular non-parametric models, to perform the transformation from image information to camera motion is that it improves on the aspect of camera calibration. By eliminating the use of a strictly geometric model, the calibration methodology is substituted by a training process in which available data is used to optimize a much more flexible non-parametric model, capable of capturing nuances of the underlying function that a strictly geometric model cannot, due to inevitable simplifications in the modelling process. This non-parametric model is then able to, by exploiting similarities in optical flow distribution between training and testing images, provide motion estimates that do not make a prior assumptions in regards to environment structure or camera configuration. Thus, the same visual odometry framework can be readily applied to any visual system and platform, provided that the training and testing datasets are obtained under similar conditions.

This thesis introduces the use of Gaussian processes (GPs) as the non-parametric tool used to perform this transformation from image information to camera motion. GPs possess several properties that make them especially attractive for this application:

- **Kernel functions**. The Gaussian process literature contains a vast number of covariance functions that are able to account for different relationships between inputs. The correct selection of which covariance function, or combination of covariance functions, to use allows the GP framework to correctly model global and local patterns of optical flow throughout the image. The training process automatically learns the parameters for these covariance functions, using examples that serve as samples from the unknown underlying function.

- **Bayesian treatment of uncertainty.** As a Bayesian inference tool, the GP framework allows for the introduction of prior distributions, reflecting our beliefs on how different aspects of the algorithm should evolve over time. The treatment of uncertainties in a probabilistic fashion also increases the algorithm's robustness to differences in training and testing data. As information obtained during navigation deviates from the information used during training, the uncertainty in estimation increases.

- **Multiple outputs.** A visual odometry system is almost always a multiple output algorithm, as most vehicles possess more than one velocity component that has to be estimated during navigation. This thesis explores both constrained 2D navigation, with two velocity components (forward and angular motion on the ground plane) and unconstrained 3D navigation, with six velocity components (linear and angular velocities in all three axes of motion). The Multiple-Output GP framework (MOGP) is capable of exploring the dependencies generated by vehicle constraints to improve results in each particular motion estimate. This thesis also extends the MOGP framework to allow the simultaneous estimation of all outputs, providing a full covariance matrix of uncertainties.

- **Scale recovery on a monocular configuration.** If the ground-truth data used during training contains scale information (i.e. it was obtained using a range sensor), the resulting non-parametric model will encode this information and provide an estimate for scale in new data as well, even in monocular configurations where scale recovery is not a trivial task. Again, as training data deviates from testing data this scale estimation will become less accurate, and the corresponding uncertainty will increase to reflect this deviation.

This chapter presents and describes the proposed visual odometry algorithm using Gaussian processes, explaining its various stages and components from the initial feature extraction from a pair of images to the final motion estimation between frames. It starts by providing an overview of the algorithm, introducing its various stages and components and how they relate to each other. It then describes how the input vector for the GP framework is obtained from sparse optical flow information between frames, in such a way that it maintains a fixed dimensionality and preserves spatial structure. Afterwards, the GP inference process is described, with emphasis on the vehicle models used in the 2D and 3D scenarios and on the selection of the covariance function that best models the optical flow distribution throughout the image. Finally, it concludes by presenting and discussing experimental results obtained using both ground and aerial vehicles, along with comparisons with other techniques and possible shortcomings that will be addressed in the next chapter.

3.1 Algorithm Overview

A diagram of the proposed visual odometry algorithm is presented in Fig. 3.1, with all its main steps and stages. This is a simplified version of the algorithm, containing only the basic components necessary to perform visual odometry. This basic algorithm will be further extended in the next chapter to include more functionalities (compare to Fig. 4.1). It is iterative, in the sense that it estimates camera translation and rotation between frames, and this motion estimate is then integrated over time to generate a pose estimate according to a global coordinate system.

As input the algorithm receives two images, IMG_1 and IMG_2, obtained using the same camera at different time intervals. It is assumed that these images contain a significant amount of overlapping, enough to allow a substantial matching between them, and also that the environment is static, so all motion observed is due to the camera's own translation and rotation (an assumption that will be relaxed in Chapter 5). As output the algorithm returns the velocity vector Y_{12} containing estimates for all degrees of freedom involved, along with a corresponding Σ_{12} covariance matrix.

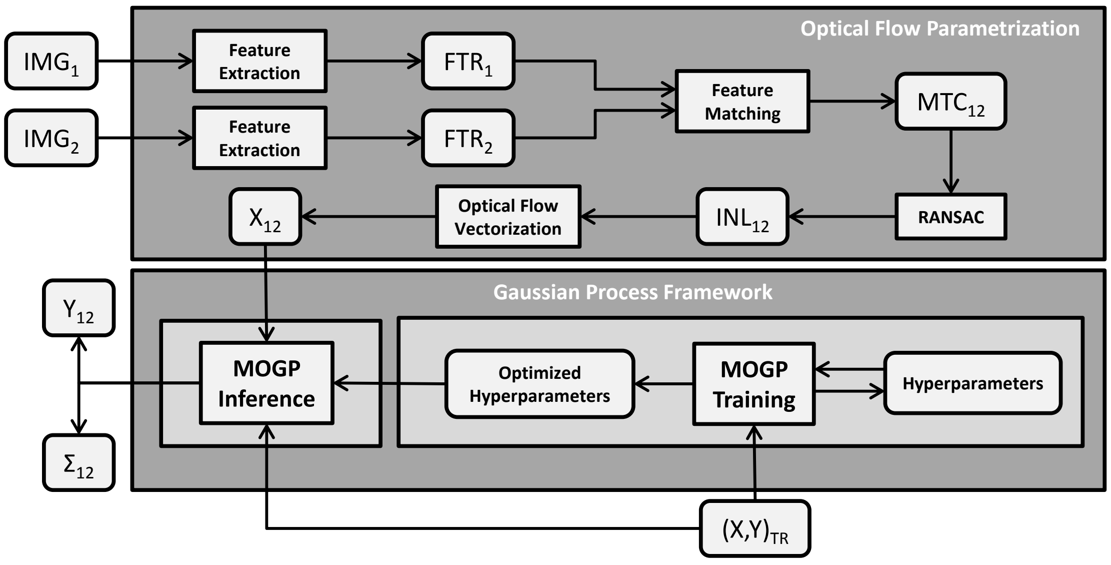


Figure 3.1 – Diagram of the proposed algorithm (simplified version). The *Optical Flow Parametrization* stage deals with the processing of a pair of images, generating the vector X_{12} that will serve as input for the *Gaussian Process Framework*, the second stage. The Gaussian Process Framework can be further divided into *Training*, where the optimized hyperparameters are obtained using a training dataset $(X, Y)_{TR}$, and *Inference*, where these hyperparameters are used to map the input vector X_{12} into the mean vector Y_{12} and covariance matrix Σ_{12}, that contain the motion estimates for all degrees of freedom and their corresponding uncertainty values.

In this basic implementation Σ_{12} is diagonal, containing one uncertainty estimate for each degree of freedom. Furthermore, a training dataset is deemed available, containing a set of images X_{TR} and their corresponding ground-truth information Y_{TR}, obtained from a different and independent sensor. For now we assume that this training dataset was collected under similar conditions (i.e. using the same vehicle and camera configuration, but from a different trajectory).

The proposed algorithm can be divided into two main stages: *Image Information Extraction* and *Gaussian Process Framework*. The first stage is responsible for processing both images and generates an input vector X_{12} containing the optical flow information between frames. This input vector is then passed to the second stage, responsible for mapping this input into the velocity estimate Y_{12} and covariance Σ_{12}. This mapping between inputs and outputs is performed using the hyperparameters obtained during a training process conducted prior to the beginning of navigation, utilizing the training dataset $(X, Y)_{TR}$. The next sections describe these two stages in further detail.

3.2 Image Information Extraction

The proposed approach uses sparse optical flow information from a pair of monochromatic images obtained using a single camera configuration (the camera is pointing forward in the case of 2D navigation and downwards in the case of 3D navigation). Dense optical flow methods, such as Lucas-Kanade [71], were also explored but discarded due to the number of parameters that needed to be manually determined, and also due to a wide variation in performance in different environment and driving conditions. A histogram filter is applied to all images, to account for global changes in luminosity. The environment around the vehicle is assumed to be mostly static, so any optical flow detected is solely due to the camera's own translation and rotation, and the frames-per-second rate is also assumed to be constant to allow the direct transformation from vehicle velocity to vehicle displacement. No other prior knowledge of the environment and/or visual system is necessary.

3.2.1 Optical Flow Parametrization

The initial feature extraction and matching processes are conducted using a combination of both the SIFT and the Shi-Tomasi corner detection algorithms (as described in Section 2.3.1), with sub-pixel accuracy and frame-to-frame tracking. This is done in order to ensure a dense distribution of features throughout the entire image, that can be translated into optical flow estimations without large gaps. Empirical tests show that the invariance properties of SIFT features ensure robustness during the matching process, whereas the Shi-Tomasi corner detector is particularly suitable for tracking over a series of frames. Any other similar method could be readily applied, both for speed purposes [7] or as a way to model different environment properties.

Examples of initial feature sets FTR in 2D navigation for two particular frames are presented in Fig. 3.2a, along with the corresponding matching sets MTC obtained in relation to their subsequent frames (Fig. 3.2b). Each matching pair is depicted by a line connecting both features, and it is possible to see a substantial amount of false matches, mostly due to structure similarity, poorly textured regions and occlusion caused by changes in viewpoint. These false matches are then removed using the RANSAC algorithm, as described in Section 2.3.3. If the environment is considered mostly static, it is natural to assume that the most probable motion hypothesis elected by RANSAC will correspond to the camera's own motion. This step is also useful in minimizing the interference of dynamic objects, since their features will generate matches that are not consistent with the most probable motion hypothesis, and therefore will be discarded as outliers.

The resulting inlier sets INL are depicted in Fig. 3.2c, and the corresponding epipolar lines generated by RANSAC, representing the most probable motion hypothesis, are presented in Fig. 3.3. Features were tracked for an average of 6 frames, and the overlapping regions ranged from 90% (forward motion) to 75% (hard turns).

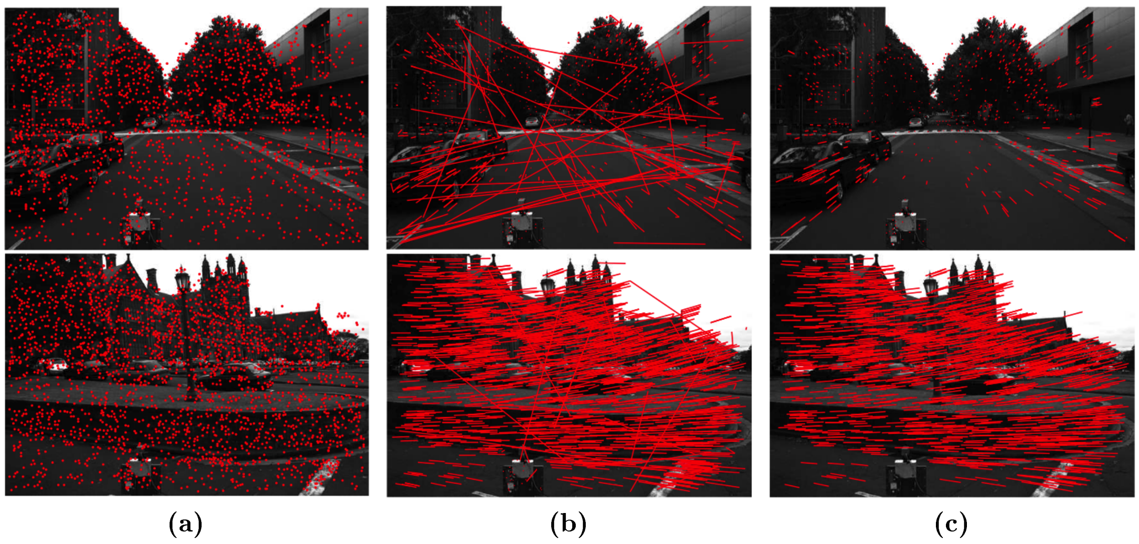


Figure 3.2 – Image information extraction in 2D navigation for the particular cases of forward motion (first row) and counter-clockwise rotation (second row). (a) Initial feature sets. (b) Initial matching sets. (c) Inlier sets, after RANSAC.

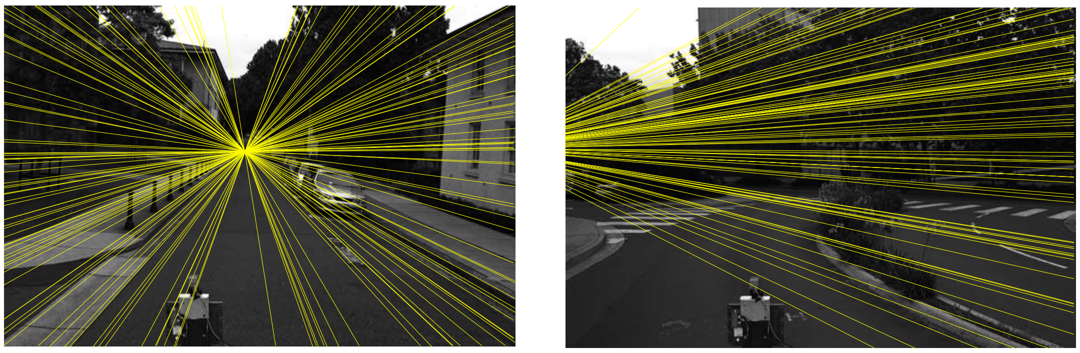

(a) (b)

Figure 3.3 – Epipolar lines in 2D navigation for the particular cases of (a) forward motion and (b) counter-clockwise rotation.

3.2.2 Preserving Spatial Structure

A straightforward way of generating the input vector X_{12} from the inlier set INL_{12} would be to use the individual optical flow information from each matching pair directly. In this scenario, X_{12} would be a vector of size ND, where N is the number of inliers and D is the number of components used to describe each individual optical flow estimate. However, the direct use of individual optical flow information to generate X_{12} would incur two problems that need to be addressed before a suitable input vector for the GP framework can be produced:

- **Different Sizes.** Two different pairs of images will most certainly generate inlier sets of different sizes, which would change the final dimension of X_{12} as a ND-dimensional vector. This would in turn change the nature of the underlying function that the GP is attempting to model, as the input space would be different for each input vector and therefore not comparable.

- **Different Distributions.** Two different pairs of images will mostly certainly generate inlier sets with a different distribution throughout the image, reflecting different portions of the overall optical flow configuration. Since optical flow information depends heavily on pixel coordinate (each portion of the image reacts differently to vehicle motion), any comparison between input vectors would also be rendered moot.

It is therefore necessary to generate an input vector X_{12} that both keeps a constant dimension regardless of the number of inliers detected and also maintains the spatial structure of optical flow distribution. The method proposed here to achieve these two conditions consists in dividing the image into equal-sized grids (Fig. 3.4), and assigning to each of them the subset of inliers whose coordinates lie within its boundaries (by convention, we use the feature coordinates on the first frame). The optical flow values for each grid can now be calculated as the average value of all its inliers' optical flow information. If a particular grid has no features, its optical flow values are calculated as the average value of its neighbouring grids, based on the assumption that changes in optical flow should be smooth throughout the image, varying radi-

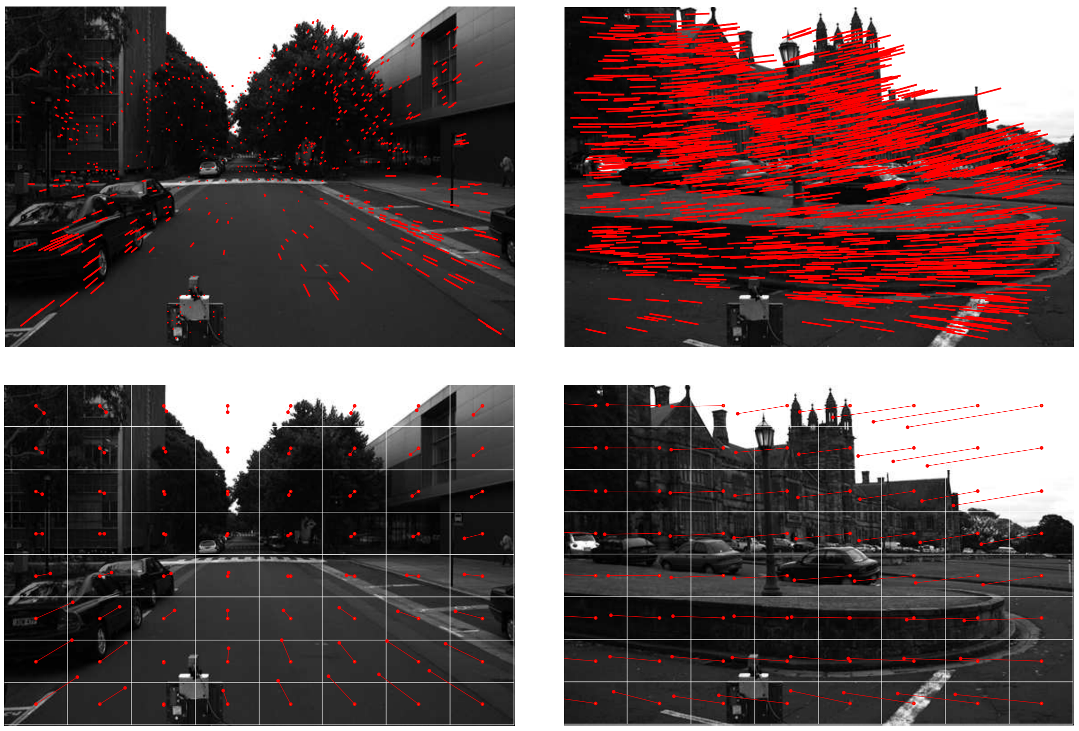


Figure 3.4 – Examples of optical flow parametrization into equal-sized grids in 2D navigation, for the particular cases of (a) forward motion and (b) counter-clockwise rotation. The average optical flow value of all matched features within each grid is used, and grids without any matched features receive the average value of its neighbouring grids (as it can be seen on the upper right portion of the images on the second column, where the sky does not have any discernible features but still registers optical flow).

cally only around the borders of objects and not inside them. This averaging process also serves to minimize the impact of dynamic objects on the calculations, since their optical flow values will now be diluted in the midst of all inliers in the same grid. The resulting input vector X_{12} is now of dimension hwD, where h and w are respectively the number of grids used to divide the image vertically and horizontally, and is generated by taking the optical flow components for each grid in a specific manner (i.e. starting on the top left grid and moving horizontally row by row).

Lastly, it is necessary to determine which optical flow components should be used to generate the input vector X_{12}. There are two different methods of parametrizing a 2D pixel shift throughout the image: the shift magnitude d and orientation θ, or the horizontal c and vertical s shift displacement. These two methods are correlated (Eq. 3.1), and therefore contain redundant information that should not be used in conjunction to generate the input vector, because it would create an unnecessarily high-dimensional problem. However, they encode optical flow in different ways, and the careful selection of which components to use could significantly improve GP inference performance.

$$s = d\sin(\theta) \qquad c = d\cos(\theta) \tag{3.1}$$

Fig. 3.5 depicts how each of the four optical flow components behaves during typical 2D translation and rotation situations. It is clear that d and c are the only ones with a consistent reaction to camera motion, indication that they encode this information in a way that can be recovered robustly. As expected, during translation d varies cyclically throughout the image, assuming higher values near the borders and lower values towards the center, and it is mostly stable during rotation. The values of c are also consistent, being vertically symmetrical during translation and stable during rotation. The angle θ is erratic due to the high angular sensitivity to errors when small distances are being covered, and s is affected by terrain irregularities, which even though could be helpful in a 3D scenario are not relevant in ground navigation.

For these reasons, the components d and c are used to generate the final input vector X_{12}, resulting in a $2hw$-dimensional vector. Tests were conducted under the MOGP

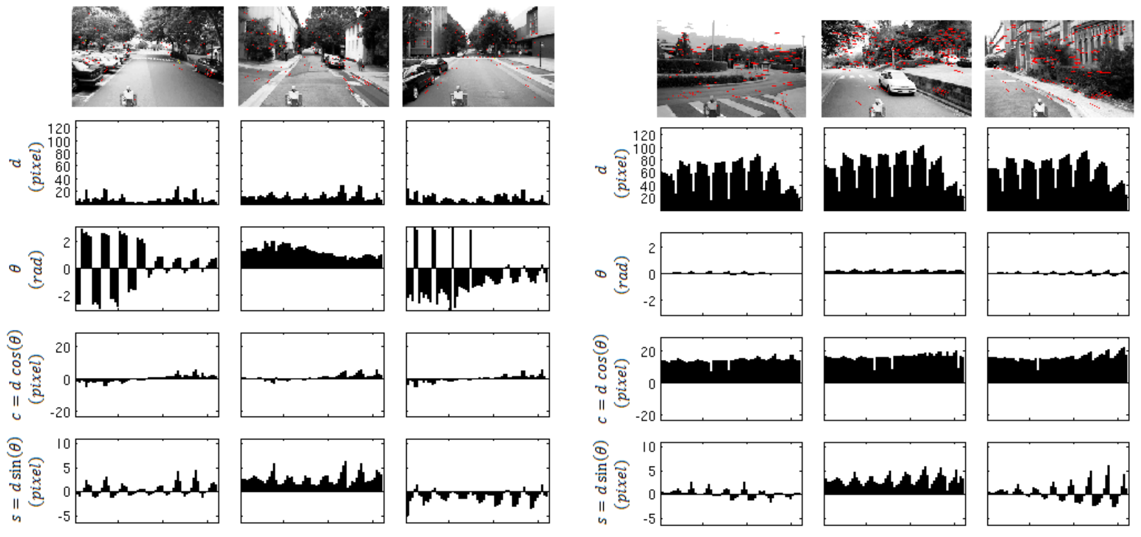


Figure 3.5 – Examples of the behaviour of each of the optical flow components in typical (a) forward motion and (b) counter-clockwise rotation. Each row indicates the distribution of a given component throughout the image (starting on the top left corner and moving horizontally row by row).

framework (Section 3.3) using different values for h and w, and the final values selected were $h = w = 8$, with a final dimensionality of 128 for the input vector. A smaller number of grids would create a poor characterization of local optical flow distributions, and a larger number of grids would create an excessively high-dimensional problem and also amplify small errors in optical flow calculation. These tests also validated the use of d and c to parametrize the final input vector, with the incorporation of other components leading to poorer performances.

3.2.3 Narrow Field of View

As stated previously, in the 3D experiments the camera was installed pointing downwards, observing the ground beneath the aircraft. In this configuration, the high altitude poses a special challenge in both feature extraction and matching, due to overall loss in detail and a high sensitivity to angular motion, that translates into inconsistent (and often small) overlapping regions between frames (see Fig. 3.6).

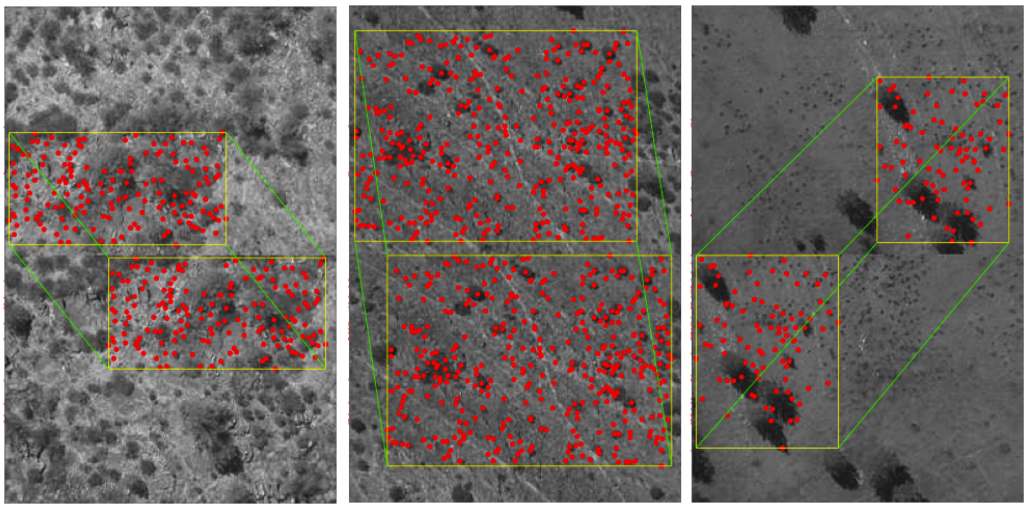


Figure 3.6 – Examples of inlier sets in 3D navigation. Red dots indicate matched features, and the yellow rectangles are the overlapping regions, representing the boundaries of matched features in each image. Note that these overlapping regions vary significantly from frame to frame, both in size and in location.

Assuming that the aircraft will maintain a considerable altitude and move roughly horizontally, it is reasonable to consider the ground plane as homogeneous, and all features to be at the same ground plane level. Under this assumption, the entire image will share the same optical flow information, that can be encoded by a single vector (i.e. by a single grid, using the methodology described in the previous section).

As before, this optical flow vector can be parametrized by either the shift magnitude d and orientation θ, or the horizontal c and vertical s shift displacement. Since dimensionality is not an issue in this particular case (the image is composed of a single grid cell), and motion is much more general and unconstrained (with components in both axes being equally relevant), all four components are used to generate the input vector X_{12}. Also, the position $(x,y)_i^c$ and size $(h,w)_i$ of the overlapping regions are directly related to camera movement, and therefore contain information that could be useful in the inference process. These parameters are illustrated in Fig. 3.7, and the final input vector is now of the form:

$$X_{12} = \{d, \theta, s, c, x_1^c, y_1^c, x_2^c, y_2^c, h_1, w_1, h_2, w_2\}. \tag{3.2}$$

It is important to note that, during the GP training process, the relevance of each parameter will be learned based on available data, leading to the removal of certain components and the strengthening of others based on their impact on the final estimates.

3.3 Gaussian Process Learning

Once the input vector X_{12} has been generated, the next step is to feed it into the GP framework to produce the corresponding velocity estimates, as described in Section 2.2.6. The GP inference process uses the optimized hyperparameter set θ obtained during the training process, from training data collected prior to the beginning of navigation. This section is dedicated to the various stages involved in the inference process. Initially, the 2D and 3D vehicle models are described, indicating which degrees of freedom are involved in the incremental localization process and how to go from velocity to pose estimates. Later on, the covariance function used in the inference process is introduced, along with a method for hyperparameter sharing that serves to decrease the dimensionality of the optimization problem during the training stage. Finally, the optimization process itself is described, showing how to obtain the final hyperparameter set that will be used for inference during navigation.

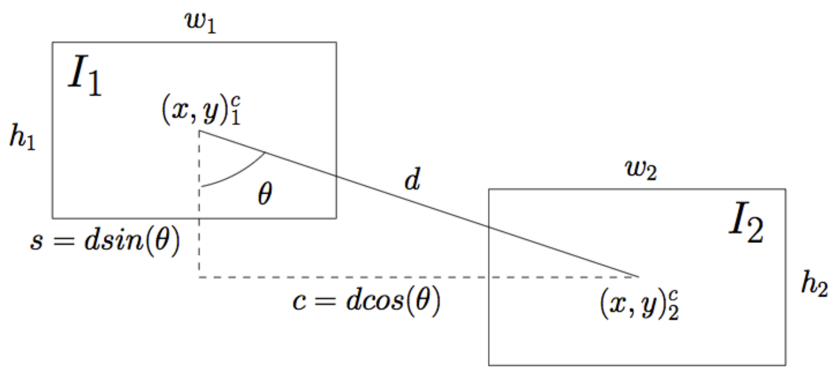
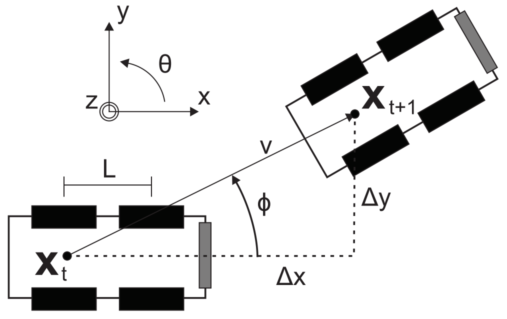


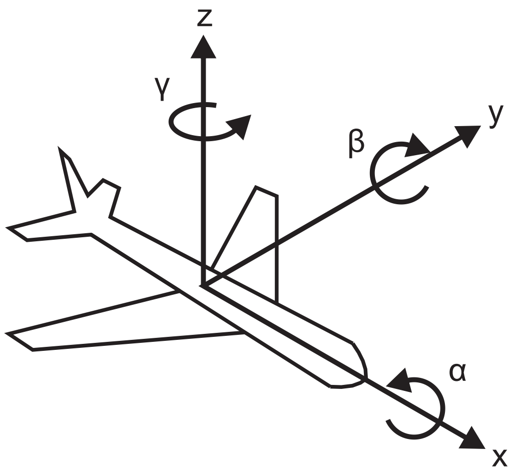
Figure 3.7 – Diagram of the parameters used to generate the input vector X_{12} for 3D unconstrained navigation.

3.3.1 Vehicle Models

The vehicle model governs how the vehicle navigates around the environment, and specifically it determines how many degrees of freedom are available for motion. The number of degrees of freedom (also known as *tasks*) in turn determines how many outputs will have to be estimated from each frame, as optical flow may be generated by a different combination of linear and angular velocities. Once these estimates are obtained, the vehicle model allows their integration into a pose estimate, that measures the vehicle's incremental motion from one frame to another. Two different vehicle models are described here (Fig. 3.8), a *constrained 2D ground* vehicle and an *unconstrained 3D aerial* vehicle. Similar platforms are used during experiments, and the unconstrained 3D aerial vehicle model provides a generic platform for any visual odometry application.



(a)



(b)

Figure 3.8 – Vehicle models used in this thesis. (a) 2D vehicle model, with two degrees of freedom: forward translation v and angular rotation ϕ. (b) 3D vehicle model, with six degrees of freedom: linear velocities $\dot{x}$, $\dot{y}$ and $\dot{z}$ and angular velocities α (roll), β (pitch) and γ (yaw), in Euler angles. The aircraft used during experiments, even though most of its linear velocity comes from $\dot{x}$, is also capable of experiencing motion in both other axes due to air resistance and draft, and therefore is modelled as an unconstrained 3D object.

Ground Vehicles

The constrained 2D vehicle model is the standard model used in ground navigation, because it reflects most of the nonholonomic vehicles, in which the degrees of freedom are limited to a linear forward motion and an angular rotation over the z-axis (a diagram of this model is depicted in Fig. 3.8a). Within this model, pose is defined by $\mathbf{x} = (x, y, \theta)$ and motion is divided into two independent steps: a rotation ϕ over the z-axis followed by a forward motion $v = \sqrt{\Delta x^2 + \Delta y^2}$ in orientation $\theta + \phi$. The pair $\mathbf{v} = (v, \phi)$ is capable of fully describing vehicle motion between frames, and therefore represents the two degrees of freedom that must be estimated by the GP framework, based on visual information. The calculation of $\mathbf{x}_{t+1}$ for a timestep Δt, given $\mathbf{x}_t$ and $\mathbf{v}_t$, is of the form:

$$x_{t+1} = x_t + v_t \cos(\theta + \phi_t)\Delta t \tag{3.3}$$

$$y_{t+1} = y_t + v_t \sin(\theta + \phi_t)\Delta t \tag{3.4}$$

$$\theta_{t+1} = \theta_t + \phi_t \Delta t. \tag{3.5}$$

As the timestep Δt decreases the order in which each step (forward motion and rotation) is calculated becomes less and less important. At the limit $\Delta t \to 0$, this implies that $dy/dx = \tan\theta$. Since $dy/dx = \dot{y}/\dot{x}$ and $\tan\theta = \sin\theta/\cos\theta$, this condition can be written as a Pfaffian constraint [16]:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0. \tag{3.6}$$

This constraint is satisfied if $\dot{x} = \cos\theta$ and $\dot{y} = \sin\theta$. Furthermore, any scalar multiple of this solution is also a solution, and the scaling factor corresponds directly to the linear speed v of the vehicle. Thus, the two first scalar components of the configuration transition equation are $\dot{x} = v\cos\theta$ and $\dot{y} = v\sin\theta$. The next step is to derive the equation for $\dot{\theta}$. Denoting s as the actual distance travelled by the car (the integral of speed, $\dot{s} = v$), and ρ as the radius of the circle that is traversed by the center of the rear axle (assuming a fixed steering wheel), it results that $ds = \rho d\theta$. From trigonometry, $\rho = L/\tan\phi$, where L is the distance between axles, which implies:

$$d\theta = \frac{\tan\phi}{L} ds. \tag{3.7}$$

Finally, dividing both sides by dt yields:

$$\dot{\theta} = \frac{v}{L} \tan\phi. \tag{3.8}$$

Aerial Vehicles

A fully unconstrained object has six degrees of freedom in the 3D space: it can move forwards and backwards in each of the three xyz-axes and it can also rotate around each of these axes. The translational part of this motion is given by the corresponding velocities $\dot{x}$, $\dot{y}$ and $\dot{z}$, while the rotational part is defined here by α (roll), β (pitch) and γ (yaw), the *Euler angles.* The transformation, in homogeneous coordinates, from point $\mathbf{x}_1 = (x_1, y_1, z_1, 1)^T$ to $\mathbf{x}_2 = (x_2, y_2, z_2, 1)^T$ is given by $\mathbf{x}_2 = RT\mathbf{x}_1$, where T is the *translation matrix* and R is the *rotation matrix.* They are of the form:

$$T = \begin{bmatrix} 1 & 0 & 0 & \dot{x} \\ 0 & 1 & 0 & \dot{y} \\ 0 & 0 & 1 & \dot{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

The translation matrix T has the following properties: the inverse T^{-1} can be obtained by reversing the velocity vector used to generate it, and the product of two translation matrices is given by adding their corresponding velocity vectors (the multiplication of two translation matrices is, therefore, commutative). The rotation matrix R is special orthogonal, which has the following properties: it is normalized (the squares of the elements in any row or column sum to one), it is orthogonal (the dot product of any pair of rows or any pairs of columns is zero), its determinant $\det(R)$ is unitary

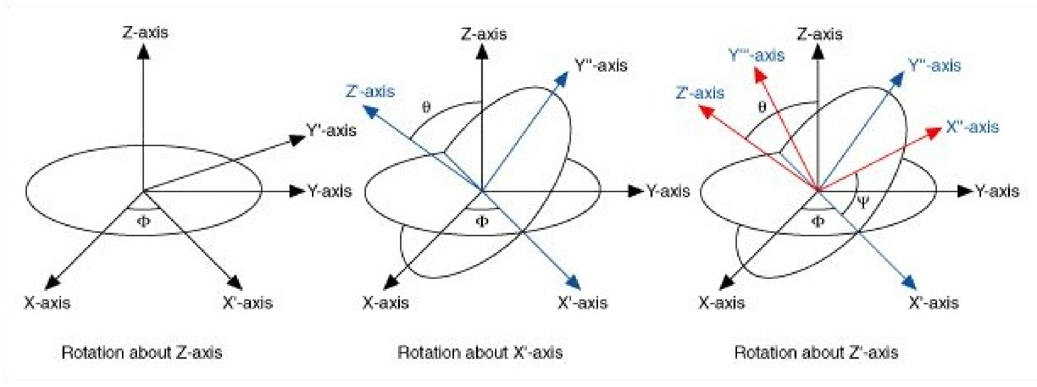


Figure 3.9 – Rotation on the zxz-axes. Initially a rotation ϕ is performed over the z axis, followed by a rotation θ on the newly-formed x'-axis and finally a rotation ψ on the newly-formed z'-axis.

and its inverse R^{-1} is equal to its transpose R^T. The combined *transformation matrix* $Tr = RT$ can then be written as:

$$Tr = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{11}\dot{x} + r_{12}\dot{y} + r_{13}\dot{z} \\ r_{21} & r_{22} & r_{23} & r_{21}\dot{x} + r_{22}\dot{y} + r_{23}\dot{z} \\ r_{31} & r_{32} & r_{33} & r_{31}\dot{x} + r_{32}\dot{y} + r_{33}\dot{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.10}$$

The elements r_{ij} of R are obtained from the notion that any general rotation can be described by a series of three individual rotations over orthogonal axes. Since rotation on tridimensional space is not commutative, it is necessary to define a consistent rotation order. A common convention, shown in Fig. 3.9, is the zxz convention, where a rotation ϕ is performed over the z-axis, followed by a rotation θ over the x'-axis and finally a rotation ψ over the z'-axis. The resulting rotation matrix $R_{zxz}(\phi, \theta, \psi)$ is calculated as follows:

$$R_{zxz}(\alpha, \beta, \gamma) = R_z(\psi)R_x(\theta)R_z(\phi) =$$

$$= \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix} \begin{bmatrix} c\phi & -s\phi & 0 \\ s\phi & c\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\psi c\phi - s\psi c\theta s\phi & -c\psi s\phi - s\psi c\theta c\phi & s\psi s\theta \\ s\psi c\phi + c\psi c\theta s\phi & -s\psi s\phi + c\psi c\theta c\phi & -c\psi s\theta \\ s\theta s\phi & s\theta c\phi & c\theta \end{bmatrix}, \tag{3.11}$$

where $s \equiv \sin$ and $c \equiv \cos$. The angles are usually chosen such that $-\pi < \phi \leq \pi$, $0 \leq \theta < \pi$ and $-\pi < \psi \leq \pi$. Inversely, it is also possible to obtain the angle values from Tr by solving:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c\psi c\phi - s\psi c\theta s\phi & -c\psi s\phi - s\psi c\theta c\phi & s\psi s\theta \\ s\psi c\phi + c\psi c\theta s\phi & -s\psi s\phi + c\psi c\theta c\phi & -c\psi s\theta \\ s\theta s\phi & s\theta c\phi & c\theta \end{bmatrix}, \tag{3.12}$$

which results in:

$$\theta = \cos^{-1}(r_{33}) \ , \ \psi = \tan^{-1}(-\frac{r_{13}}{r_{23}}) \ , \ \phi = \tan^{-1}(\frac{r_{31}}{r_{32}}). \tag{3.13}$$

It is also worth noting the problem of *singularity*, which happens when $\sin\theta = 0$. In this particular case, the z and z'-axes coincide, and therefore ϕ and ψ can be combined in one single rotation. Because of that, it is impossible to calculate their values separately, and the resulting degenerate rotation matrix R assumes the form:

$$R = \begin{bmatrix} r_{11} & r_{12} & 0 \\ r_{21} & r_{22} & 0 \\ 0 & 0 & \pm 1 \end{bmatrix}. \tag{3.14}$$

3.3.2 Covariance Function Selection

As stated previously, the covariance function plays a key role in the final performance of the GP inference process because it quantifies the relationship between points that will be used to model the underlying function between inputs and outputs. Thus, its selection must be carefully considered, based on prior knowledge of the phenomenon that the GP is attempting to learn from training data. Because optical flow information varies radically in different portions of the image, and also because there are angular measurements involved, the neural network covariance function was selected, due to its non-stationary property and the ability to model sharp transitions between different cell grids[2]. Section 2.2.4 provides an overview of different covariance functions and their properties, including the neural network, and its final equation is presented here again for convenience:

$$k_{ii}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \arcsin\left(\frac{2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}}'}{\sqrt{(1 + 2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}})(1 + 2\tilde{\mathbf{x}}'^T \Sigma \tilde{\mathbf{x}}')}}\right), \tag{3.15}$$

where $\tilde{\mathbf{x}} = \{1, x_1, \ldots, x_D\}$ is an augmented vector for the input point $\mathbf{x}$ and σ_f is the signal variance hyperparameter. The remainder of this section addresses two further issues that arise from the covariance selection process: 1) *Cross-covariance function selection*, which is the covariance function that models the relationship between data points from different outputs; and 2) *Hyperparameter sharing*, which provides a method to decrease the dimensionality of the optimization problem during training by enforcing the same length-scale values for different components of the input vector.

Cross-Covariance Function Selection

Visual odometry is a multiple-output application, meaning that each frame encodes motion from more than one degree of freedom. The previous section described two different vehicle models, a constrained 2D ground vehicle, with 2 degrees of freedom, and an unconstrained 3D aerial vehicle, with 6 degrees of freedom. The neural network covariance function (Eq. 3.15) only correlates input points that correspond to the same output $i = j$. It is also necessary to define a positive-definite covariance function that correlates points from different outputs $i \neq j$, thus allowing the GP framework to use all information available to improve results on all outputs. The constraints imposed by the vehicle model naturally correlate different degrees of freedom to only certain specific combinations (i.e. a ground vehicle can only move forward

[2]Other covariance functions, such as the squared exponential and the Matérn, were also considered, but empirical tests confirmed the assumption that the neural network covariance function is indeed the most suitable for this particular visual odometry application, mostly because it deals with angular quantities in the sparse optical flow information.

and backwards, and linear velocity tends to decrease as angular velocity increases). This configuration renders the cross-dependency modelling even more attractive in the particular case of visual odometry.

We propose a cross-covariance neural network function, in which two smoothing kernels are convolved [49] to obtain a positive-definite function capable of correlating multiple outputs (Eq. 3.16). This methodology was described and developed in Section 2.2.6 for the particular case of the squared exponential covariance function, capable of addressing T outputs, and here it is extended to generate a non-stationary cross-covariance function:

$$k_{ij}(\mathbf{x},\mathbf{x}') = \sigma^2_{f,ij} \frac{\arcsin\left(\frac{2\tilde{\mathbf{x}}^T\Sigma\tilde{\mathbf{x}}'}{\sqrt{(1+2\tilde{\mathbf{x}}^T\Sigma\tilde{\mathbf{x}})(1+2\tilde{\mathbf{x}}'^T\Sigma\tilde{\mathbf{x}}')}}\right)}{(|\Sigma_i||\Sigma_j|)^4\sqrt{|\Sigma_i+\Sigma_j|}}, \tag{3.16}$$

where $\tilde{\mathbf{x}} = \{1, x_1, \ldots, x_D\}$ is again an augmented vector for the input point $\mathbf{x}$, the signal variance hyperparameter $\sigma_{f,ij}$ is now specific for each output pair and $\Sigma = \sigma^2_{f,ij}(\Sigma_i(\Sigma_i+\Sigma_j)^{-1}\Sigma_j)$.

Hyperparameter Sharing

During the optical flow parametrization stage (Section 3.2) it was determined that, for a ground vehicle, the final vector that will serve as input for the GP framework contains 128 dimensions: two optical flow components for each of the sixty-four grids in which the image was divided, which translates to also 128 length-scale hyperparameters. A ground vehicle contains 2 degrees of freedom (Section 3.3.1), which increases the number of length-scale hyperparameters to 256, since each output maintains its own length-scale matrix Σ_i. The neural network covariance function also requires an augmented vector $\tilde{\mathbf{x}}$ for each input, a signal variance hyperparameter for each output (and one for the cross-covariance function), and two noise parameters. The total number of hyperparameters is therefore $128 \times 2 + 2 + 3 + 2 = 263$, which poses a challenge during the optimization process due to the excessively high-dimensional space in which the input points are handled.

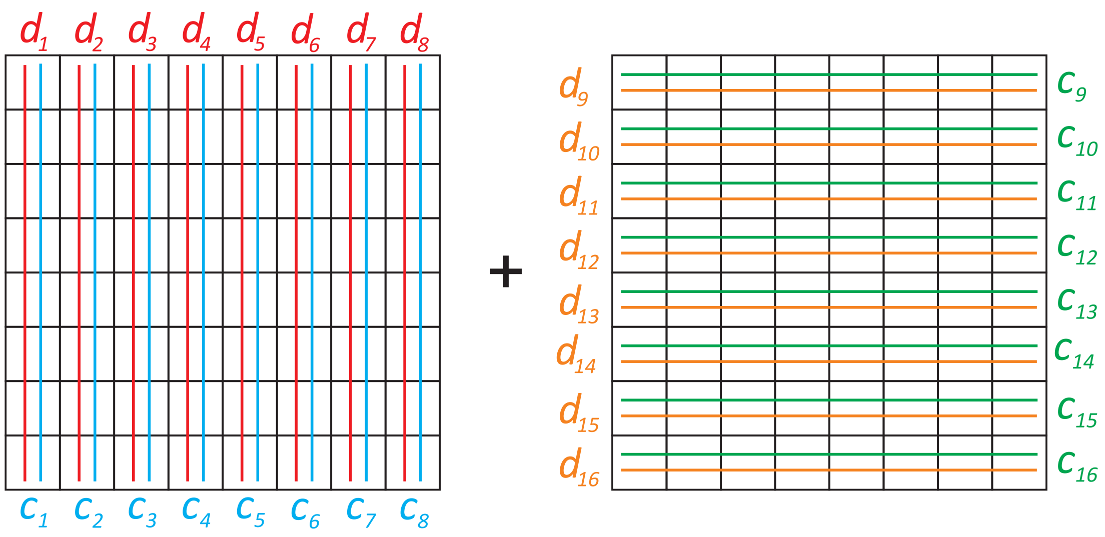

d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8
+
d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15} d_{16}
c_9 c_{10} c_{11} c_{12} c_{13} c_{14} c_{15} c_{16}

Figure 3.10 – Diagram showing the hyperparameter sharing technique. Each square represents one grid in the image, and the lines represent the row and columns that are shared by the length-scale of the same colour.

A technique for decreasing the number of hyperparameters necessary to fully describe the covariance function, and by extension for decreasing the optimization dimensionality, is presented here. It is based on the assumption that regions in the grid-divided image that belong to the same row or column have a similar impact on the underlying function modelling process, and therefore may share the same length-scale hyperparameter. Two independent neural network covariance functions are used (Fig. 3.10), one to model a similar impact in rows and another to model a similar impact in columns, and both are added in order to generate the final covariance function:

$$k^s_{ii}(\mathbf{x}, \mathbf{x}') = k^1_{ii}(\mathbf{x}, \mathbf{x}', \theta_1) + k^2_{ii}(\mathbf{x}, \mathbf{x}', \theta_2) \tag{3.17}$$

$$k^s_{ij}(\mathbf{x}, \mathbf{x}') = k^1_{ij}(\mathbf{x}, \mathbf{x}', \theta_1) + k^2_{ij}(\mathbf{x}, \mathbf{x}', \theta_2). \tag{3.18}$$

There is no sharing between length-scales of different outputs, and each optical flow component (in this case, distance d and cosine c) is still modelled independently, resulting in 16 length-scales for each neural network covariance function and 32 for the final covariance function. The total number of hyperparameters is now 16 ×

$2 \times 2 + 2 + 3 + 2 = 71$, roughly 25% of the original number. This hyperparameter sharing technique also reduces the *curse of dimensionality* problem, where data in a high-dimensional space becomes sparse and insufficient to correctly represent the underlying function. By imposing constraints in different dimensions, it becomes easier to recover correlations between input points in this high-dimensional space with a reasonable amount of training data.

3.3.3 Hyperparameter Optimization

Once the auto and cross-covariance functions have been defined, along with the corresponding hyperparameter set θ, an optimization stage is necessary to obtain the non-parametric model that best represents the underlying function, given by a training dataset Λ obtained prior to the beginning of navigation. This training dataset is composed of a set of N input vectors $\mathbf{x}_n$ and their corresponding output vectors $\mathbf{y}_n$, containing motion estimates for all degrees of freedom. Ideally, the training dataset should be collected under similar conditions as the ones the vehicle will encounter during navigation (i.e. same vehicle, camera configuration and a similar environment), to minimize the deviation in optical flow distribution between available and new information. Variations in each of these aspects are further explored in the next chapter, providing an insight on how well the algorithm is able to generalize to unforeseen conditions.

The cost function used during the optimization stage is the multiple-output log-likelihood function $\mathcal{L}(\mathbf{y}|\Lambda, \theta)$, as described in Section 2.2.6 and repeated here for convenience (Eq. 3.19). The optimization starts from a random hyperparameter set, and Fig. 3.11 shows the influence of initial conditions on the maximization of the negative log-likelihood function. Even though all initial random hyperparameters generated similar log-likelihood values, they converged to different values, and one in specific (light blue line) converged to a value much smaller than all the others, indicating that it represents a better fit to the training data. This sensitivity to initial conditions is amplified by the large number of variables involved in the optimization

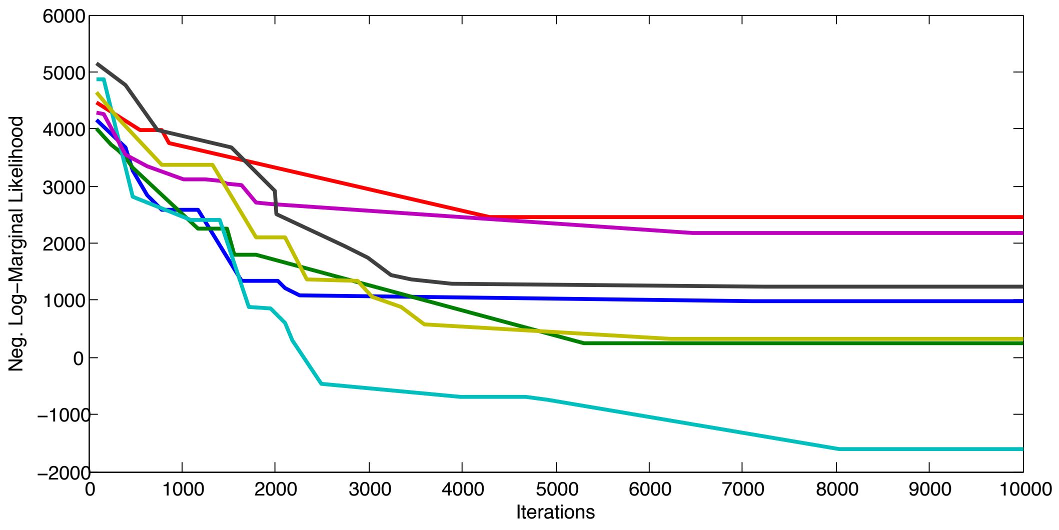


Figure 3.11 – Examples of marginal log-likelihood optimization with different hyperparameter initialization. It is possible to see that, even though the initial log-likelihood values were similar, different initial hyperparameters converged to different final values, and one in particular converged to a value much lower than the other ones, indicating that it represents a better fit to the training data given the model.

process, that translates into a large number of local minima in which a gradient descent algorithm may get trapped.

$$\mathcal{L}(\mathbf{y}|\Lambda,\theta) = -\frac{1}{2}\ln|K| - \frac{1}{2}\mathbf{y}^T K^{-1}\mathbf{y} - \frac{N}{2}\ln(2\pi) \tag{3.19}$$

To address this sensitivity to initial conditions, heuristic approaches for initial hyperparameter selection, such as the Monte Carlo-based *simulated annealing* [60], are considered. The name annealing comes from a namesake technique in metallurgy, that involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects, both attributes that depend on thermodynamic free energy. While the same amount of cooling brings the same amount of decrease in temperature, it will bring a bigger or smaller decrease in the thermodynamic free energy depending on the rate in which it occurs, with a slower rate producing a bigger decrease. The notion of slow cooling is implemented in the simulated annealing algorithm as a slow decrease in the probability of accepting worse solutions as it explores the solution space (Fig. 3.12). This is a fundamental property in meta-heuristics, be-

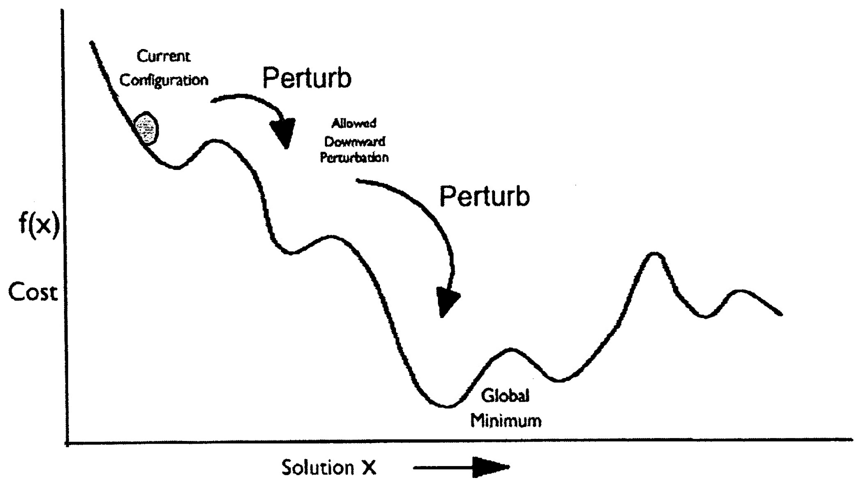


Figure 3.12 – Example of simulated annealing. The solution is allowed to move into a higher energy level in the search for the global minimum, and this allowed perturbation decreases over time until convergence.

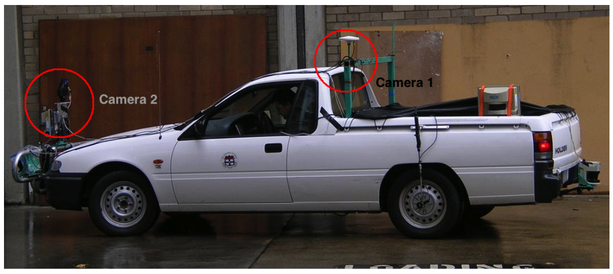
cause it allows for a more extensive search for the optimal solution. Once this search is finished, the resulting hyperparameter is further refined by a standard gradient descent algorithm, converging to the nearest local minima which serves as the final hyperparameter set.

Another useful technique in hyperparameter selection is the cross-validation (described in Section 2.2.5), where part of the training dataset is withheld during the optimization process and then used to provide a measurement of inference error. A large error indicates that the current hyperparameter set, although providing a good fit for the training information, lacks the generalization necessary to address new, yet unobserved, information. These three techniques (simulated annealing, gradient descent and cross-validation) comprise the core of the training stage used in the visual odometry algorithm proposed in this thesis. A random set of hyperparameters goes through simulated annealing, to find suitable initial conditions for gradient descent, and the cross-validation error of the resulting hyperparameter set is calculated using part of the training data withheld during optimization. This process is repeated a certain number of times, and the hyperparameter set with the lowest values for negative log-likelihood and cross-validation error are selected and used during the experimental tests.

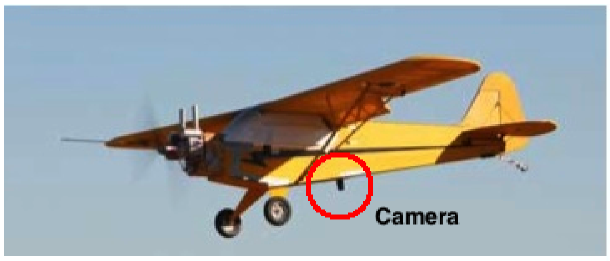
3.4 Experimental Results

The visual odometry methodology described in this chapter was evaluated using real data, collected from modified vehicles (Fig. 3.13) equipped with a single uncalibrated camera and other sensors for ground-truth and comparison purposes. These tests are divided into two categories: ground (using the constrained 2D vehicle model) and aerial (using the unconstrained 3D vehicle model) experiments. Initial tests were conducted using ground vehicles, because they are constrained to 2D navigation and therefore contain fewer degrees of freedom. This translates into fewer outputs for the GP framework, and also a smaller computational complexity during the training and inference stages. Once the algorithm was working satisfactorily, the same framework was extended to 3D environments in the aerial experiments, which provide a platform for any visual odometry application. It is worth noting that this transition was seamless, no change in the algorithm was necessary.

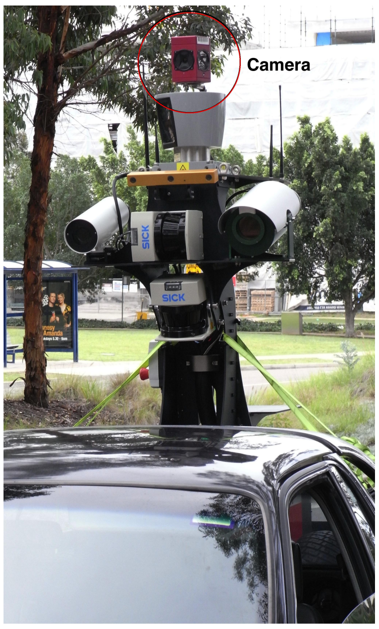
This section presents and discusses the results obtained in these experiments, starting with ground vehicles and later moving on to aerial vehicles. These results are further



(a) Modified car



(b) Unmanned Aerial Vehicle



(c) Robotic platform for generalization testing

Figure 3.13 – Vehicles used in experiments.

improved in the next chapter, where extensions to the basic algorithm described in this chapter are proposed as a way to address some of the limitations discussed here. The next chapter also provides a quantitative study of the results (Tables 4.2 and 4.3), comparing estimates obtained using different methods and how they relate to the techniques proposed in this thesis.

3.4.1 2D Experiments

The 2D experiments were conducted using the vehicle in Fig. 3.13a, equipped with a single camera, wheel odometry, two SICK laser sensors and a GPS (with a 5m precision). The wheel odometry system was not used during the experiments, the SICK lasers were used to provide ground-truth for the GP framework, and the GPS was used for comparison purposes only. Images were obtained at a rate of 5 frames per second and a resolution of 1152x758 pixels, which was then downsampled to 384x252 pixels (one third of the original size) and converted into monochromatic intensity values. The reasons for this downsample and monochromatic conversion are two-fold: 1) To verify the robustness of the algorithm in low-resolution cameras (marginally better results were obtained with higher resolutions); 2) To speed up the SIFT (or equivalent) feature extraction and matching processes. During data acquisition the vehicle navigated an urban environment at speeds of up to 40 km/h, and interacted normally with pedestrians and other vehicles.

The training dataset is composed of 2500 images, collected as the vehicle was driven for roughly 2.5 km in the trajectory shown in Fig. 3.14a. Ground-truth information was obtained based on laser data, using the *Iterative Closest Point* (ICP) algorithm [70], and the resulting localization estimates are also shown in the same figure. Because they are incremental, these estimates are by themselves subject to drift due to the accumulation of small errors over time. Even though this drift could in principle be greatly reduced by fusing the estimates with an absolute sensor (such as GPS), here it was decided to use the ICP estimates directly as ground-truth information, with no further refinement. This is done to verify the ability of the proposed visual odometry algorithm to average over small errors by using a large training dataset to learn the underlying function directly from noisy information, and also to minimize the need for high-precision sensors during the training stage. Empirical tests show marginal improvements in localization when more accurate ground-truth is used, and virtually no improvement when the extensions to the basic framework proposed in the next chapter are incorporated.

The testing dataset is composed of 2000 images collected using the same vehicle and camera, but from a different trajectory of roughly 2 km, as shown in Fig. 3.14b. The ICP localization results are also depicted, however they are now calculated solely for comparison purposes, as no information is used during the inference process other than visual data. Also for comparison[3], the localization results obtained using Mono-SLAM [24] and the Structure-From-Motion algorithm described in Section 2.3.5 are presented in Fig. 3.15. The Mono-SLAM algorithm employs Exended Kalman Filters to track the position of features in the environment, and uses this information to recover the vehicle position. While this approach minimizes the impact of error accumulation from relative estimates, it also has a high computational cost and struggles with failures in the feature matching process. The calibration parameters were obtained

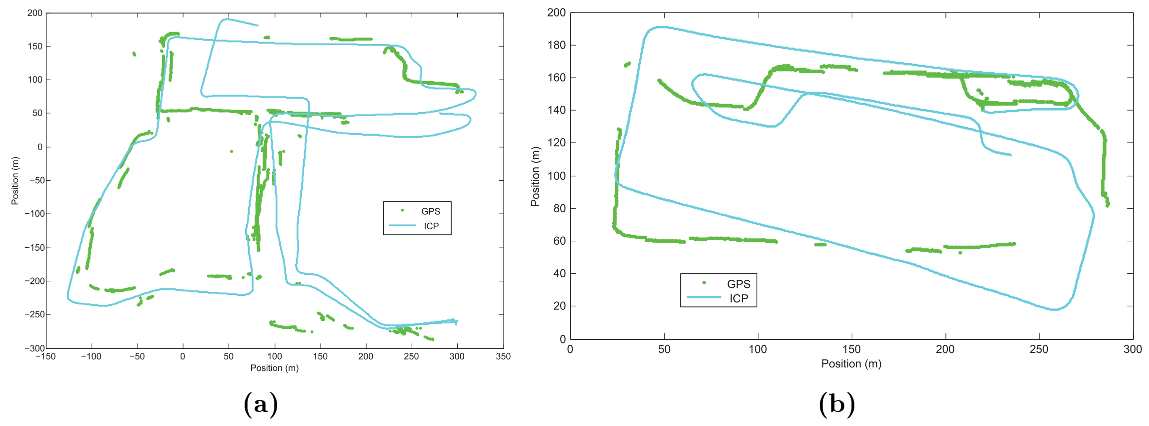


Figure 3.14 – Training and testing datasets. Even though tests were conducted outdoors, there are several areas in which the GPS signal (green dots) is not available, due to tree coverage or the presence of tall buildings nearby.

[3]The 1-Point RANSAC algorithm [106] was also tested, however it was unable to outperform SFM in the testing environments, mostly due to sharp turns and sudden changes in luminosity.

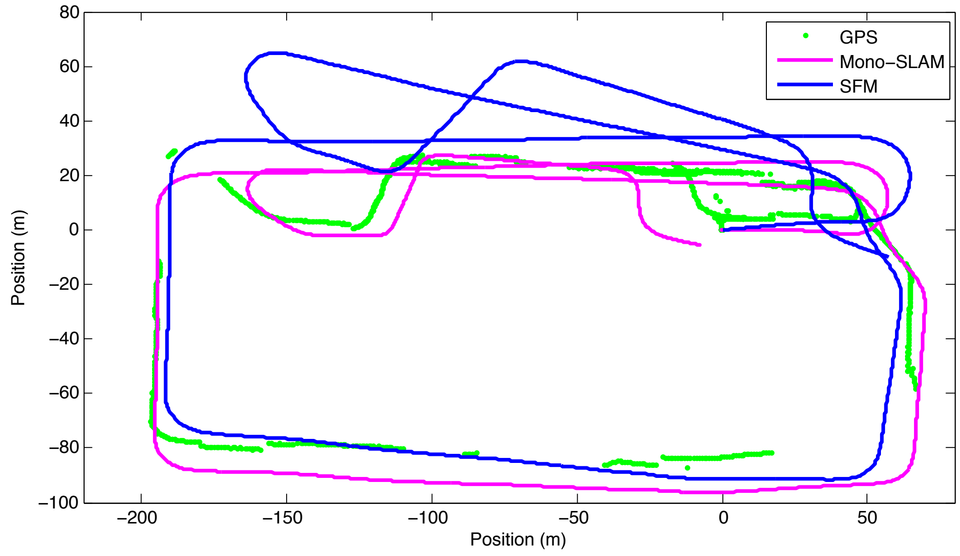


Figure 3.15 – Localization results obtained using standard geometric models. The purple line shows the Mono-SLAM [24] position estimates, while the blue line shows the position estimates obtained using the Structure-From-Motion (SFM) algorithm, with manual scale adjustment and constrained for 2D navigation.

manually [48], and scale was adjusted manually to minimize the overall translational error of the entire trajectory. Even so, it is possible to see some scale inconsistencies, especially in the initial portions of the algorithm (upper right corner of the map), where the vehicle is still moving at low speeds and manual scale adjustment tends to overestimate linear velocity. The SFM algorithm performed better at angular motion estimation, mostly due to the presence of distant features that can be used as fixed landmarks to measure rotation, however there is still some residual drift that accumulates over time and impacts significantly the localization results towards the end of the trajectory.

Finally, the localization results obtained using the proposed method, with a multiple-output Gaussian process (MOGP) to estimate all degrees of freedom, are depicted in Fig. 3.16. The first interesting aspect of these results is that the GP framework was capable of recovering scale to a high degree of precision (no scale adjustment was performed), by exploiting similarities in the optical flow distribution between training and testing images. Essentially, the scale information provided by the sensor used to generate the ground-truth estimates (a SICK laser, in this case) was encoded into the non-parametric model during the learning stage, and extrapolated to new data using inference based on the GP framework regression methodology.

The GP framework, however, struggles with angular motion estimation, resulting in a residual drift that also compromises long-term localization results. We attribute this angular drift to the presence of smaller overlapping areas between frames, which compromises the optical flow distribution throughout the entire image, and also to the presence of fewer vehicle turning samples in the training dataset. Since the vehicle moves mostly in a straight forward motion during navigation, the various smooth and sharp turns encountered during tests were under-represented and there was not enough information for a robust recovery. Furthermore, the MOGP framework is not capable of correctly modelling the cross-dependencies between outputs, generated by vehicle constraints that limit linear and angular motion to only certain specific combinations, and thus linear velocity information does not translate into a better angular velocity estimation, and vice-versa. This information exchange between different outputs would be valuable as a way to decrease the amount of training information

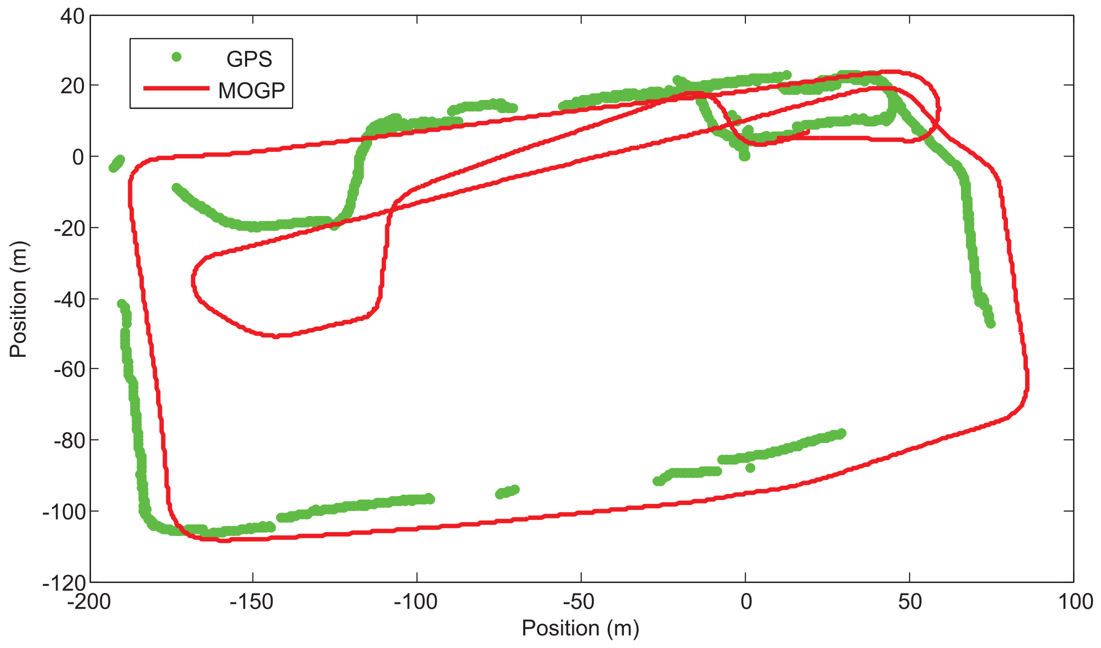


Figure 3.16 – Localization results obtained using the proposed algorithm, with a multiple-output Gaussian process (MOGP) to estimate all degrees of motion.

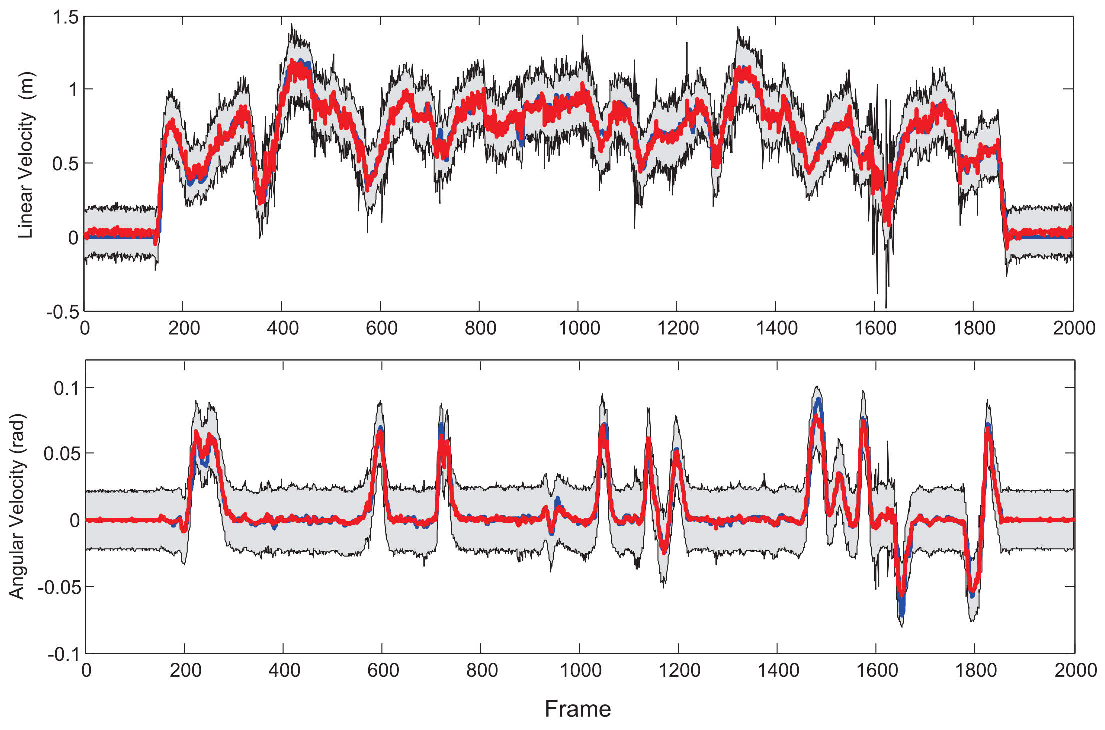


Figure 3.17 – Motion estimates for each output in 2D navigation. The blue line indicates ground-truth, the red line indicates the MOGP estimate, and the grey areas indicate the variance intervals within 2 standard deviations.

necessary, and also to increase performance in under-represented portions of the input space.

The mean and variance values for each output in all testing frames is shown in Fig. 3.17, where the blue line indicates ground-truth, the red line indicates the GP estimate and the grey area represents the variance intervals within 2 standard deviations. As we can see, virtually all ground-truth values fall within the variance intervals provided by the GP framework, indicating that the proposed approach's confidence in its own estimations is correctly represented. As expected, the most noticeable discrepancies between ground-truth and GP estimation values appear during sharp turns (the peaks and valleys in angular velocity), that as stated before are under-represented in the training dataset simply because there are not enough samples of this behaviour in a real navigational scenario in comparison to forward motion and smooth turns.

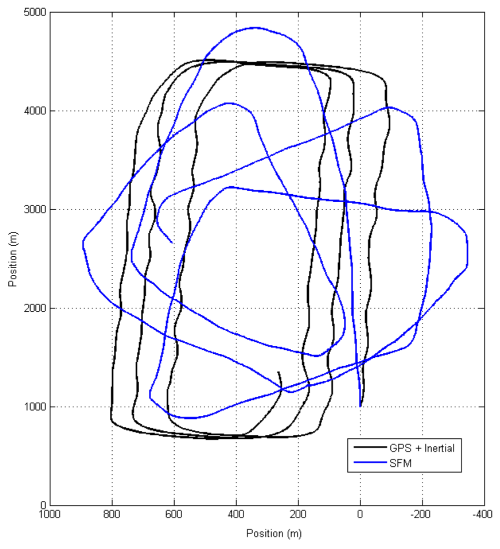
3.4.2 3D Experiments

The same visual odometry algorithm was also tested using data collected from an unmanned aerial vehicle (UAV, Fig. 3.13b) flight over a deserted area, at a rate of 3 frames per second and an average speed of 110 km/h. Due to the high altitudes (during training and testing the UAV maintained an average altitude of 80-100 m), the narrow field of view approach to optical flow parametrization was used to obtain the input vectors used by the GP framework. The UAV was also equipped with inertial sensors and GPS, that were fused to provide ground-truth data for training and comparison purposes. The first 4000 frames after stabilization were used for training, and the following 2000 frames were used for algorithm evaluation. The feature extraction and matching techniques failed to find any correspondences in around 2% of the image pairs, due to a lack of overlapping areas caused by severe angular motion, and in overall there was a wide variation in the size and shape of these overlapping areas (see Fig. 3.6), which constitutes a challenge for standard visual odometry algorithms. Frame pairs with no corresponding matches were avoided during training, and during evaluation the motion estimates from the previous timestep were repeated.

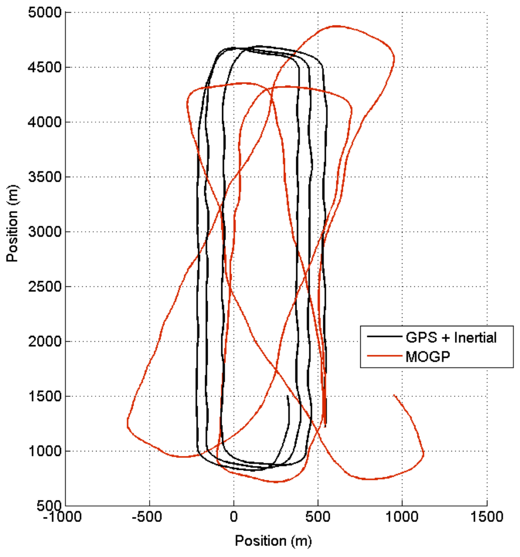
Fig. 3.18a shows the localization results obtained using the SFM algorithm, now unconstrained to address all six degrees of freedom[4]. We attribute this poor performance to three reasons: 1) Small and inconsistent overlapping areas between frames; 2) The high altitudes create a lack of depth perception in the ground plane; 3) Poor camera calibration, due to the narrow field of view that influenced the calibration process negatively. Similarly, the localization results obtained using the proposed approach are depicted in Fig. 3.18b, where it is possible to see a significant improvement over SFM in the sense that the overall shape of the trajectory was maintained throughout the entire run. However, there is still a substantial drift that compromises localization in both the vertical and horizontal planes, as it can be seen in Fig. 3.18c where the same results are presented in a 3D plot. Again, we attribute this drift to the lack of motion representation in the training dataset, which now becomes even more evident

[4]The Mono-SLAM algorithm was incapable of outperforming SFM under this new framework, due to difficulties in tracking features over a reasonable period of time in order to produce an accurate landmark map of the environment.

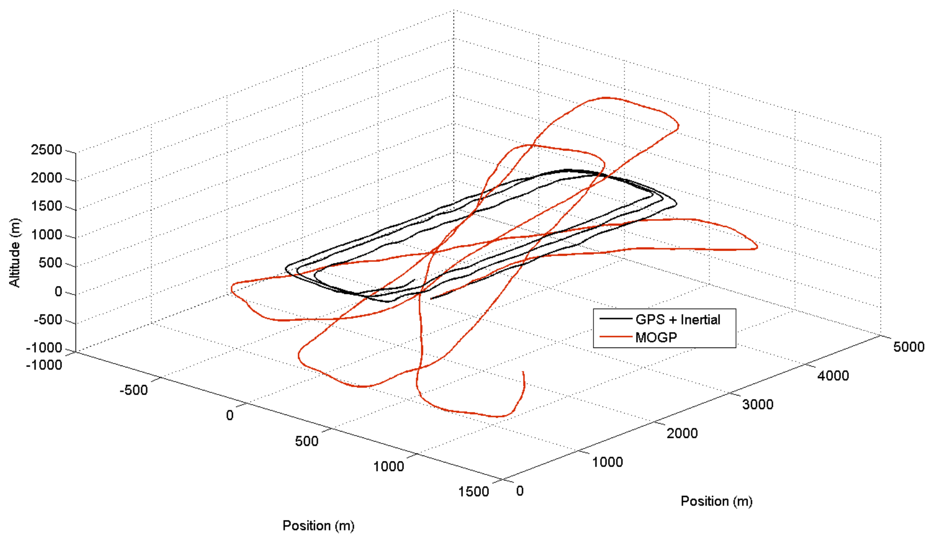
because in the 3D space there is a larger space of possible motion combinations that the GP framework has to correctly map to optical flow information.



(a)



(b)



(c)

Figure 3.18 – Localization results in 3D environments. (a) Results obtained using a structure from motion algorithm. (b) 2D plot of the results obtained using the proposed approach. (c) 3D plot of the results obtained using the proposed approach.

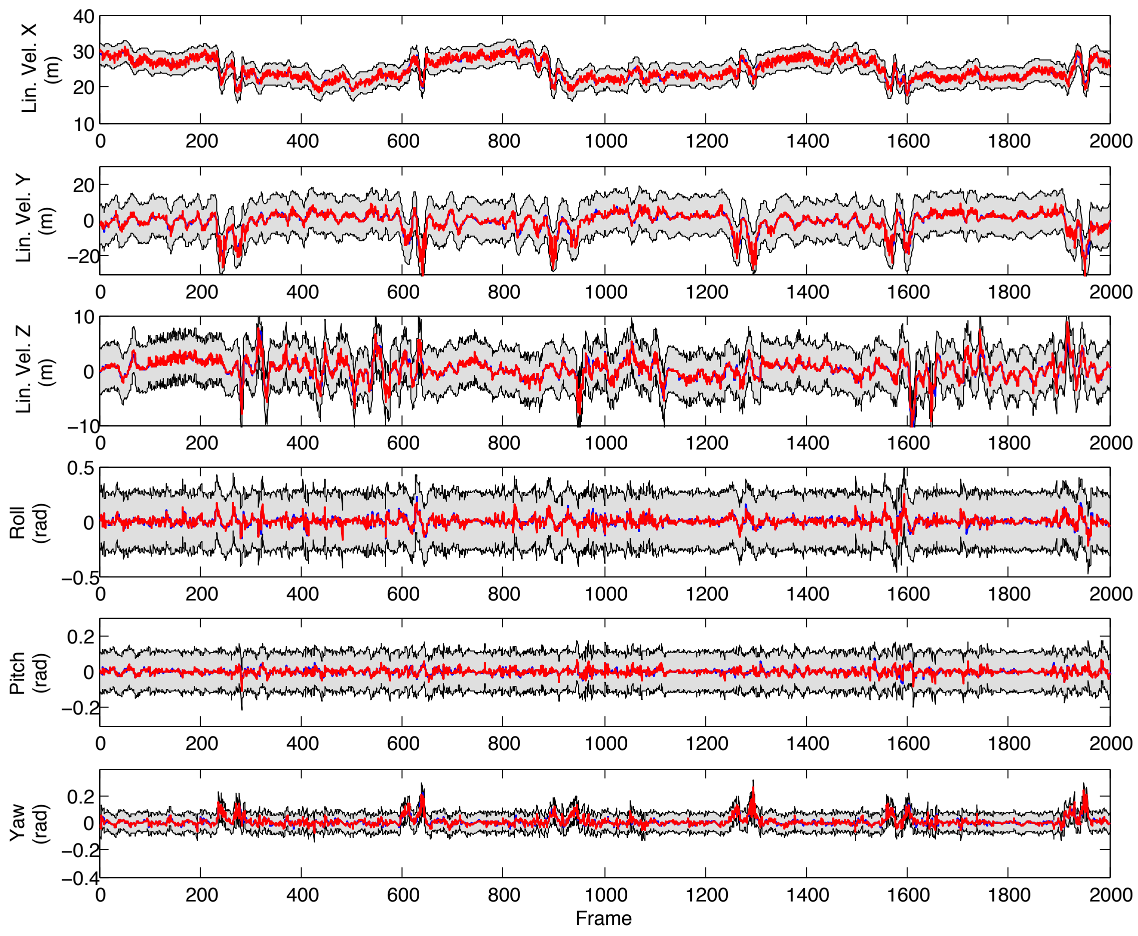


Figure 3.19 – Motion estimates for each output in 3D navigation. The blue line indicates ground-truth, the red line indicates the MOGP estimate, and the grey areas indicate the variance intervals within 2 standard deviations.

The mean and variance values for each output in all testing frames is shown in Fig. 3.19, where the blue line indicates ground-truth, the red line indicates the GP estimate and the grey area represents the variance intervals within 2 standard deviations. First of all, we can see that most of the linear motion comes from the x-axis, which is to be expected since this is the axis that represents forward motion in the UAV's relative coordinate system, however there is also some motion in the y and z-axes[5], oscillating both positively and negatively. Again, virtually all ground-truth values fall within

[5]The UAV was initially modelled with only four degrees of freedom (forward motion $\dot{x}$ and Euler angles α, β and γ), however this approach proved to be insufficient for this particular application and was extended to address all six degrees of freedom.

the variance intervals provided by the GP framework, indicating that these estimates, even though less accurate than the ones obtained during the 2D experiments, are still valid from a probabilistic standpoint. The proximity between ground-truth and GP estimation values also attest to the degree of accuracy that a visual odometry algorithm requires in order to provide useful information, as even minor imprecisions quickly accumulate over time compromising localization results.

3.5 Summary

This chapter introduced and described the first steps in obtaining a solution to the problem of visual odometry that uses Gaussian processes as the mapping function between optical flow information and vehicle motion estimates. The information encoded in two images is processed to extract a matching set that represents motion between frames, and this matching set is parametrized in order to generate a vector that is suitable as an input for the GP framework. Different methodologies are described, one to address the traditional scenario in which a camera is placed on top of a moving vehicle and another to address the more involving scenario of a camera placed on an aircraft pointing downwards. In the second case, the narrow field of view and the high sensitivity to angular motion pose a challenge in the feature extraction and matching processes.

Afterwards, the GP framework as a viable solution to the visual odometry problem is presented, introducing the vehicle models from which the GP outputs are defined, the choice of which covariance function to use and the process of hyperparameter optimization during the training stage. The goal of the proposed GP framework is to eliminate the need for a explicitly defined model, thus allowing the system to automatically learn the best transformation between inputs (sparse optical flow information) and output (motion estimates) given the available data. It is natural do assume that similar motions will generate similar optical flow patterns, and the GP framework is capable of exploiting these correlations to create a regression model robust enough to infer the behaviour of new, unobserved data.

Finally, experiments conducted in both 2D and 3D navigation scenarios are presented and discussed, showing the initial results of the proposed method in comparison with standard Structure-from-Motion algorithms. These results validate the assumption that GPs can indeed serve as a modelling tool for visual odometry applications, and achieve accuracies that are comparable to state-of-the-art traditional motion estimation techniques. The next chapter will focus on further improving this basic algorithm, addressing the various issues raised in this chapter and providing tools that increase the overall robustness and precision of the proposed method.

Chapter 4

Semi-Parametric Coupled Gaussian Processes

The previous chapter introduced the proposed visual odometry algorithm, describing its basic functionalities that allow image information to be mapped directly into vehicle motion estimates. The traditional camera calibration process, which optimizes a parametric model (the geometric camera model), is substituted by a much more powerful non-parametric model, the GP framework, that is capable of capturing and encoding nuances that a strictly geometric model struggles with. The GP framework eschews the need for any assumption in regards to the visual system utilized (the same algorithm can be transferred seamlessly to any camera configuration) or environment structure. The GP framework is also capable of encoding scale information into its non-parametric model, thus allowing the recovery of absolute scale even on a single camera configuration, a task that is non-trivial in monocular visual odometry.

However, as it was also stated in the previous chapter, there are still several shortcomings in the proposed method that need to be addressed before it can be truly used as a robust visual odometry solution[1]. One of these shortcomings is the algorithm's inability to correctly model cross-dependencies between different outputs, that would

[1]The techniques described in this chapter were presented at the International Conference on Robotics and Automation (ICRA) 2011 and 2012, under the titles *Visual Odometry Learning for Unmanned Aerial Vehicles* [42] and *Semi-Parametric Models for Visual Odometry* [43].

allow the GP framework to exploit the constraints created by vehicle dynamics and facilitate correlation on the high-dimensional input space. The MOGP framework, as described in Section 2.2.6, is capable of estimating each output using information from all inputs, but each output is still calculated individually, and thus there is no cross-dependency modelling (the resulting covariance matrix is diagonal). The calculation of these cross-dependencies would also improve the use of the estimates in filtering and SLAM scenarios, since now uncertainty is fully modelled and encoded into the covariance matrix.

Another shortcoming mentioned in the previous chapter is the algorithm's inability to generalize over different optical flow distributions. Once the GP hyperparameters have been optimized in the training stage, they are fixed and will not change during navigation, even though there is a constant flow of new information that could be used to improve the non-parametric model. As training data deviates from testing data, so does the algorithm's overall performance, even though there is also a corresponding increase in uncertainty, which maintains the validity of the solution from a probabilistic standpoint. Regions in the input space that are under-represented in the training dataset suffer the most, such as sharp turns and other uncommon behaviours. By allowing the non-parametric model to be iteratively updated, with new information being incorporated and redundant information being discarded, the algorithm would be able to gradually learn new optical flow distributions.

This chapter is devoted to address these shortcomings, and some others, in an attempt to improve the proposed method to a point in which it can be robustly used as a visual odometry solution, with results that are on par with traditional parametric solutions. It starts by providing an overview of the proposed visual odometry algorithm, with all the extensions proposed in this thesis, and it continues by exploring each of the following subjects in more detail:

- **Coupled Gaussian Processes.** The CGP (*Coupled Gaussian Processes*) is an extension to the MOGP (Multiple Output Gaussian Processes) framework in which all outputs are calculated simultaneously, based on all available input information. This technique also allows the recovery of a full covariance matrix, containing not only the uncertainty estimates for all outputs but also the cross uncertainty estimates between each individual output pair. Hence, if one particular output is well-represented in the input space, this information can be used to improve results on outputs that are under-represented, creating a more robust solution that does not require an exceedingly large training dataset to provide accurate results.

- **Temporal Dependency between Frames.** This extension was conceived as a way to increase the amount of information available as input for the GP framework, especially in the particular case of a narrow field of view (Section 3.2.3), where the entire image is considered homogeneous and therefore represented by a single optical flow vector. In this scenario, a similar optical flow distribution may be responsible for several distinct motion combinations in different degrees of freedom, creating ambiguities in the estimation process that could compromise results. By assuming that vehicle velocity changes gradually between frames, we propose using the outputs in a given timestep as part of the input vector for the next one, thus constraining the space of possible solutions to those that are similar to the ones obtained previously.

- **Incremental Updates of the Covariance Matrix.** A truly generic visual odometry algorithm should be able to provide accurate motion estimates regardless of the environment the vehicle encounters during navigation, which is clearly not the case if a fixed training dataset is used to generate the non-parametric model and optimize the hyperparameters. This shortcoming is addressed here by allowing the iterative incorporation of new data into the non-parametric model, along with the removal of data considered redundant. Assuming that the environment changes smoothly as the vehicle moves, the resulting algorithm is now able to gradually adapt to new environments and learn new (and potentially useful) optical flow distributions. The hyperparameters, and consequently the transformation function between inputs and outputs, are also iteratively updated to reflect this environment change.

- **Semi-Parametric Visual Odometry.** The GP framework eliminates the need for a geometric model by learning the transformation function from image information to vehicle motion directly from training data in a non-parametric fashion. However, geometric models have been extensively studied and are known to provide accurate results in a wide variety of situations, so there is no reason to avoid their use completely if this could lead to better estimates. We propose the combination of a geometric parametric model with a non-parametric GP model to create a *semi-parametric* solution to visual odometry (SPCGP, as in *Semi-Parametric Coupled Gaussian Processes*), where the geometric model provides an initial estimate that is then further refined by the CGP framework. The camera calibration parameters are treated as hyperparameters and learned during the training stage, thus maintaining the assumption that no traditional camera calibration is necessary.

- **Simultaneous Localization and Mapping.** The ability of the CGP framework to recover a full covariance matrix is exploited here in an extension to a SLAM scenario. A loop-closure algorithm is implemented to determine when an area is being revisited by the vehicle, and an *Exact Sparse Information Filter* (ESIF) [136] is used to keep track of all pose estimates, along with their corresponding covariance matrices. If an area is assumed revisited, these poses are updated to decrease global uncertainty, eliminating residual drifts and imposing an upper bound on uncertainty that allows for accurate localization estimates even after long periods of navigation.

The chapter then describes the experiments, presenting and discussing results obtained using the proposed method in both 2D and 3D scenarios, which testify to the various benefits gained from incorporating different extensions into the main algorithm. Experiments that address the generalization capabilities of the proposed method are also presented, testing its limits in handling deviations between training and evaluation conditions and showing how much similarity is necessary before the algorithm starts to fail and no longer provides useful estimates. Finally, the chapter concludes by providing a summary of contributions and delineating the motivation behind the implementation of the dynamic object removal algorithm described in the next chapter.

4.1 Algorithm Extensions

A simplified version of the proposed visual odometry algorithm was presented in Section 3.1, and here this simplified version is extended to include all modules described in this chapter. A complete diagram of the proposed method is shown in Fig. 4.1, where it is possible to see that most of the changes take place in the *Gaussian Process Framework* stage, which now follows the SPCGP framework principles. Prior to the beginning of navigation, the *optimized calibration parameters* and *optimized SPCGP parameters* are obtained based on training information, which now also includes the set F_{TR} containing the fundamental matrices obtained from the training images. These fundamental matrices encode the epipolar constraints that allow the geometric model to estimate camera translation and rotation between frames, which in turn acts as the *SFM estimate*, the parametric portion of this semi-parametric approach to visual odometry. A random set of hyperparameters and calibration parameters may be used as the starting point for the training process, and if an estimate is already available it can be used instead, thus increasing algorithm convergence speed. It is important to note that both camera calibration parameters and SPCGP hyperparameters are optimized simultaneously, as equal layers in the semi-parametric framework, and therefore the resulting calibration parameters may not reflect the actual camera intrinsic parameters.

Once the training process is completed, the optimized calibration parameters are used to generate the SFM estimate based on F_{12}, which serves as an initial guess that is then further refined by the *SPCGP* Inference process based on the current non-parametric model $(X, Y, F)_{NP}$, the input vector X_{12} and the optimized SPCGP hyperparameters. The final output is comprised of the mean vector Y_{12}, containing the vehicle motion estimates for all degrees of freedom, and the covariance matrix Σ_{12}, containing the auto-covariance values for all degrees of freedom and the cross-

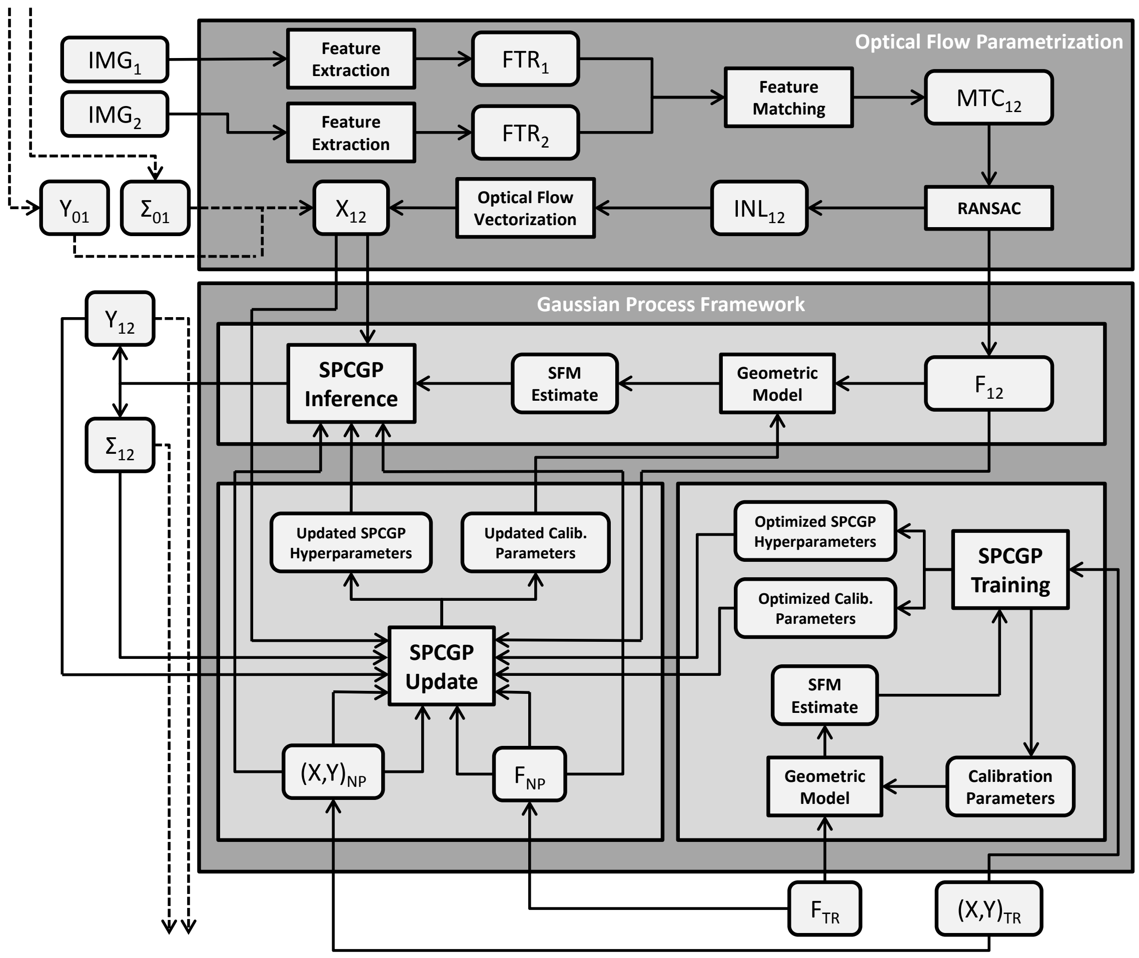


Figure 4.1 – Diagram of the proposed algorithm (final version, with all extensions). The *Optical Flow Parametrization* stage deals with the processing of a pair of images, generating the vector X_{12} and the fundamental matrix F_{12} that will serve as input for the *Gaussian Process Framework*, the second stage. The Gaussian Process Framework can be further divided into three modules: *Training*, where the optimized hyperparameters are obtained from the training dataset prior to the beginning of navigation; *Update*, where the optimized hyperparameters are further refined using new information, iteratively incorporating and removing data from the non-parametric model; and *Inference*, where the current optimized hyperparameters are used to calculate an initial SFM estimate that is then further refined by the non-parametric model, generating the mean vector Y_{12} and covariance matrix Σ_{12} from the input vector X_{12}. If temporal dependency is being used, the motion estimates Y_{01} are used to complete the input vector X_{12} and the resulting motion estimates Y_{12} are maintained to be incorporated into the input vector X_{23} from the next timestep.

covariance values between each degree of freedom. If the temporal dependency module is active (dotted lines), both output components are stored and used in the next iteration to complete the input vector X_{23}, to increase the amount of information available for the SPCGP Inference process.

Afterwards, the mean vector Y_{12}, the input vector X_{12} and the fundamental matrix F_{12} are used as input in the *SPCGP* Update process, where the non-parametric model is updated as new information becomes available. If X_{12} resides in an under-represented portion of the input space, it would be desirable to incorporate it into the non-parametric model, thus increasing accuracy in a new family of optical flow distributions. The next step is to check the values in Σ_{12}, because if the corresponding uncertainty is high this indicates that the algorithm is not confident about Y_{12}, and therefore this estimate may not be reliable enough to be incorporated into the non-parametric model. These two aspects (the benefit of incorporating X_{12} versus the inherent uncertainty in Y_{12}) are weighted, and if the result is favourable the current estimate is added as a new data point into the non-parametric point. If this incorporation makes the covariance matrix surpass a certain size, a data point must also be removed in order to keep the computational cost roughly constant. This point is selected among those in a well-represented portion of the input space, to minimize the amount of information discarded from the non-parametric model.

4.1.1 Coupled Gaussian Processes

The multiple-output Gaussian process (MOGP) derivation [10] improves on standard Gaussian processes by allowing the simultaneous modelling of T outputs, using the assumption that there is a latent correlation between them (Fig. 2.9). This correlation is quantified using a cross-covariance function, that quantifies the relationship between input points that are mapped to different outputs. This approach allows the sharing of information between different outputs, and if they are indeed correlated this sharing improves on results by constraining the solution space. However, each output is still calculated independently, and therefore there is no modelling of cross-dependencies between outputs. These cross-dependencies (the off-diagonal terms in the covariance matrix Σ) represent the uncertainty values of each output pair, indicating how much the knowledge of one output increases the information available for the estimation of the other one.

An extension to the standard MOGP derivation is presented here, where all outputs are calculated simultaneously from the same input information. This approach allows the recovery of a full covariance matrix $\mathcal{V}(\bar{\mathbf{f}})$, containing both auto and cross-dependencies between each output. The higher a particular cross-dependency value is, the more correlated the corresponding output pair will be, and by exploiting these dependencies the algorithm is capable of further constraining the solution space to reduce uncertainty and improve results. This ability to quantify the correlation between different outputs, and determine specific coupling pairs whose correlation could be exploited, gives this extension the name of *Coupled Gaussian Process* (CGP). This is the GP framework used in the proposed visual odometry algorithm, as a way to correctly model the constraints generated by vehicle dynamics, that naturally limits motion to only certain specific combinations.

As in the standard MOGP framework, the training dataset Λ is divided into T separate $\Lambda_t = \{\mathbf{x}_n, y_{n,t}\}_{n=1}^N$ datasets, each containing N observations $\mathbf{x}_n$ and their corresponding ground-truth $y_{n,t}$ information for that particular output[2]. The covariance matrix K is now defined as:

$$K = K_f \otimes \mathbf{K}_x + \Sigma_n, \tag{4.1}$$

where $\otimes$ denotes the Kronecker product, K_f is a $T \times T$ positive-definite matrix that models the correlation amplitude between each output (a two-dimensional analogue to the signal variance values $\sigma_{f,ij}^2$ in Eqs. 3.15 and 3.16) and Σ_n is a $T \times T$ matrix containing noise values. The matrix $\mathbf{K}_x$ is a $T \times T$ block matrix that encodes the

[2]In principle, each particular training dataset may be composed of a different set of observations, and may also be of different sizes. However, since in the visual odometry scenario this is generally not the case (each image has a corresponding motion estimate for all degrees of freedom), for the sake of simplicity we will assume from now on that all training datasets Λ_t contain the same observation set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.

auto and cross-covariance values between training inputs $\mathbf{x}_{n,ij}$, for each output pair, and it is given by:

$$\mathbf{K}_x = \begin{bmatrix} K_{11} & \dots & K_{1T} \\ \vdots & \ddots & \vdots \\ K_{T1} & \dots & K_{TT} \end{bmatrix}, \tag{4.2}$$

where

$$K_{ij} = \begin{bmatrix} k_{ij}(\mathbf{x}_1, \mathbf{x}_1) & \dots & k_{ij}(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k_{ij}(\mathbf{x}_N, \mathbf{x}_1) & \dots & k_{ij}(\mathbf{x}_1, \mathbf{x}_N) \end{bmatrix} \tag{4.3}$$

and k_{ij} is the auto-covariance function $k_{ii}(\mathbf{x}, \mathbf{x}')$ if $i = j$ and the cross-covariance function $k_{ij}(\mathbf{x}, \mathbf{x}')$ if $i \neq j$. The covariance functions used here are slightly different from the ones proposed previously (Eqs. 3.15 and 3.16), due to the introduction of the correlation matrix K_f, and are positive-definite (a proof can be found in [134]) and of the form:

$$k_{ii}(\mathbf{x}, \mathbf{x}') = \arcsin\left(\frac{2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}}'}{\sqrt{(1 + 2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}})(1 + 2\tilde{\mathbf{x}}'^T \Sigma \tilde{\mathbf{x}}')}}\right) \tag{4.4}$$

$$k_{ij}(\mathbf{x}, \mathbf{x}') = \frac{\arcsin\left(\frac{2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}}'}{\sqrt{(1+2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}})(1+2\tilde{\mathbf{x}}'^T \Sigma \tilde{\mathbf{x}}')}}\right)}{(|\Sigma_i||\Sigma_j|)^4 \sqrt{|\Sigma_i + \Sigma_j|}}. \tag{4.5}$$

In the equations above, $\tilde{\mathbf{x}} = \{1, x_1, \dots, x_D\}$ is an augmented vector for the input point $\mathbf{x}$ and $\Sigma = \Sigma_i(\Sigma_i + \Sigma_j)^{-1}\Sigma_j$. The hyperparameter set θ is now composed of the length-scales Σ_t for each particular output, the correlation parameters in K_f and the noise values in Σ_n. The hyperparameter sharing technique introduced in Section 3.3.2 still applies, and the training stage is conducted by optimizing the marginal likelihood function as described in Eq. 3.19. Once training is complete, the optimized hyperparameters are used for inference, which for a test point $\mathbf{x}_*$ is defined as:

$$\bar{\mathbf{f}}_* = K_s^T K^{-1} \mathbf{y} \tag{4.6}$$

$$\mathcal{V}(\bar{\mathbf{f}}_*) = K_* - K_s^T K^{-1} K_s, \tag{4.7}$$

where $\mathbf{y} = [(y_{1,1} \dots y_{1,N_1}) \dots (y_{t,1} \dots y_{t,N_t}) \dots (y_{T,1} \dots y_{T,N_T})]^T$ is a vector containing the ground-truth information for all outputs and

$$K_* = \begin{bmatrix} k_{11}(\mathbf{x}_*, \mathbf{x}_*) & \dots & k_{1T}(\mathbf{x}_*, \mathbf{x}_*) \\ \vdots & \ddots & \vdots \\ k_{T1}(\mathbf{x}_*, \mathbf{x}_*) & \dots & k_{TT}(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \tag{4.8}$$

is a matrix containing the auto-covariance values for the test point $\mathbf{x}_*$ for each output $y_{*,t}$, and the corresponding cross-covariance values for each output pair. Lastly, K_s is a T-column matrix that contains the covariance values between the test point $\mathbf{x}_*$ and the training points $\mathbf{x}_n$ for all outputs (again, we assume that all training datasets Λ_t are composed of the same set of observations), and it is defined as:

$$K_s = \begin{bmatrix} k^f_{1,1} k_{1,1}(\mathbf{x}_*, \mathbf{x}_1) & \dots & k^f_{T,1} k_{T,1}(\mathbf{x}_*, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ k^f_{1,1} k_{1,1}(\mathbf{x}_*, \mathbf{x}_N) & \dots & k^f_{T,1} k_{T,1}(\mathbf{x}_*, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k^f_{1,T} k_{1,T}(\mathbf{x}_*, \mathbf{x}_1) & \dots & k^f_{T,T} k_{T,T}(\mathbf{x}_*, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ k^f_{1,T} k_{1,T}(\mathbf{x}_*, \mathbf{x}_N) & \dots & k^f_{T,T} k_{T,T}(\mathbf{x}_*, \mathbf{x}_N) \end{bmatrix}. \tag{4.9}$$

This new inference methodology is the main contribution of CGPs over the standard MOGP framework, allowing the simultaneous recovery of all vehicle motion estimates $\bar{\mathbf{f}}_*$ and also a full covariance matrix $\mathcal{V}(\bar{\mathbf{f}}_*)$, as seen in Fig. 4.2. The CGP inference methodology comprises the core of the proposed visual odometry algorithm, and results presented later on in the experiments section (Section 4.3) testify to the improvement given by this approach over the MOGP results shown in the previous chapter. The next sections are devoted to further improve the CGP framework by introducing temporal dependencies between frames, the iterative incorporation and removal of information from the covariance matrix, and the introduction of a geometric model as the mean function for the posterior distribution (thus eliminating the standard zero mean assumption mentioned in Section 2.2.3).

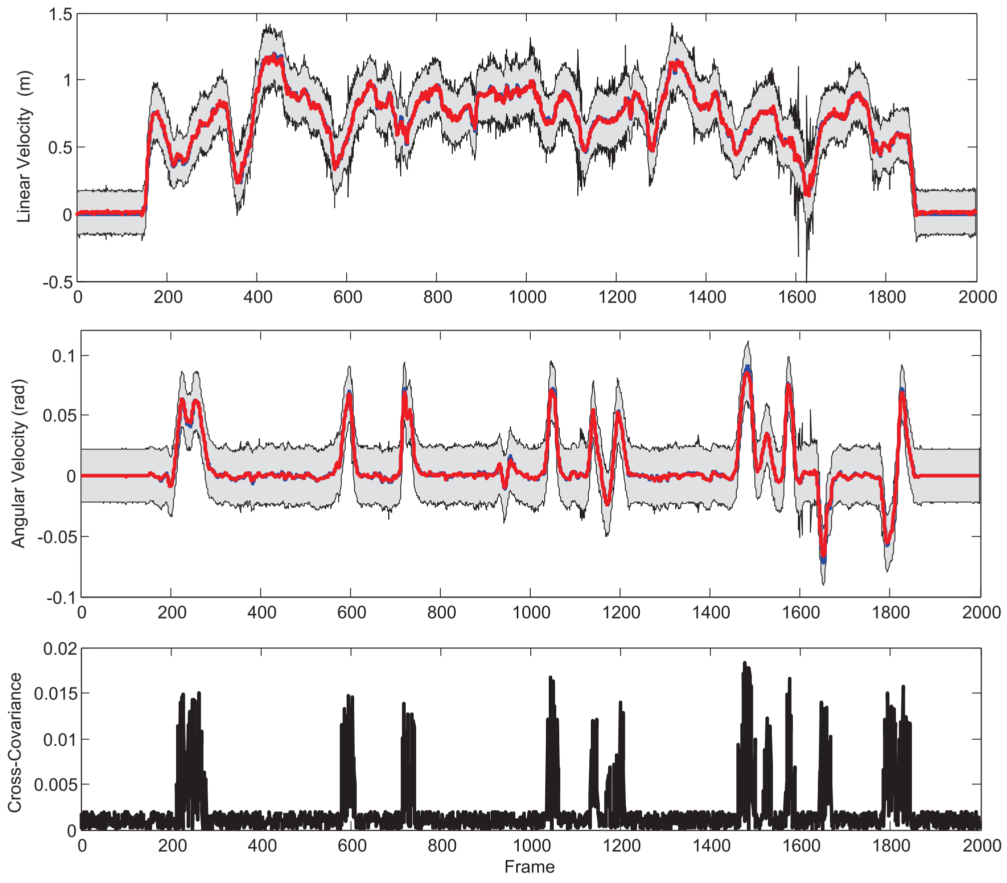


Figure 4.2 – Motion estimates obtained using the CGP framework (compare with Fig. 3.17). The bottom figure shows the cross-covariance values between linear and angular velocities, and it is possible to see that it consistently increases during rotation. This is to be expected, as it is when the vehicle turns that its dynamic constraints become more prominent, and the CGP framework is able to exploit these constraints to further improve the accuracy of its estimates even in under-represented regions of the input space.

4.1.2 Temporal Dependency between Frames

The previous section explored the nature of cross-dependencies between outputs, which is a natural assumption in visual odometry applications, where the dynamic vehicle model naturally constrains motion and correlates different degrees of freedom. However, this is not the only one, and here another type of correlation between outputs is explored: temporal dependency. It is safe to assume that a real vehicle will change its velocity in a smooth manner, without discontinuities, and therefore its motion estimates will also vary smoothly over time. A first-order temporal dependency between outputs implies that $\mathbf{y}_{k-1}$ will be correlated to $\mathbf{y}_k$, with k being the timestep for each frame. This is modelled into the CGP framework by incorporating the motion estimate $\overline{\mathbf{f}}^*_{k-1}$ into the input vector $\mathbf{x}_k$. For a test point $\mathbf{x}^*_k$ at timestep k the new augmented input vector $\mathbf{z}^*_k$ is defined as:

$$\mathbf{z}^*_k = \{\mathbf{x}^*_k, \overline{\mathbf{f}}^*_{k-1}\}. \tag{4.10}$$

The introduction of $\mathbf{z}$ as an augmented input vector does not interfere with the CGP inference methodology, other than requiring the corresponding augmentation of the length-scale matrix Σ_t to deal with the new input dimensions that were incorporated. However, this new framework disturbs the traditional training methodology because the new complete set of observations $Z = \{\mathbf{z}_n\}_{n=1}^N$ is not readily available for evaluation, since it needs to be calculated incrementally based on information obtained in the previous iteration. It is possible to use ground-truth information to complete Z, but this would generate a best-case scenario that is not consistent with real applications, where small estimation errors propagate over successive iterations.

A new training methodology is proposed here, which allows an incremental hyper-parameter optimization while maintaining temporal dependency by using motion estimates obtained using the CGP inference process to generate the input vectors $\mathbf{z}_n$ in the training dataset. This new methodology is described in details in Algorithm 4.1, and it requires the division of the training dataset Λ into two subsets, Λ^1 and Λ^2, each composed of half the training data. In the first subset, the ground-truth

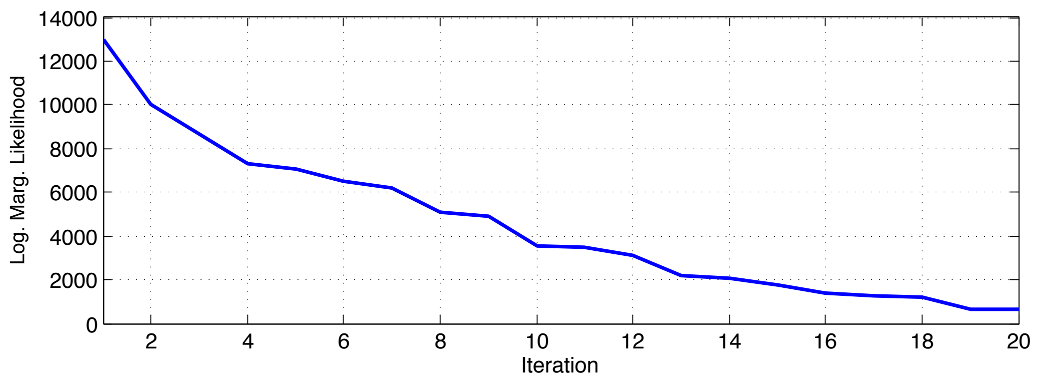
Algorithm 4.1: Temporal Dependency Training

```
Input  :  $\Lambda^1$ and $\Lambda^2$  - Training datasets
              $\theta$          - Initial hyperparameter set
Output:  $\theta$  - Optimized hyperparameter set
$likelihood\_old \leftarrow \infty$
$likelihood\_new \leftarrow 0$
while $likelihood\_new - likelihood\_old \neq 0$ do
    $likelihood\_old = likelihood\_new$
    foreach $\boldsymbol{x}_i$ in $\Lambda_1$ do
        $Z_i^1 \leftarrow (\mathbf{x}_i, \mathbf{y}_{i-1}^1)$
    end
    % Expectation step
    foreach $\boldsymbol{x}_i$ in $\Lambda_2$ do
        $\mathbf{y}_{CGP} = CGP\_INFER(Z^1, \mathbf{x}_i, \theta)$
        $Z_i^2 \leftarrow (\mathbf{x}_i, \mathbf{y}_{CGP})$
    end
    % Maximization step
    $(likelihood\_new, \theta) = CGP\_TRAIN(Z^2, \mathbf{y}^2, \theta)$
    $\Lambda^1 \leftrightarrow \Lambda^2$
end
return $\theta$
```

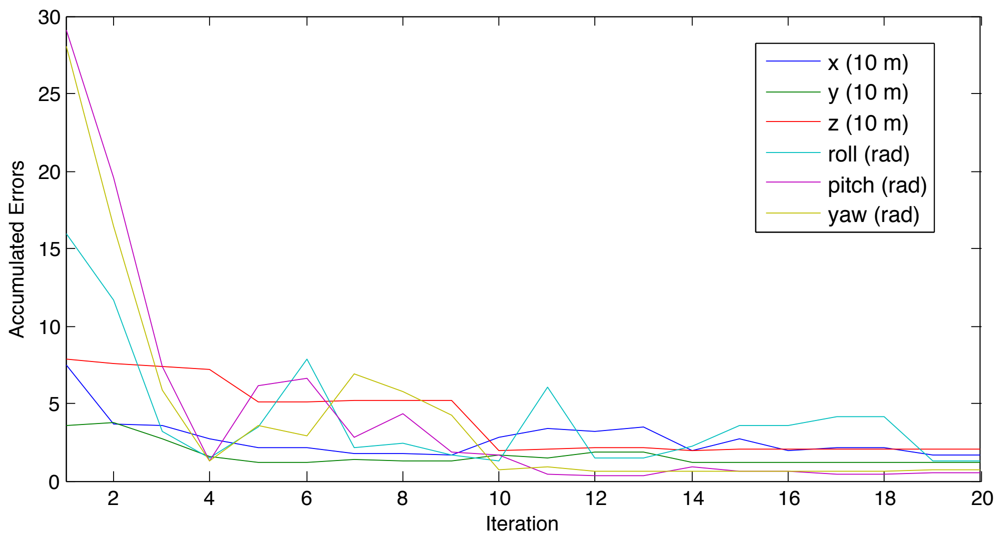
values of $\mathbf{y}^1$ are used to complete Z^1 directly (lines 4-6), in such a manner that $\mathbf{y}_{k-1}^1$ completes $\mathbf{z}_k^1$. The observation set Z^1 is then used to evaluate Z^2 iteratively (lines 8-11), according to the CGP inference methodology described previously.

Once this evaluation process is complete, the observation set Z^2 is used to optimize the hyperparameters (line 13) according to the marginal log-likelihood function (Eq. 3.19) and using a gradient-descent method. After this optimization is complete, the process is repeated with inverted subsets (Λ^1 is now used for inference and Λ^2 for training) until the cost function converges (lines 14-15). It was determined empirically that the hyperparameters assigned as length-scales for the m_{k-1} components of $\mathbf{z}_k$ should be kept from taking too low values, since this would increase the sensitivity to small errors in estimation. Also, the gradient-descent method should be limited to only a few steps, in order to increase convergence speed and discourage over-fitting to any particular iteration of the training process.

This technique resembles the *expectation-maximization* (EM) algorithm [26], in the sense that it alternates between computing motion estimates from current hyperparameters (the expectation step) and optimizing hyperparameter values using current motion estimates (the maximization step). There is no guarantee of convergence to the global minimum, so heuristic approaches for escaping local minima, such as random restart or simulated annealing, should still be considered as discussed previously. The intermediary results of this new training methodology are depicted in Fig. 4.3,



(a)



(b)

Figure 4.3 – Intermediary results of the new training methodology introduced to address temporal dependency between outputs. (a) Optimization of the marginal log-likelihood function. (b) Accumulated errors for each output and for each iteration (assuming all six degrees of freedom).

where it is possible to see a steady decrease in the cost function value for each iteration. The corresponding accumulated errors for each output are also depicted and show a steady decrease as well, even though there are some occasional increases due to the multiple-output nature of the optimization process (certain errors might increase as others decrease). This particular training methodology is particularly useful in 3D aerial visual odometry, where the narrow field of view imposed by the camera limits the information available for the CGP framework. By incorporating temporal dependencies it is possible to severely constrain the solution space and significantly improve results, as shown in Section 4.3.2.

4.1.3 Incremental Updates of the Covariance Matrix

As stated previously, the performance of the CGP framework is heavily dependent on similarities between optical flow distributions in the training and testing datasets, since inference is performed by comparing available data (and their corresponding ground-truth) with new unobserved information. As new information deviates from available data, the input space becomes under-represented and the CGP framework does not have enough samples to generate accurate estimates, even though the corresponding uncertainty increases to balance this phenomenon from a probabilistic standpoint. Since it is infeasible to produce a training dataset with all possible optical flow distributions in all possible environments, both due to sheer logistical reasons and the computational complexity of the resulting model, another solution would be to allow the incremental update of the covariance matrix, incorporating useful data and removing redundant data as they become available.

Since ground-truth information is not available during navigation, this data incorporation framework must use CGP estimates as ground-truth, obtained iteratively during the inference process. Any data incorporated should represent a previously poorly described portion of the input space, to increase the overall knowledge of the underlying function without excessive redundancy. As a new point is incorporated, if the resulting covariance matrix K surpasses a pre-determined size, another point should be removed to maintain computational complexity roughly constant. This point is selected among those that describe an already well-represented portion of the input space (and are therefore considered redundant). These rules are observed by determining the k nearest neighbours of the test point $\mathbf{x}_*$ in the input space, and performing the following three steps:

- **Uncertainty check.** To avoid the incorporation of inaccurate estimates, the test point is discarded if its inherent uncertainty Σ_* is the highest among its neighbours, of if it is higher than a certain threshold κ_u. This step assumes that the environment in which the vehicle is navigating changes smoothly, to allow the gradual incorporation of new accurate data throughout this transition and "prepare" the vehicle for the new environment.
- **Data Incorporation.** If the number of neighbours of the test point within a certain radius is lower than a certain threshold κ_i, then the point is accepted for incorporation (it represents a previously unknown portion of the input space). This incorporation process is performed according to Eqs. 4.11 and 4.12.
- **Data removal.** If the size of the covariance matrix K exceeds a certain threshold κ_r, a point must be selected for removal. This is done by selecting the point in the current non-parametric model with the highest number of neighbours (and therefore it is considered redundant). This removal process is done according to Eqs. 4.13 and 4.14.

This neighbourhood information can be implemented efficiently using structures such as *kd*-trees [85], and new points are incorporated and removed iteratively from this structure (which can also be done efficiently since only one new input point is generated at each iteration). The parameter set $\kappa = \{\kappa_u, \kappa_i, \kappa_r\}$ is selected according to the speed in which the environment is expected to change over time (κ_u) and the expected size of the covariance matrix (κ_i and κ_r). The incorporation and removal of points in the covariance matrix is conduced using *Cholesky decompositions*, a common approach in GP literature [96, 111]. The Cholesky factor is an upper triangular matrix C such that $K = C^T C$, and it allows for a faster and more numerically stable computation of the inverse K^{-1}. Assuming that the covariance matrix K and the Cholesky matrix C are defined [118] as:

$$K = \begin{bmatrix} K_{11} & \mathbf{k}_{12} & K_{13} \\ \mathbf{k}_{12}^T & k_{22} & \mathbf{k}_{23} \\ K_{13}^T & \mathbf{k}_{23}^T & K_{33} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & \mathbf{c}_{12} & C_{13} \\ \mathbf{0} & c_{22} & \mathbf{c}_{23} \\ \mathbf{0} & \mathbf{0} & C_{33} \end{bmatrix}, \tag{4.11}$$

the resulting Cholesky matrix C' obtained by marginalizing (removing) the central row and column is given by:

$$C' = \begin{bmatrix} C_{11} & C_{13}^T \\ \mathbf{0} & \gamma(C_{33}^T C_{33} + \mathbf{c}_{23}^T \mathbf{c}_{23}) \end{bmatrix}, \tag{4.12}$$

where γ is the Cholesky update operator, readily available in packages such as [28, 80] and which exploits the special structure of $\mathbf{c}_{23}^T \mathbf{c}_{23}$ to attain a computational complexity of $\mathcal{O}(n^2)$. The marginalization of K is obtained simply by removing its middle row and column. Similarly, if the covariance matrix K and the Cholesky matrix C are defined as:

$$K = \begin{bmatrix} K_{11} & K_{13} \\ K_{13}^T & K_{33} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{13}^T \\ \mathbf{0} & C_{33} \end{bmatrix}, \tag{4.13}$$

the resulting Cholesky matrix C' obtained by expanding (adding) a central row and column is given by:

$$C' = \begin{bmatrix} C_{11} & C_{11}^T \backslash k_{12} & C_{13} \\ \mathbf{0} & \sqrt{k_{22} - \mathbf{c}_{12}^T \mathbf{c}_{12}} & \frac{\mathbf{k}_{23} - \mathbf{c}_{12}^T C_{13}}{c_{22}} \\ \mathbf{0} & \mathbf{0} & \gamma(C_{33}^T C_{33} + \mathbf{c}_{23}^T \mathbf{c}_{23}) \end{bmatrix}, \tag{4.14}$$

where the operator $\backslash$ is used to indicate the solution of $\mathbf{c}_{12}$ in the equation $C_{11}^T \mathbf{c}_{12} = \mathbf{k}_{12}$, obtained by the use of backwards or forward substitution for the upper triangular matrix C_{11}. Again, the expansion of K is done directly by incorporating the middle row $\mathbf{k}_* = [\mathbf{k}_{12}^T, k_{22}, \mathbf{k}_{23}]$ and column $\mathbf{k}_*^T$, obtained by calculating the covariance between

$\mathbf{x}_*$ and all training points $\mathbf{x}_n$ (points prior to the middle are part of $\mathbf{k}_{12}$, points after the middle are part of $\mathbf{k}_{23}$, and k_{22} is the diagonal term of the new row and column). The above equations address the particular case of one single output, and the extension to multiple-outputs is done by updating each individual covariance matrix K_{ij} in the now block-matrix $\mathbf{K}$ (Eq. 4.2). It is worth noting that each point incorporated into the covariance matrix increases its overall size by T, the number of outputs, and therefore the problem of scalability becomes even more pronounced. In principle this update process may be conducted independently for each output, incorporating estimates for particular degrees of freedom that are under-represented in that specific portion of the input space and discarding estimates for others that are already well-represented, however this was not explored in this thesis.

4.1.4 Semi-Parametric Visual Odometry

The standard GP derivation presented in Section 2.2.3 assumes that the joint distribution of any finite set of samples removed from the Gaussian process will have a mean value equal to zero. This zero mean assumption can be made without any loss of generalization, by correctly normalizing the input information, and is indeed very common throughout the literature. Another way of interpreting this assumption is to imagine that there is no prior knowledge about the underlying function the GP is trying to model, and therefore the initial guess is simply zero in the entire input space. From this initial "unknown" state the non-parametric model then attempts to learn the correlation between different input points from training data and extrapolate these correlations to address new unobserved data.

There are some applications, however, where the underlying function can be roughly estimated using a parametric model, which is a much more compact and efficient way of performing regression. For example, if we know that the underlying function is roughly linear, it is possible to incorporate this knowledge into the non-parametric model by introducing a linear function $f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ as the mean vector for the GP framework. The coefficients in A and $\mathbf{b}$ are treated as hyperparameters and

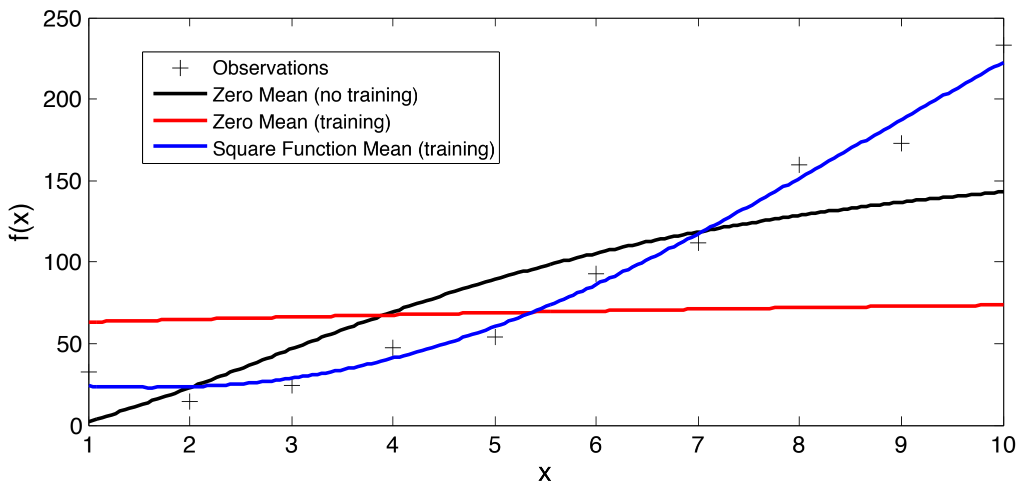


Figure 4.4 – Effects of using a mean function to estimate a quadratic function $f(x) = ax^2 + b$ with a noise value $\sim \mathcal{N}(0, 10)$. The black line shows the estimates obtained with a zero mean assumption and no training (random hyperparameters). The red line shows the estimates obtained with a zero assumption and gradient-descent training, where it is possible to see that the resulting model interprets the quadratic behaviour as noise. The blue line shows the results obtained using a quadratic function $f(x) = ax^2 + bx + c$ as the mean function, with parameters $\{a, b, c\}$ trained alongside the GP hyperparameters.

optimized as such, thus maintaining the same methodology described previously but introducing a new component that constrains the solution space according to our prior beliefs about the phenomenon at hand. The result is a *semi-parametric* model that benefits from the best of both worlds: the compactness and efficiency of a parametric model and the flexibility of a non-parametric model. In this new framework, the non-parametric model no longer has to completely estimate the underlying function and all its intricacies, since now it only has to refine the initial estimate provided by the parametric model.

The visual odometry scenario is one of such applications, where the various geometric models available for different camera configurations could be used as the mean function that is then further refined by the GP framework. We propose here the incorporation of the standard SFM algorithm[3], as described in Section 2.3.5, to generate

[3]The Mono-SLAM algorithm was also considered as the mean function, however its high computational cost would not allow the calculation of real-time estimates during navigation. Any other algorithm could in principle be readily incorporated without any further modifications to the frame-

the initial SFM estimates for each degree of freedom. The fundamental matrix F_n necessary for this geometric model is already calculated during the RANSAC stage, where outliers are removed from the initial matching set before the input vector $\mathbf{x}_n$ is generated. The calibration parameters $\{f_x, f_y, \alpha, c_x, c_y\}$, as defined in Eq. 2.99, are treated as hyperparameters and optimized alongside the GP hyperparameters, thus maintaining the assumption that no traditional camera calibration is necessary. If a different geometric model is used, any other parameter involved in the calculations may also be included as an extra hyperparameter, still eliminating the traditional calibration stage. Indeed, this proposed semi-parametric approach to visual odometry (SPCGP) can be used in conjunction with any of the current available visual odometry algorithms, introducing an extra non-parametric layer that refines their initial estimates based on training data. Since the incremental aspect of visual odometry is highly sensitive to errors, the flexibility of the GP framework allows the modelling of nuances in the underlying function that a strictly parametric model struggles with.

The training methodology remains the same, except for the introduction of the SFM calibration parameters as a new set of hyperparameters to be optimized alongside the CGP hyperparameters. These hyperparameters are optimized iteratively as described in Fig. 4.1, where an initial SFM estimate is calculated, refined by the CGP framework, and then used to generate a new set of hyperparameters in an attempt to minimize the cost function. The marginal log-likelihood still serves as the cost function, however it has to be slightly modified to incorporate the presence of a non-zero mean vector $\bar{\mathbf{f}}$. The new marginal log-likelihood cost function is now of the form:

$$\mathcal{L}(\mathbf{y}|\Lambda, \theta) = -\frac{1}{2}\ln|K| - \frac{1}{2}\boldsymbol{\epsilon}^T K^{-1}\boldsymbol{\epsilon} - \frac{N}{2}\ln(2\pi), \tag{4.15}$$

where $\boldsymbol{\epsilon} = (\mathbf{y} - m(\mathbf{x}))$ is an error vector that quantifies the distance between the initial SFM estimates $m(\mathbf{x})$ and the ground-truth values $\mathbf{y}$. If $\boldsymbol{\epsilon}$ is small, this means that the initial SFM estimate is already accurate and there is no need for further improvement. On the other hand, if $\boldsymbol{\epsilon}$ is large, the non-parametric model takes over and tries to compensate the difference using training data. The inference methodology

work.

(Eqs. 4.6 and 4.7) is also slightly modified to incorporate the presence of a non-zero mean vector:

$$\bar{\mathbf{f}}_* = m(\mathbf{x}_*) + K_s^T K^{-1}(\mathbf{y} - m(\mathbf{x})) \tag{4.16}$$

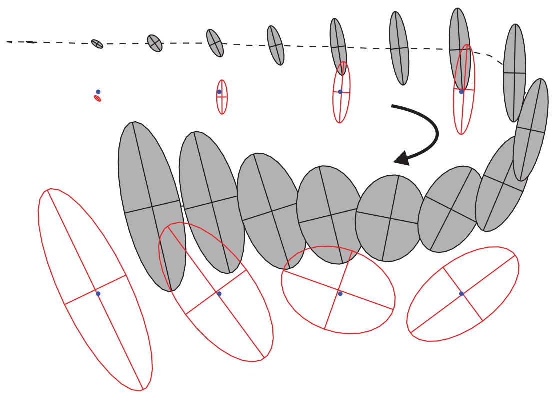
$$\mathcal{V}(\bar{\mathbf{f}}_*) = \kappa - K_s^T K^{-1} K_s. \tag{4.17}$$

By incorporating the mean function into the inference methodology we assure that, as testing data deviates from training data, the outputs will converge to the SFM estimates, as the non-parametric model will have less information to further improve results. Hence, this methodology also decreases the impact of dissimilarities between training and testing data, as now the SPCGP framework will, in the worst case scenario, be at least as accurate as the geometric model used. As these similarities start to emerge, the non-parametric model becomes able to exploit them as a way to refine the initial estimates and further improve results.

4.2 Simultaneous Localization and Mapping

Up to this point, the problem of visual odometry has been addressed from an incremental perspective, meaning that motion estimates are obtained independently from each other. Even the temporal dependency between frames established in Section 4.1.2 does not enforce any correlation among estimates from different timesteps, but rather generates an extra constraint in the input space that the CGP framework is able to exploit as a way to resolve ambiguities and improve results. Because of that, any imprecision in these motion estimates will propagate to the next iterations, generating an error component (drift) that quickly accumulates to compromise global localization results. Any improvement in such estimates will serve only to delay, but never prevent, this error accumulation, and therefore purely incremental localization methods are inherent flawed in long-term navigation. A reliable long-term navigation algorithm should be able to provide absolute localization estimates, that are not dependent on the vehicle's previous states.

This section explores the extension of the proposed visual odometry algorithm into a *Simultaneous Localization and Mapping* (SLAM) framework [25, 65, 82], where absolute localization estimates are obtained by incrementally building a map of the environment during navigation. The vehicle pose $\mathbf{x}_t$ at each instant is given by a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)$, where $\boldsymbol{\mu}_t$ is the current pose estimate and Σ_t is its corresponding covariance matrix. Intuitively, Σ_t should increase over time due to drift, causing the vehicle to be less and less certain of its true position and orientation in regards to a global coordinate system. Similarly, each landmark $\mathbf{m}_i$ observed is stored as a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, where $\boldsymbol{\mu}_i$ is the landmark's position estimate in the environment and Σ_i is its corresponding covariance matrix. This covariance matrix Σ_i has two components: one generated by the sensor model used by the vehicle to observe the environment (which is roughly constant and given by Σ_s) and one generated by the vehicle's own pose uncertainty at the moment in which the landmark was observed (which increases monotonically over time due to drift). An example of this increase in uncertainty is shown in Fig. 4.5a, where the grey ellipses


(a)

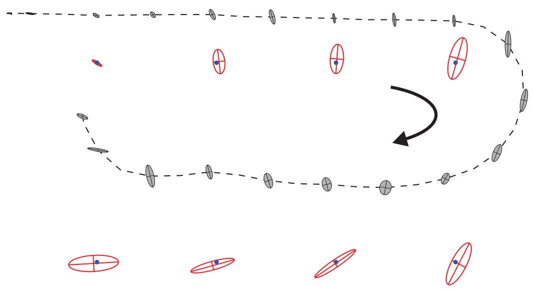

(b)

Figure 4.5 – Example of uncertainty reduction in SLAM. The vehicle starts moving (top left portion of the image) and its pose uncertainty increases monotonically over time (grey ellipses), alongside the uncertainty of observed landmarks (red ellipses). The last timestep before loop-closure is shown in (a), and (b) shows the effects of this loop-closure on the uncertainty ellipses after the first landmark is revisited. The loop-closure information is used to remove all drift accumulated since the beginning of navigation, and this information is then propagated backwards throughout the entire trajectory, causing a global decrease in vehicle and landmark uncertainty.

represent vehicle uncertainty and the red ellipses represent landmark uncertainty. The vehicle starts at the top left corner of the image, and as it moves clock-wise its pose uncertainty increases over time due to drift, which in turns increases the uncertainty of the landmarks it observes during navigation.

However, when there is a *loop-closure* (i.e. a previously observed landmark is revisited), the vehicle is able to generate a pose estimate that is not dependent on its previous state $\mathbf{x}_{t-1}$, but rather on the landmark's position $\mathbf{m}_i$ as it is stored on the map. Since Σ_t increases monotonically over time due to drift, it is natural to assume that $\Sigma_i < \Sigma_t$, because the landmark $\mathbf{m}_i$ was observed on a previous timestep $t_i < t$. It is possible then to combine these two estimates into a new more accurate estimate $\mathbf{x}'_t$, eliminating the drift error accumulated between t_i and t and estimating the vehicle's position more accurately. Additionally, since the vehicle poses in different timesteps are correlated (they were obtained incrementally), it is also possible to propagate this information backwards over the entire trajectory, causing a global decrease in vehicle pose uncertainty. Lastly, since landmark uncertainty is correlated to the vehicle's own uncertainty at the time they were observed, this information can be used to globally decrease landmark position uncertainty. An example of this global decrease in uncertainty is shown in Fig. 4.5b, where the vehicle is revisiting the first landmark it observed at the beginning of navigation (top left corner). This landmark estimate does not contain any drift error component, and the vehicle is capable of using this information to refine its pose estimate and retroactively decrease the uncertainty of the entire trajectory, alongside the uncertainty of all observed landmarks up to this point.

This section describes a SLAM algorithm based on the works of [33] and [125], where a *Exact Sparse Information Filter* (ESIF) is used to track all vehicle and landmark poses over time, along with their corresponding uncertainties and the various correlations between estimates. The use of information filters, instead of the more traditional covariance filters [139], is beneficial due to its natural sparsity, with most of its non-diagonal elements being very close to zero. It is shown in [130] how to approximate these elements to be exactly zero, thus creating a near-constant time solution to the SLAM problem, where new information can be incorporated and recovered very efficiently. This concept is further explored in [136], where weak robot-landmark correlations are actively broken to enforce a desired level of sparsity. By relocalizing the robot within the map it is possible to produce estimates that are both globally and locally accurate relatively to a non-sparse solution.

4.2.1 Marginalization and Conditioning

We start by defining $\boldsymbol{\zeta}_t$ as a random vector distributed according to a multivariate Gaussian probability distribution, such that $\boldsymbol{\zeta}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)$, where $\boldsymbol{\mu}_t$ is the mean vector and Σ_t is the covariance matrix. Expanding the quadratic term within the Gaussian exponential, we arrive at an equivalent representation for the multivariate distribution $\mathcal{N}^{-1}(\boldsymbol{\eta}_t, \Lambda_t)$:

$$\begin{aligned} p(\boldsymbol{\zeta}_t) &= \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t) \\ &= \frac{1}{\sqrt{|2\pi\Sigma_t|}} \exp\left(-\frac{1}{2}(\boldsymbol{\zeta}_t - \boldsymbol{\mu}_t)^T \Sigma_t^{-1} (\boldsymbol{\zeta}_t - \boldsymbol{\mu}_t)\right) \\ &= \frac{1}{\sqrt{|2\pi\Sigma_t|}} \exp\left(-\frac{1}{2}(\boldsymbol{\zeta}_t^T \Sigma_t^{-1} \boldsymbol{\zeta}_t - 2\boldsymbol{\mu}_t^T \Sigma_t^{-1} \boldsymbol{\zeta}_t + \boldsymbol{\mu}_t^T \Sigma_t^{-1} \boldsymbol{\mu}_t)\right) \\ &= \frac{e^{\frac{1}{2}\boldsymbol{\mu}_t^T \Sigma_t^{-1} \boldsymbol{\mu}_t}}{\sqrt{|2\pi\Sigma_t|}} \exp\left(-\frac{1}{2}\boldsymbol{\zeta}_t^T \Sigma_t^{-1} \boldsymbol{\zeta}_t + \boldsymbol{\mu}_t^T \Sigma_t^{-1} \boldsymbol{\zeta}_t\right) \\ &= \frac{e^{\frac{1}{2}\boldsymbol{\eta}_t^T \Lambda_t^{-1} \boldsymbol{\eta}_t}}{\sqrt{|2\pi\Lambda_t^{-1}|}} \exp\left(-\frac{1}{2}\boldsymbol{\zeta}_t^T \Lambda_t \boldsymbol{\zeta}_t + \boldsymbol{\eta}_t^T \boldsymbol{\zeta}_t\right) \\ &= \mathcal{N}(\boldsymbol{\eta}_t, \Lambda_t). \end{aligned} \tag{4.18}$$

The canonical form of the Gaussian distribution above is completely parametrized by the information vector $\boldsymbol{\eta}_t$ and the information matrix Λ_t, which are related to the mean vector and covariance matrix as follows:

$$\Lambda_t = \Sigma_t^{-1} \qquad \boldsymbol{\eta}_t = \Sigma_t^{-1}\boldsymbol{\mu}_t. \tag{4.19}$$

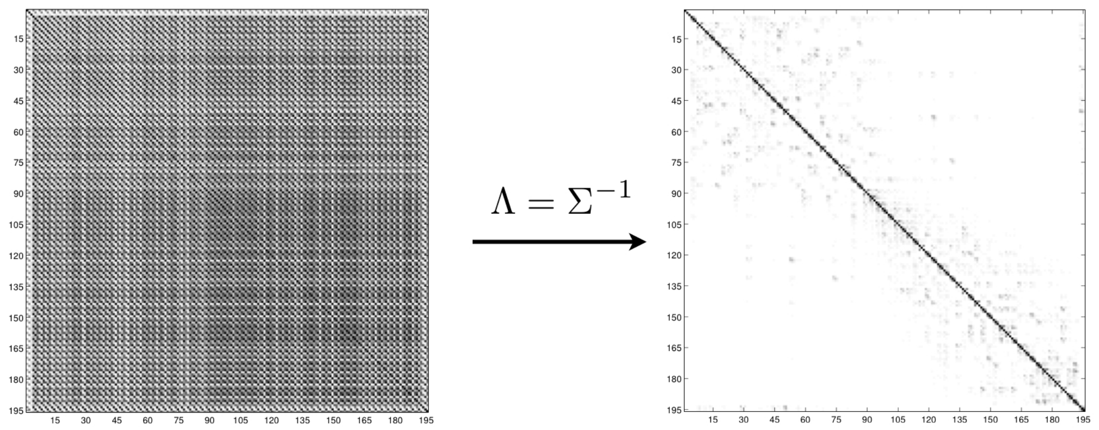


Figure 4.6 – Sparsity of the information matrix Λ in comparison to the covariance matrix Σ.

This canonical parametrization for the multivariate Gaussian distribution is the dual form of the standard derivation in regards to the marginalization and conditioning operations, as demonstrated in Table 4.1. Marginalizing over variables with the standard form is simple, since it involves the removal of the corresponding elements from the mean vector and covariance matrix. However, the same operation for the canonical form involves calculating a Schur complement and is computationally costly. The opposite is true when calculating the conditional from the joint distribution: it is complex with the standard form and simple with the canonical parametrization.

An advantageous property of the canonical parametrization is that the information matrix provides an explicit representation for the structure of the corresponding *Gaussian Markov Random Field* (GMRF) [99]. This property follows from the factorization of a general Gaussian probability density:

$$
\begin{aligned}
p(\boldsymbol{\zeta}) &\propto \exp\left(-\frac{1}{2}\boldsymbol{\zeta}^T \Lambda \boldsymbol{\zeta} + \boldsymbol{\eta}^T \boldsymbol{\zeta}\right) \\
&= \prod_i \exp\left(\frac{1}{2}(\lambda_{ii}\zeta_i^2 - \eta_i \zeta_i)\right) \prod_{i,j(i\neq j)} \exp\left(-\frac{1}{2}\zeta_i \lambda_{ij} \zeta_j\right) \\
&= \prod_i \Psi_i(\zeta_i) \prod_{i,j(i\neq j)} \Psi(\zeta_i, \zeta_j),
\end{aligned}
\tag{4.20}
$$

where

$$\Psi_i(\zeta_i) = \exp\left(\frac{1}{2}(\lambda_{ii}\zeta_i^2 - \eta_i\zeta_i)\right) \tag{4.21}$$

$$\Psi_{ij}(\zeta_i, \zeta_j) = \exp\left(\frac{1}{2}\zeta_i\lambda_{ij}\zeta_j\right) \tag{4.22}$$

are the node and edge potentials for the corresponding undirected graph. Random variable pairs with zero off-diagonal elements in the information matrix (i.e. $\lambda_{ij} = 0$) have an edge potential $\Psi_{ij}(\zeta_i, \zeta_j) = 1$, signifying the absence of a link between the nodes representing the variables. Conversely, non-zero shared information indicates that there is a link joining the corresponding nodes with a strength of the edge proportional to λ_{ij}. As the link topology for an undirected graph explicitly captures the conditional dependencies among variables, so does the structure of the information matrix. The presence of off-diagonal elements equal to zero implies that the corresponding variables are conditionally independent, given the remaining states. Interestingly, the same conclusion can be obtained from a simple analysis of the conditioning operation for the information form. As stated in Table 4.1, conditioning a pair of random variables $\boldsymbol{\alpha} = [\boldsymbol{\zeta}_i^T \boldsymbol{\zeta}_j^T]^T$ on the remaining states $\boldsymbol{\beta}$ involves extracting the $\Lambda_{\alpha\alpha}$ sub-block from the information matrix. When there is no shared information between $\boldsymbol{\zeta}_i$ and $\boldsymbol{\zeta}_j$, the sub-block $\Lambda_{\alpha\alpha}$ is diagonal, as is its inverse (i.e. the covariance matrix). Conditioned upon $\boldsymbol{\beta}$, the two variables are uncorrelated and therefore conditionally independent, or $p(\boldsymbol{\zeta}_i, \boldsymbol{\zeta}_j|\boldsymbol{\beta}) = p(\boldsymbol{\zeta}_i|\boldsymbol{\beta})p(\boldsymbol{\zeta}_j|\boldsymbol{\beta})$.

	Marginalization $p(\boldsymbol{\alpha}) = \int p(\boldsymbol{\alpha}, \boldsymbol{\beta})d\boldsymbol{\beta}$	Conditioning $p(\boldsymbol{\alpha}\|\boldsymbol{\beta}) = p(\boldsymbol{\alpha}, \boldsymbol{\beta})/p(\boldsymbol{\beta})$
Covariance Form	$\boldsymbol{\mu} = \boldsymbol{\mu}_\alpha$ $\Sigma = \Sigma_{\alpha\alpha}$	$\boldsymbol{\mu}' = \boldsymbol{\mu}_\alpha + \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta)$ $\Sigma' = \Sigma_{\alpha\alpha} - \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}\Sigma_{\beta\alpha}$
Information Form	$\boldsymbol{\eta} = \boldsymbol{\eta}_\alpha - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\boldsymbol{\eta}_\beta$ $\Lambda = \Lambda_{\alpha\alpha} - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\Lambda_{\beta\alpha}$	$\boldsymbol{\eta}' = \boldsymbol{\eta}_\alpha - \Lambda_{\alpha\beta}\boldsymbol{\beta}$ $\Lambda' = \Lambda_{\alpha\alpha}$

$$p(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_\alpha \\ \boldsymbol{\mu}_\beta \end{bmatrix}, \begin{bmatrix} \Sigma_{\alpha\alpha} & \Sigma_{\alpha\beta} \\ \Sigma_{\beta\alpha} & \Sigma_{\beta\beta} \end{bmatrix}\right) = \mathcal{N}^{-1}\left(\begin{bmatrix} \boldsymbol{\eta}_\alpha \\ \boldsymbol{\eta}_\beta \end{bmatrix}, \begin{bmatrix} \Lambda_{\alpha\alpha} & \Lambda_{\alpha\beta} \\ \Lambda_{\beta\alpha} & \Lambda_{\beta\beta} \end{bmatrix}\right)$$

Table 4.1 – Duality between Covariance and Information Filters. Covariance filters are more efficient in marginalization, whereas Information filters are more efficient in Conditioning.

4.2.2 State Augmentation

We describe here the method for state augmentation, which is how the state vector $\boldsymbol{\zeta}_t$ is augmented to receive a new vehicle pose estimate $\mathbf{x}_t$. This operation occurs whenever there is a new state that is deemed worth storing. In our framework, a new state in principle could be added at each iteration, augmenting the state vector to include the new vehicle pose estimate. However, since this would quickly generate an excessively large and mostly redundant state vector, we choose to add a new state only when the Euclidean distance between the previous stored vehicle pose estimate and the current one is larger than a certain threshold d.

Adding a Delayed-State

Assume for the moment that the estimate at time t is described by the following distribution expressed in both covariance and information form:

$$\begin{aligned} p(\mathbf{x}_t, M|\mathbf{z}^t, \mathbf{u}^t) &= \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{x_t} \\ \boldsymbol{\mu}_M \end{bmatrix}, \begin{bmatrix} \Sigma_{x_t x_t} & \Sigma_{x_t M} \\ \Sigma_{M x_t} & \Sigma_{MM} \end{bmatrix}\right) \\ &= \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\eta}_{x_t} \\ \boldsymbol{\eta}_M \end{bmatrix}, \begin{bmatrix} \Lambda_{x_t x_t} & \Lambda_{x_t M} \\ \Lambda_{M x_t} & \Lambda_{MM} \end{bmatrix}\right), \end{aligned} \tag{4.23}$$

where $\mathbf{M}$ is the environment map, $\mathbf{z}^t$ is the entire measurement history and $\mathbf{u}^t$ is the entire control history. The environment map $\mathbf{M}$ is used here in a general sense, and could include a set of sparse landmarks, or raw sensor data, or any information that may be used to enforce a loop-closure. For now this is not important, and we will focus on what happens when the state vector $\boldsymbol{\zeta}$ is augmented to include the time-propagated vehicle state $\mathbf{x}_{t+1}$, generating the distribution $p(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1})$ that can be factored as:

$$\begin{aligned} p(\mathbf{x}_{t+1}, \mathbf{x}_t, &\mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) = \\ &= p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{M}, \mathbf{z}^t, \mathbf{u}^{t+1}) p(\mathbf{x}_t, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) \end{aligned}$$

$$= p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_{t+1})p(\mathbf{x}_t, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^t). \tag{4.24}$$

Eq. 4.24 is obtained by factoring the posterior into the product of a probabilistic state-transition multiplied by the prior, using the common assumption that the robot state evolves according to a first-order Markov process. The general non-linear discrete-time Markov vehicle motion model is then given by Eq. 4.25, and the first-order linearized form is given by Eq. 4.26, where F is the Jacobian evaluated at $\boldsymbol{\mu_{x_t}}$ and $\mathbf{w}_t \propto \mathcal{N}(\mathbf{0}, Q)$ is a white noise process.

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mathbf{w}_t \tag{4.25}$$

$$\approx f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) + F(\mathbf{x}_t + \boldsymbol{\mu}_{x_t}) + \mathbf{w}_t \tag{4.26}$$

Augmentating the Information Form

Before obtaining the information form of the augmented state vector, it is necessary to obtain its covariance form. This is done according to the first-order Markov factorization described in Eq. 4.24, and is of the form:

$$p(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) = \mathcal{N}(\boldsymbol{\mu}'_{t+1}, \Sigma'_{t+1}) \tag{4.27}$$

$$\boldsymbol{\mu}'_{t+1} = \begin{bmatrix} f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) \\ \boldsymbol{\mu}_{x_t} \\ \boldsymbol{\mu}_M \end{bmatrix} \quad \Sigma'_{t+1} = \begin{bmatrix} F\Sigma_{x_t x_t}F^T + Q & F\Sigma_{x_t x_t} & F\Sigma_{x_t M} \\ \Sigma_{x_t x_t}F^T & \Sigma_{x_t x_t} & \Sigma_{x_t M} \\ \Sigma_{M x_t}F^T & \Sigma_{M x_t} & \Sigma MM \end{bmatrix}. \tag{4.28}$$

The lower 2×2 sub-block of Σ'_{t+1} corresponds to the covariance between the delayed-state element $\mathbf{x}_t$ and the map $\mathbf{m}$, and has remained unchanged from the prior. The first row and column, on the other hand, contain the cross-covariances associated with the time-propagated vehicle state $\mathbf{x}_{t+1}$, which includes the vehicle motion model. This covariance form can now be transformed into the information form, which requires the inversion of the 3×3 block-covariance matrix Σ'_{t+1} and is given by:

$$p(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) = \mathcal{N}^{-1}(\boldsymbol{\eta}'_{t+1}, \Lambda'_{t+1}) \tag{4.29}$$

$$\begin{aligned}\boldsymbol{\eta}'_{t+1} &= \begin{bmatrix} Q^{-1}\left(f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) - F\boldsymbol{\mu}_{x_t}\right) \\ \boldsymbol{\eta}_{x_t} - F^T Q^{-1}\left(f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) - F\boldsymbol{\mu}_{x_t}\right) \\ \boldsymbol{\eta}_M \end{bmatrix} \\ \Lambda'_{t+1} &= \begin{bmatrix} Q^{-1} & -Q^{-1}F & 0 \\ -F^T Q^{-1} & \Lambda_{x_t x_t} + F^T Q^{-1} F & \Lambda_{x_t M} \\ 0 & \Lambda_{M x_t} & \Lambda M M \end{bmatrix}.\end{aligned} \tag{4.30}$$

4.2.3 Measurement Updates

One of the most attractive properties of the information form is that measurement updates are done in constant-time [130], in contrast to the covariance form which is of quadratic complexity per update. Assume the following general non-linear measurement model and its first-order linearized form:

$$\begin{aligned}\mathbf{z}_t &= h(\boldsymbol{\zeta}_t) + \mathbf{v}_t \\ &\approx h(\bar{\boldsymbol{\mu}}_t) + H(\boldsymbol{\zeta}_t - \bar{\boldsymbol{\mu}}_t) + \mathbf{v}_t,\end{aligned} \tag{4.31}$$

where $\boldsymbol{\zeta}_t$ is the predicted state vector distributed according to $\boldsymbol{\zeta}_t \sim \mathcal{N}(\bar{\boldsymbol{\mu}}_t, \bar{\Sigma}_t) \equiv \mathcal{N}_{-1}(\bar{\boldsymbol{\eta}}_t, \bar{\Lambda}_t)$, $\mathbf{v}_t$ is the white measurement noise $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, R)$ and H is the Jacobian evaluated at $\bar{\boldsymbol{\mu}}_t$. The covariance form update [139] requires the computation of the Kalman gain and the updating of $\bar{\boldsymbol{\mu}}_t$ and $\bar{\Sigma}_t$:

$$K = \bar{\Sigma}_t H^T \left(H\bar{\Sigma}_t H^T + R\right)^{-1} \tag{4.32}$$

$$\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + K\left(\mathbf{z}_t - h(\bar{\boldsymbol{\mu}}_t)\right) \tag{4.33}$$

$$\Sigma_t = (I - KH)\,\bar{\Sigma}_t\,(I - KH)^T + KRK^T. \tag{4.34}$$

This calculation modifies all elements in the covariance matrix, resulting in a quadratic computational complexity per update. In contrast, the corresponding information form is given by:

$$\begin{aligned}\boldsymbol{\eta}_t &= \bar{\boldsymbol{\eta}}_t + H^T R^{-1}\left(\mathbf{z}_t - h(\bar{\boldsymbol{\mu}}_t) + H\bar{\boldsymbol{\mu}}_t\right)\\ \Lambda_t &= \bar{\Lambda}_t + H^T R^{-1} H.\end{aligned} \tag{4.35}$$

4.2.4 Motion Prediction

Motion prediction corresponds to a time propagation of the vehicle state from time t to time $t+1$. Expressions for the information form of the augmented distribution $p(\mathbf{x}_{t+1}, \mathbf{x}_t, \mathbf{M}|\mathbf{z}^T, \mathbf{u}^{t+1})$, containing the time predicted vehicle state $\mathbf{x}_{t+1}$ and its previous state $\mathbf{x}_t$, are given in Eq. 4.30. To derive the time-propagated distribution $p(\mathbf{x}_{t+1}, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1})$, it is necessary to marginalize out the previous state $\mathbf{x}_t$. Going back to Table 4.1 for the marginalization of a Gaussian distribution in the information form we arrive at:

$$p(\mathbf{x}_{t+1}, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) = \int p(\mathbf{x}_{t+1}, \mathbf{x}, \mathbf{M}|\mathbf{z}^t, \mathbf{u}^{t+1}) d\mathbf{x}_t = \mathcal{N}^{-1}(\bar{\boldsymbol{\eta}}_{t+1}, \bar{\Lambda}_{t+1}) \tag{4.36}$$

$$\begin{aligned}\bar{\boldsymbol{\eta}}_{t+1} &= \begin{bmatrix} Q^{-1}\left(f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) - F\boldsymbol{\mu}_{x_t}\right) \\ \boldsymbol{\eta}_M \end{bmatrix} - \begin{bmatrix} -Q^{-1}F \\ \Lambda_{Mx_t} \end{bmatrix} \Omega^{-1} \boldsymbol{\eta}^{\star}_{x_t} \\ &= \begin{bmatrix} Q^{-1}F\Omega^{-1}\boldsymbol{\eta}_{x_t} + \Psi\left(f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) - F\boldsymbol{\mu}_{x_t}\right) \\ \boldsymbol{\eta}_M - \Lambda_{Mx_t}\Omega^{-1}\boldsymbol{\eta}^{\star}_{x_t} \end{bmatrix} \\ \bar{\Lambda}_{t+1} &= \begin{bmatrix} Q^{-1} & 0 \\ 0 & \Lambda_{MM} \end{bmatrix} - \begin{bmatrix} -Q^{-1}F \\ \Lambda_{Mx_t} \end{bmatrix} \Omega^{-1} \begin{bmatrix} -F^TQ^{-1} & \Lambda_{x_tM} \end{bmatrix} \\ &= \begin{bmatrix} \Psi & Q^{-1}F\Omega^{-1}\Lambda_{x_tM} \\ \Lambda_{Mx_t}\Omega^{-1}F^TQ^{-1} & \Lambda_{MM} - \Lambda_{Mx_t}\Omega^{-1}\Lambda_{x_tM} \end{bmatrix},\end{aligned} \tag{4.37}$$

where

$$\boldsymbol{\eta}^{\star}_{x_t} = \boldsymbol{\eta}_{x_t} - F^T Q^{-1} \left(f(\boldsymbol{\mu}_{x_t}, \mathbf{u}_{t+1}) - F\boldsymbol{\mu}_{x_t} \right) \tag{4.38}$$

$$\Omega = \Lambda_{x_t x_t} + F^T Q^{-1} F \tag{4.39}$$

and

$$\begin{aligned}\Psi &= Q^{-1} - Q^{-1} F \Omega^{-1} F^T Q^{-1} \\ &= Q^{-1} - Q^{-1} F \left(F^T Q^{-1} F + \Lambda_{x_t x_t} \right)^{-1} F^T Q^{-1} \\ &= \left(Q + F \Lambda^{-1}_{x_t x_t} F^T \right)^{-1}. \end{aligned} \tag{4.40}$$

4.2.5 State Recovery

The information form of the Gaussian distribution is parametrized by its information vector $\boldsymbol{\eta}_t$ and matrix Λ_t. However, the expressions for motion prediction (Eq. 4.37) and measurement update (Eq. 4.35) still require sub-elements from the mean vector $\boldsymbol{\mu}_t$, so that the non-linear models in Eqs. 4.26 and 4.31 can be linearized. Therefore, in order for the information form to be a computationally efficient parametrization for delayed-states, it is necessary to find a way to easily recover portions of the mean vector. Fortunately, this can be done by exploiting the sparse structure of the information matrix Λ_t. Two different approaches are described here: the direct approach, where the entire state estimate is recovered, and the more efficient approach, where only the relevant portion of the state estimate is recovered.

Full State Recovery

The naive recovery of the entire state estimate involves matrix inversion and is therefore of cubic complexity, eliminating any efficiency gained by using the information form over the covariance form. Fortunately, it turns out that the recovery of the state mean $\boldsymbol{\mu}_t$ can be posed more efficiently as the solution of a sparse, symmetric, positive-definite linear system of equations:

$$\Lambda_t \boldsymbol{\mu}_t = \boldsymbol{\eta}_t, \tag{4.41}$$

which can be solved via the classic iterative method of *conjugate gradients* [116]. In general, conjugate gradients can solve this system in n iterations with cost $\mathcal{O}(n)$ per iteration, with a maximum cost of $\mathcal{O}(n^2)$ that could be a lot smaller if the initialization is suitable. Additionally, since the state mean $\boldsymbol{\mu}_t$ typically does not change significantly with each measurement update (excluding key events such as loop-closure), this relaxation can take place over multiple timesteps using a fixed number of iterations per update [29]. The problem with this approach is that convergence, and therefore an optimal state recovery, is not guaranteed. Other techniques [36] propose computational complexity reduction by subsampling poses and performing the relaxation over multiple spatial resolutions.

Partial State Recovery

A key observation about the expressions for motion prediction (Eq. 4.37) and measurement update (Eq. 4.35) is that they only require the knowledge of subsets of the state mean vector $\boldsymbol{\mu}_t$. Because of that, instead of always solving for the complete state mean vector, it is possible to partition Eq. 4.41 into two sets of coupled equations:

$$\begin{bmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} = \begin{bmatrix} \boldsymbol{\eta}_a \\ \boldsymbol{\eta}_b \end{bmatrix}. \tag{4.42}$$

This partitioning of $\boldsymbol{\mu}_t$ into two subsets allows for the sub-optimal solution for local portions of the state vector in constant-time. By holding the current estimate for $\boldsymbol{\mu}_b$ fixed, Eq. 4.42 can be solved for an estimate of $\boldsymbol{\mu}_a$ as such:

$$\hat{\boldsymbol{\mu}}_a = \Lambda_{aa}^{-1} \left(\boldsymbol{\eta}_a - \Lambda_{ab} \hat{\boldsymbol{\mu}}_b \right). \tag{4.43}$$

The equation above provides a method for recovering an estimate $\hat{\boldsymbol{\mu}}_a$ of $\boldsymbol{\mu}_a$, given that the initial estimate $\hat{\boldsymbol{\mu}}_b$ is a decent approximation of $\boldsymbol{\mu}_b$. In particular, it is used to provide an accurate approximation for recovering the state mean during motion prediction and measurement updates. Since the vehicle state is only serially connected to the map, Λ_{ab} has only one non-zero block-element, and therefore Eq. 4.43 can be solved in constant time.

4.3 Experimental Results

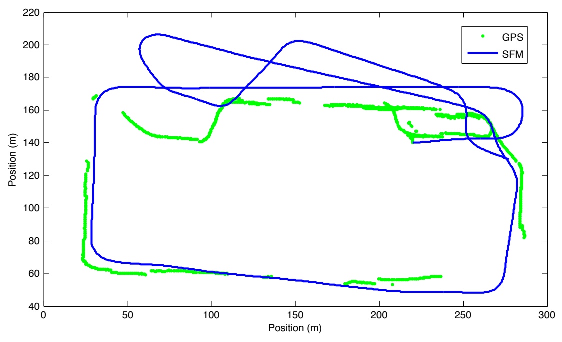
In the previous chapter results were presented using a basic version of the visual odometry algorithm proposed in this thesis, containing only the minimum stages necessary to generate motion estimates from optical flow information based on a training dataset. This section evaluates the proposed visual odometry algorithm, now including all extensions described in this chapter as a way to address the shortcomings mentioned previously. The CGP framework (Section 4.1.1) allows the correct modelling of cross-dependencies between outputs, exploiting constraints in vehicle dynamics to generate more accurate motion estimates. The temporal dependency (Section 4.1.2) increases the amount of information available for inference, and is used in the 3D aerial experiments due to the camera's narrow field of view, generating ambiguity in optical flow distributions. The online update of the covariance matrix (Section 4.1.3) allows the visual odometry algorithm to gradually adapt to new environments, by incorporating new information as it becomes available and discarding redundant information to maintain computational cost roughly constant. The SPCGP framework (Section 4.1.4) improves on the CGP inference process by incorporating a geometric model as the mean function, providing an initial estimate that is then further refined by the non-parametric model. The SLAM framework (Section 4.2) takes advantage of the full covariance matrix recovered by the CGP framework to generate absolute pose estimates, allowing the removal of drift accumulated during navigation by recognizing when a determined area is revisited.

Initially, the same experiments from the previous chapter are repeated, both in 2D and 3D environments, adding the proposed extensions. Comparisons, both qualitative and quantitative, are provided as a way to evaluate the improvements generated by the proposed extensions on the final motion estimates. These results testify to their ability to address the shortcomings inherent to the basic algorithm described in the previous chapter and provide a much more reliable and robust approach to visual odometry. Experiments testing the proposed algorithm's ability to generalize to different conditions are presented and discussed, including changes in camera configuration and environment while navigating. Extreme tests are conducted to test the limit of such generalization ability, showing how much similarity between training and testing datasets is necessary before the algorithm starts to fail (i.e. the uncertainty becomes too high to generate any useful estimates). Finally, the section concludes with a brief discussion on the impact of dynamic objects in visual odometry, which is the motivation for the automatic segmentation of dynamics objects algorithm described in the next chapter.

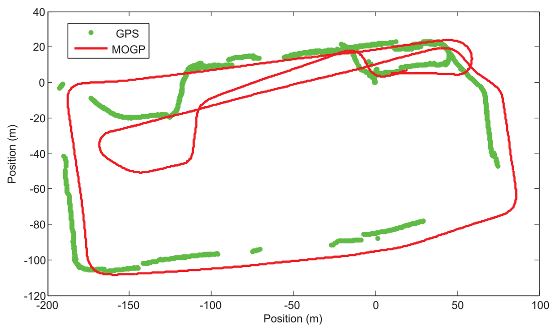
4.3.1 2D Experiments

The same training and testing datasets depicted in Fig. 3.14, collected by a ground vehicle navigating in an urban environment with a single pin-hole camera, are used to evaluate the improvements generated by the proposed extensions. For convenience, Figs. 4.7a and 4.7b show the results presented in the previous chapter: the localization estimates obtained using the structure from motion (SFM) algorithm described in Section 2.3.5 and the localization estimates obtained using the basic algorithm (MOGP) described in Section 3.3. The SFM estimates were adjusted manually to account for the inability to recover scale using a purely geometric single camera approach, while the MOGP framework is capable of recovering scale to a high degree of precision by exploring similarities between training and testing data (ground-truth information was, as before, obtained from ICP based on laser data). In both cases, it is clear that the algorithm struggles mostly with angular drift, which quickly accumulates to compromise global localization results.

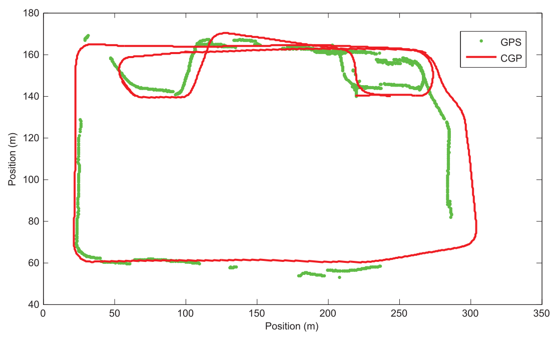
Localization results obtained using the CGP framework without the incorporation of the semi-parametric model (assuming $\bar{\mathbf{f}}_* = 0$) are depicted in Fig. 4.7c, where the first noticeable aspect is that scale is also recovered to a high degree of precision. Also, angular drift is much less pronounced, allowing the vehicle to correctly return to its initial pose even after a trajectory of roughly 2 km. We attribute this improvement over the MOGP framework to the modelling of cross-dependencies between outputs, that allows the algorithm to use linear information to improve angular estimates and vice-versa. Vehicle turns are under-represented in the training dataset, simply because in real situations a vehicle mostly drives forward in a straight road, and by exploiting these cross-dependencies it is possible to constrain the solution space and generate better estimates even in under-represented areas (see Fig. 4.2 for the motion estimates used to generate these localization results).



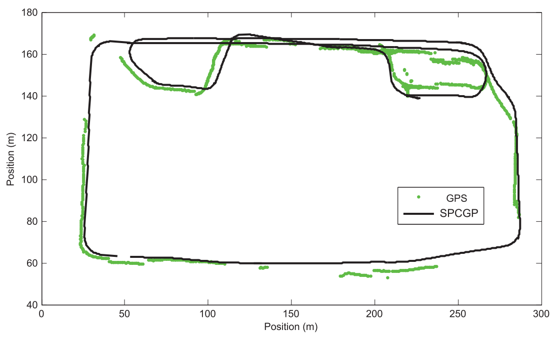
(a)



(b)



(c)



(d)

Figure 4.7 – Localization results using different methods. (a) Structure From Motion algorithm. (b) Multiple-Output Gaussian Processes. (c) Coupled Gaussian Processes. (d) Semi-Parametric Coupled Gaussian Processes.

Method	**Trans. Error (rmse) (10^{-2} m)**	**Rot. Error (rmse) (10^{-2} rad)**
It. Closest Point	2.92 ± 4.70	0.06 ± 0.14
Struct. Motion	9.75 ± 12.12	0.23 ± 0.16
Mono-SLAM	4.24 ± 2.25	0.11 ± 0.09
MOGP	5.82 ± 9.21	0.12 ± 0.19
CGP	5.74 ± 8.18	0.07 ± 0.08
SPCGP	5.12 ± 7.49	0.05 ± 0.07
SPCGP + SLAM	5.98 ± 6.54	0.03 ± 0.05

Table 4.2 – Linear and angular errors per frame for different methods in ground experiments (with ICP + ESIF as ground-truth).

Finally, localization results obtained using the SPCGP framework are depicted in Fig. 4.7d, where we can see that the incorporation of the geometric model was able to marginally improve results, both in scale (especially on the bottom street) and angular motion (especially on the top street, that is revisited by the vehicle). This marginal improvement is attributed to the use of the same vehicle and similar environments for training and testing, that creates an ideal scenario for the CGP framework that leaves little space for improvement. The impact of the SPCGP framework becomes more apparent as training and testing data deviate from each other, because under these conditions the mean function has more weight on the final estimates. This will be explored further during the generalization experiments, where the algorithm's ability to deal with such dissimilarities is evaluated in a variety of different situations.

A quantitative comparison of such approaches is presented in Table 4.2, in terms of *root mean square error* (rmse) per frame. The ground-truth for such comparisons was obtained using laser-based ICP estimates integrated into the ESIF algorithm described in Section 4.2. As expected, ICP has the lowest translational error, because distances can be measured directly from a laser scanner. Even with manual scale adjustment, the SFM estimates show the highest translational error, and all GP-based estimates performed similarly in the scale recovery aspect. Rotational errors, on the other hand, decreased significantly with the introduction of the GP framework, and continued to decrease consistently with the incorporation of cross-dependency modelling and a geometric model as the mean function. Even though ICP has a

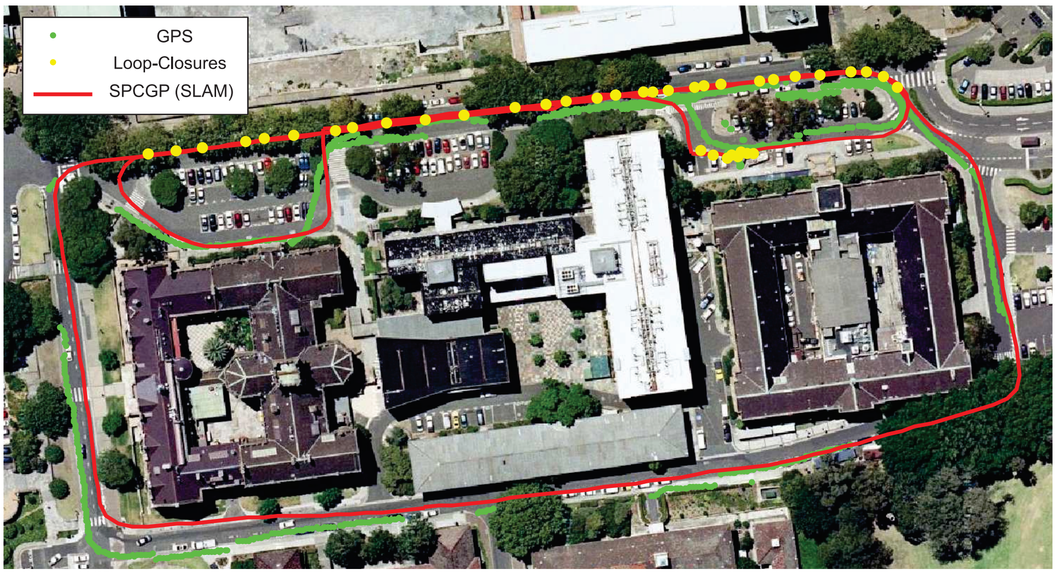


Figure 4.8 – Localization results from the proposed visual odometry algorithm incorporated into a SLAM framework.

rotational error comparable to the SPCGP framework, its variance shows that this error is not spread evenly throughout the entire trajectory, but rather concentrated in only a few frames (Fig. 3.14b), whereas the CGP framework is able to smooth out these errors and generate more consistent results, without any large localized discrepancies.

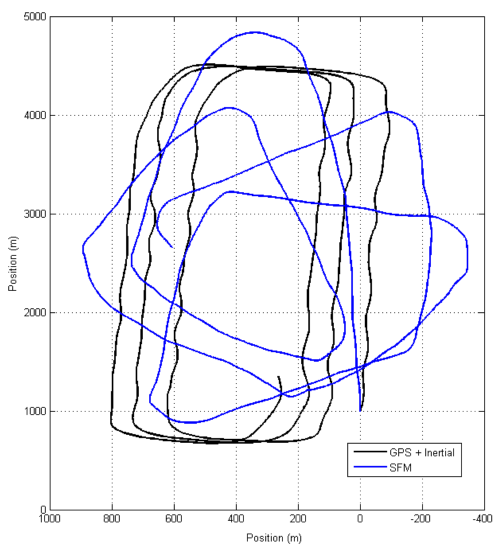
The SPCGP estimates were also incorporated into a SLAM framework, where all vehicle poses are tracked during navigation and a loop-closure algorithm was implemented to recognize when an area is revisited. The loop-closure process is done by matching features from the current frame with features from previous frames (downsampled by a factor of 5 for speed purposes, without impacting results), and an area is assumed revisited if the number of successful matches is higher than a given threshold. The localization results obtained using this framework are shown in Fig. 4.8, where it is possible to see how the loop-closure algorithm was able to correctly recognize the second pass over the top street and use this information to correct residual misalignments in this area. The loop-closure algorithm was also capable of recognizing when the vehicle returned to its starting position, connecting these two points in the state vector and minimizing the effects of drift in the entire trajectory.

4.3.2 3D Experiments

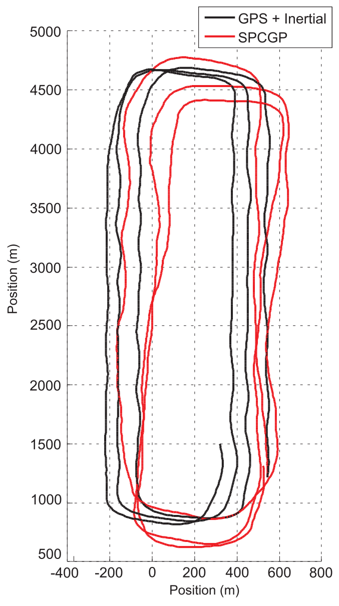
For the 3D experiments, the same dataset introduced in the previous chapter, obtained from a UAV flying over a deserted area with a camera pointing downwards, was also used to evaluate the improvement generated by the proposed extensions. It can be seen from the results in Fig. 3.18 that this dataset is much more challenging for a visual odometry algorithm, due to the small and inconsistent overlapping areas between frames, the ambiguity in optical flow distribution and the narrow field of view in the camera. To address these shortcomings, the temporal dependency extension described in Section 4.1.2 was introduced into the SPCGP framework, and the results are depicted in Fig. 4.9, alongside results obtained by using the structure from motion algorithm as described in Section 2.3.5.

As expected, a combination of accumulated error and lack of matching features generated a drift over time that could not be avoided, however it is clear that the SPCGP framework was able to improve significantly over the results obtained using only the geometric model. The absolute scale was recovered to a high degree of precision, and the overall shape of the trajectory was also recovered, without any missing corners or changes in the plane of navigation. In Fig. 4.9b it is possible to see the cyclical changes in altitude during flight, ranging from 80 to 100 meters, which poses a challenge for the GP framework as a regression tool due to the difficulty in separating what is a trend and should be modelled and what is noise and should be ignored. Interestingly, the use of temporal dependencies between frames generated a "smooth and delay" effect as a response to sudden variations, because of the proximity constraint imposed to outputs in subsequent timesteps.

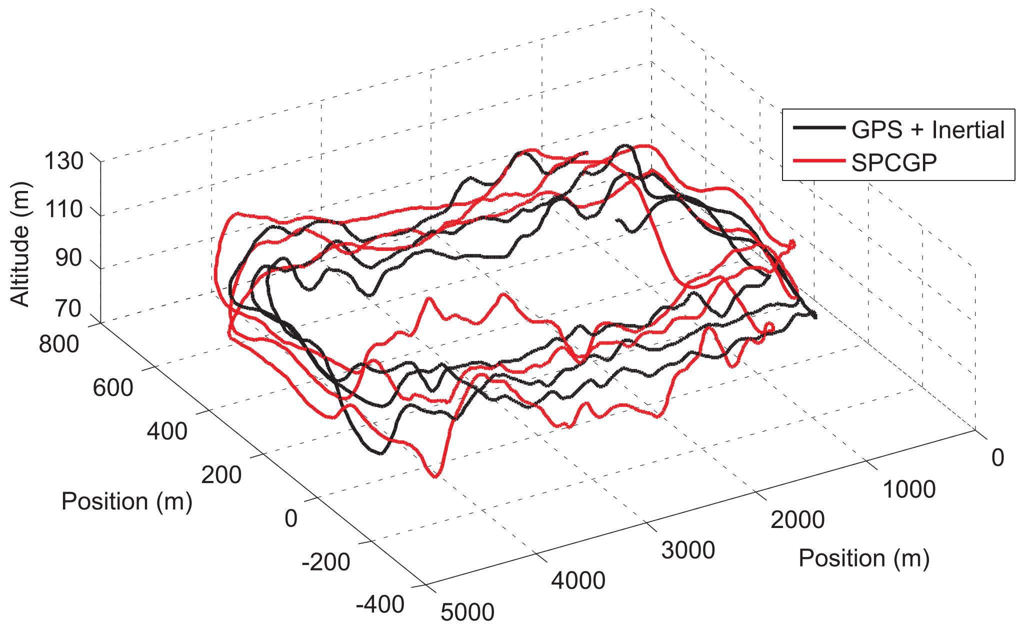
A quantitative comparison of these results is presented in Table 4.3, in terms of *root mean square error* (rmse) per frame based on GPS and inertial information. From this table we can see the same progression as shown in Table 4.2, which depicts the errors in 2D experiments. The purely geometric approach is not capable of recovering scale and therefore has the highest translational error, while all GP-based approaches have similar translational errors regardless of further extensions to the basic algorithm. Rotational error, on the other hand, decreased significantly with the introduction



(a)



(b)



(c)

Figure 4.9 – Localization results in 3D environments, using the SPCGP framework and temporal dependency between frames. (a) Geometric model with automatic camera calibration. (b) Top view. (c) Isometric view.

Task	SFM	MOGP	CGP	SPCGP
X	1384.10 ± 25.72	20.47 ± 0.1552	8.49 ± 0.0668	8.11 ± 0.0727
Y	453.56 ± 5.76	6.84 ± 0.0541	5.95 ± 0.0472	5.71 ± 0.0269
Z	325.50 ± 6.69	10.16 ± 0.0806	10.23 ± 0.0812	9.89 ± 0.0714
Roll	11.48 ± 0.56	0.69 ± 0.0056	0.66 ± 0.0053	0.47 ± 0.0051
Pitch	5.09 ± 0.01	0.35 ± 0.0027	0.26 ± 0.0021	0.18 ± 0.0025
Yaw	19.07 ± 0.55	0.41 ± 0.0032	0.33 ± 0.0027	0.25 ± 0.0021

Table 4.3 – Root Mean Square linear ($10^{-2}\ m$) and angular ($10^{-2}\ rad$) errors per frame for different methods in aerial experiments (with IMU + GPS as ground-truth).

of the GP framework and continued to decrease consistently with the incorporation of cross-dependency modelling and a geometric model as the mean function. The wide variation in overlapping areas precluded the detection of revisited areas by the loop-closure algorithm, and therefore the SPCGP results were not incorporated into a SLAM framework. The fusion of this SPCGP information with GPS could lead to further improvements in the results, by introducing absolute pose estimates to eliminate drift accumulated during the incremental visual odometry estimation.

4.3.3 Generalization Experiments

This section focuses on testing the generalization ability of the proposed visual odometry algorithm in regards to changes between training and testing datasets. As a non-parametric regression technique, the GP framework essentially depends on similarities between available information and new, unobserved information to uncover patterns that can be used to provide accurate estimates, assuming the same underlying function. Thus, as the optical flow distributions that the vehicle encounters during navigation deviate from those available for the non-parametric inference, uncertainty increases and the resulting estimates will be less accurate. This chapter described two techniques that are capable of minimizing the influence of such dissimilarities: the online update of the covariance matrix (Section 4.1.3) and the SPCGP framework (Section 4.1.4). By incorporating new information into the non-parametric model, it is possible to gradually learn new useful optical flow distributions and adapt to new environments seamlessly. A geometric model is not affected by dissimilarities between training and testing data, since its constraints depend solely on the camera configuration, and so the SPCGP framework will provide reasonable results even if the non-parametric model is unable to further refine the initial estimates.

Even so, dissimilarities between training and testing conditions still affect negatively the performance of the proposed visual odometry algorithm, and here this impact is measured in several different situations. Basically, this dissimilarity in optical flow distributions can be attributed to three distinct causes: changes in the environment, changes in camera configuration and changes in vehicle dynamics. These three scenarios are addressed here in the particular case of 2D navigation, and both qualitative and quantitative results are presented. Initially, the impact of changes in the environment is measured by training the non-parametric model in an urban environment and testing it in an off-road environment, composed mostly of trees and open areas. The impact of changes in camera configuration is then measured by training the non-parametric model with one camera and testing it with a different camera in the same environment. Lastly, the impact of changes in vehicle dynamics is measured by training the non-parametric model with one vehicle in particular and testing it with a different robotic platform. The section concludes by providing a study on gradual increases in dissimilarity, showing when the algorithm starts to fail and is no longer capable of providing useful estimates.

Changing Environments

To test the proposed algorithm's ability to generalize over different environments, a new dataset was collected using the same vehicle (Fig. 3.13a) and camera configuration, but now in a public park (the Victoria Park dataset, a common benchmark used in the SLAM literature). This dataset is composed of 4000 images and cover a trajectory of roughly 4 km, in which the vehicle navigated mostly over grass terrain. The same urban training dataset from Fig. 3.14a was used for initial optimization, and there was no further training conducted using information from the new environment. The localization results obtained using the SPCGP framework are shown in

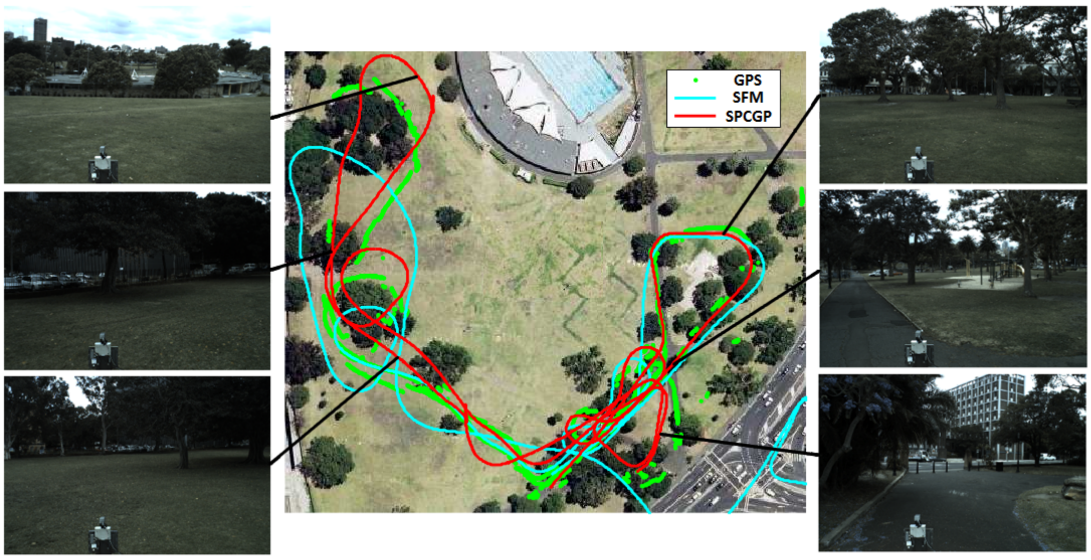


Figure 4.10 – SPCGP localization results in ground experiments with different environments.

Fig. 4.10, along with sample images used as input for the algorithm. The green dots represent GPS information and the blue line represents the localization results obtained using the structure-from-motion algorithm, with automatic camera calibration and manual scale adjustment. As expected, the SFM algorithm provides reasonably accurate estimates at the beginning of navigation, however drift quickly accumulates and compromises global localization results towards the end of the trajectory.

The localization results obtained using the SPCGP framework are represented by the red line, and even though this is also an incremental technique we can see that the proposed approach was capable of reducing the effects of drift over time, maintaining consistency in estimates throughout the entire trajectory. We attribute this improvement to the semi-parametric aspect of the SPCGP framework, that allows the non-parametric model to focus on the nuances of the training data, while the parametric model provides an initial estimate that already contains the overall shape of the underlying function. Although the structures the vehicle encounters during navigation are quite different (buildings and cars vs. trees and open fields), the optical flow distributions are similar enough to provide useful information that the non-parametric model can use to further refine results. A quantitative comparison

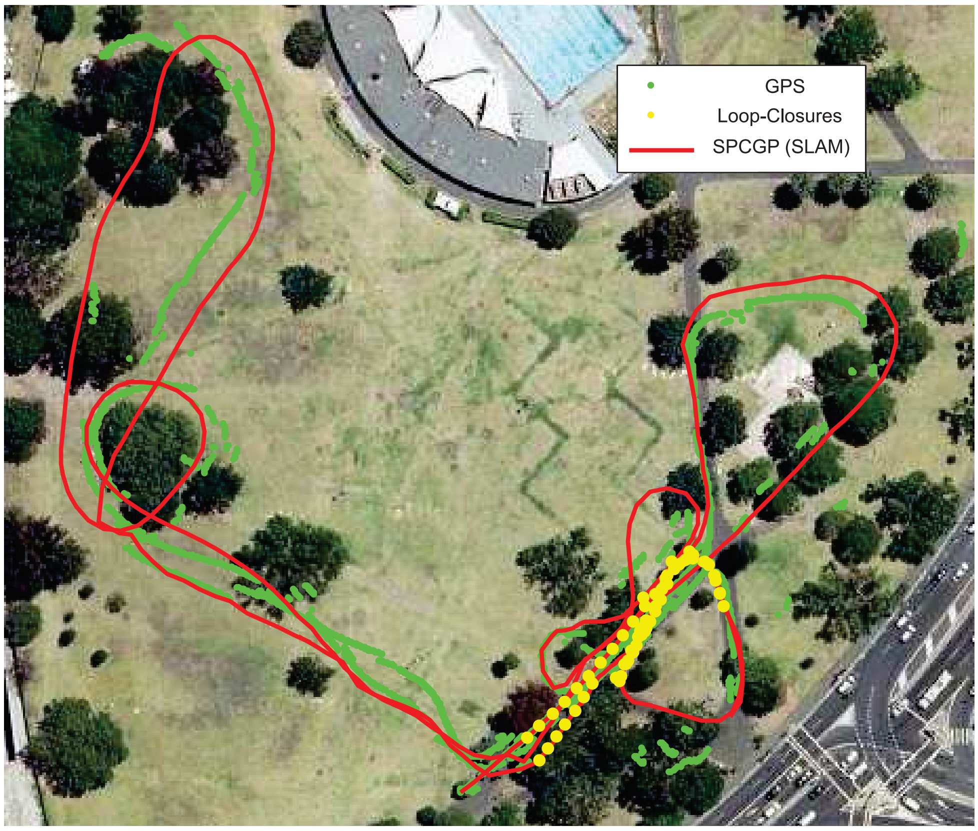


Figure 4.11 – Localization results in ground experiments with different environments, in a SLAM framework.

of these results with other generalization experiments is given in Table 4.4, showing that translational and rotational errors are indeed higher than when a similar environment is used, however they are still smaller than the ICP and SFM estimates. This decrease in overall performance should become more prominent as the optical flow patterns available for training differ more and more from those encountered during evaluation (i.e. lack of structures around the vehicle, greater proximity to objects, radically different shapes).

These localization results were also incorporated into a SLAM framework as described in Section 4.2, with a loop-closure algorithm based on feature matching between frames. The results are depicted in Fig. 4.11, where we can see that the loop-closures (yellow dots) occurred as the vehicle returned to the beginning of its trajectory. This information allowed the retroactive correction of localization estimates at the left portion of the image, reducing misalignments and increasing accuracy in a global scale. These results testify to the algorithm's ability to generalize over different environments without further training, a valuable attribute that a visual odometry algorithm should have in order to be truly useful in real applications. Experiments with online updates of the covariance matrix are conducted later on, as a way to stress test the proposed algorithm and determine how much similarity is necessary before it starts to fail.

Changing Cameras

Here we explore the impact on the localization results of changing the camera from which the images are acquired. This was done in order to verify the SPCGP framework's ability to deal with variations in camera parameters, as well as variations in optical flow distributions that are not caused solely by different structures in the environment. The same training dataset from Fig. 3.14a is used, and for testing the same trajectory from Fig. 3.14b is adopted, however now the images are obtained using a second camera placed on the same vehicle. The new camera has a lower resolution of 640x480 (which was then downsampled to the same 384x252 as the training

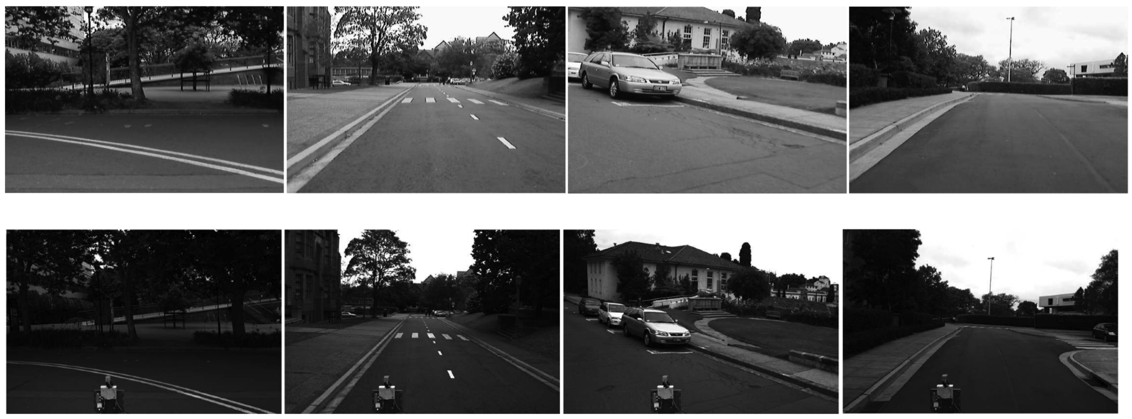


Figure 4.12 – Examples of images taken at the same vehicle position with different cameras. The first row corresponds to the original camera and the second row corresponds to the new camera (the new camera can be seen in the images captured by the original camera). The displacement between cameras is approximately 1.4 m horizontally and 0.5 m vertically, with a pitch change of roughly 10º.

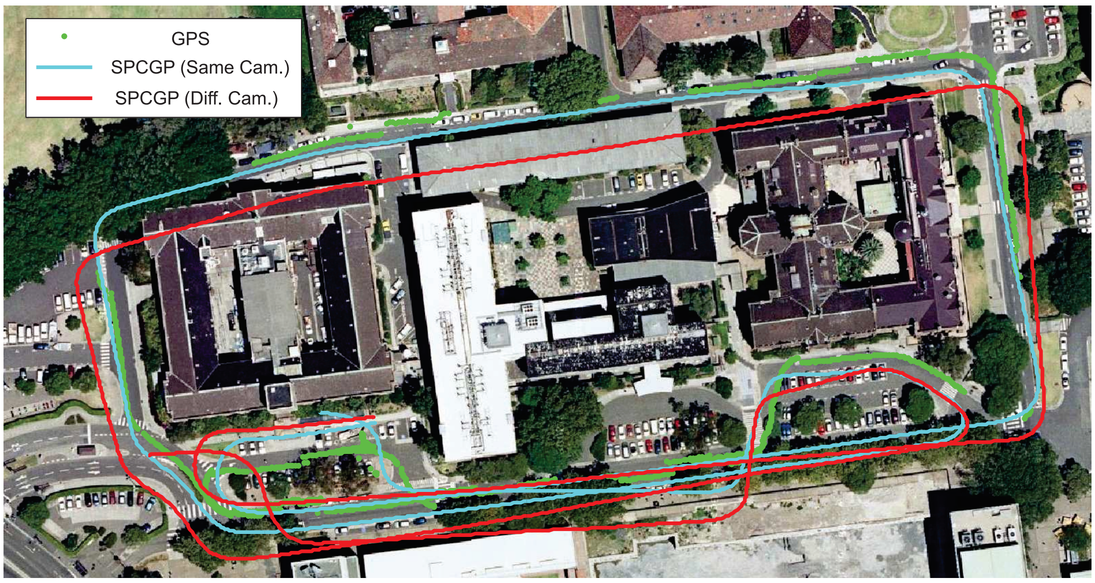


Figure 4.13 – SPCGP localization results in ground experiments with different cameras.

images) and was positioned in such a way that it captures the same portion of the environment as the other camera, but from a different perspective (see Fig. 4.12). Again, no further training was conducted once the optimized camera parameters and GP hyperparameters were obtained from the training dataset.

The localization results obtained using the SPCGP framework are presented in Fig. 4.13, both using the same camera (blue line) and different cameras (red line) for training and testing. The first noticeable aspect in these results is the impact that changing cameras has on scale recovery, which is noticeably less accurate. This is to be expected, since a GP's ability to recover scale in visual odometry from a monocular configuration is dependent on optical flow similarities between training and testing data. The inference process assumes that the environment reacts in a predictable manner to vehicle motion, and is able to extrapolate scale information based on ground-truth data. If the camera changes this assumption is weakened, since the environment now reacts differently to vehicle motion due to new geometric constraints that were not modelled during training. The rotational error also increases, however not as much as translational error, since angular motion is not subject to scaling and can therefore still be modelled accordingly using training information.

Scenario	**Trans. Error (rmse) (10^{-2} m)**	**Rot. Error (rmse) (10^{-2} rad)**
It. Closest Point	2.92 ± 4.70	0.06 ± 0.14
Struct. Motion	9.75 ± 12.12	0.23 ± 0.16
Same Configuration	5.12 ± 7.49	0.05 ± 0.07
Different Environment	6.57 ± 8.07	0.09 ± 0.09
Different Camera	8.42 ± 9.96	0.11 ± 0.19

Table 4.4 – Root Mean Square linear and angular errors per frame for each output in ground generalization experiments.

These assumptions are confirmed by the quantitative results in Table 4.4, which show that changes in camera configuration have a bigger impact on localization results than changes in the environment. Also, it confirms that translational errors are indeed more affected by such changes, whereas rotational errors also increase but not so significantly. Interestingly, we can see that these increases in error are accompanied by an increase in uncertainty, showing that these results, even though less accurate, are still valid from a probabilistic standpoint and can be used in conjunction with other sensors to provide robust estimates. The translational and rotational errors obtained from these two generalization experiments are still lower than those obtained using the SFM algorithm, indicating that the SPCGP framework still outperforms traditional visual odometry techniques even with unforeseen changes in the environment or camera configuration.

Changing Vehicles

As a final generalization experiment, the SPCGP framework was tested using a completely different robotic platform (Fig. 3.13c), also equipped with a single camera, GPS and inertial sensors that provide ground-truth information. This new configuration, other than changing the environment and visual sensor, also introduces a different cinematic model that changes between the training and inference stages. The new evaluation dataset is composed of 14500 images obtained in a highly dynamic urban environment (downtown Sydney during work hours), over a trajectory of roughly 10 km. Throughout this trajectory the vehicle interacted normally with pedestrians and other moving vehicles, and experienced several changes in environment structures and lighting conditions (some examples of images collected for this dataset are shown in Fig. 4.14).

Initially, we present the results obtained using half of this dataset for training and half for testing (thus maintaining the same camera configuration and vehicle dynamics in a similar environment) in Fig. 4.15a. Even though the overall shape of the trajectory is maintained, it is possible to see a significant amount of drift occurring in specific places, which quickly compromises global localization results. We attribute this localized drift to the presence of dynamic objects in the environment, that generate optical flow patterns that do not correspond to vehicle motion and therefore should have been discarded during the outlier removal process. The influence of dynamic objects in visual odometry is further explored in the next chapter, where a novel technique for automatic dynamic object segmentation and removal is presented.

Afterwards, localization results obtained using the urban dataset in Fig. 3.14a for training (thus changing all three components: camera configuration, vehicle dynamics and environment structures) are depicted in Fig. 4.15b. These results show that the proposed visual odometry algorithm indeed suffers from dissimilarities between training and testing datasets, and if they are radically different performance could decrease up to a point in which no further useful estimates can be obtained. To determine the progression of this decrease in performance with the increase in dissimilarities, a series of tests were conducted using different percentages of information from each

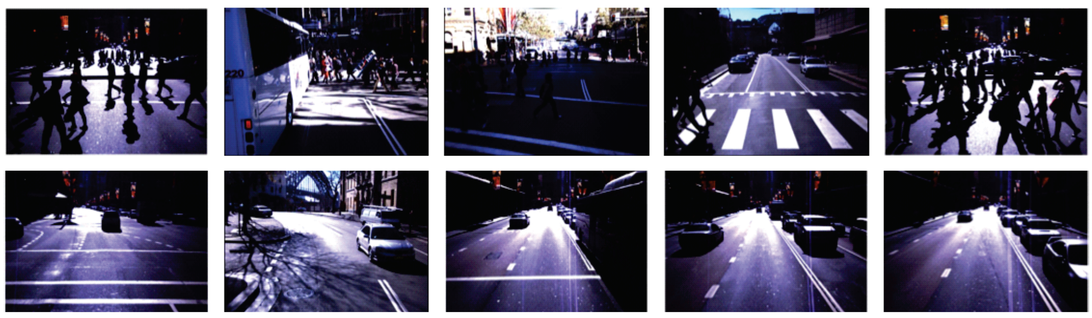
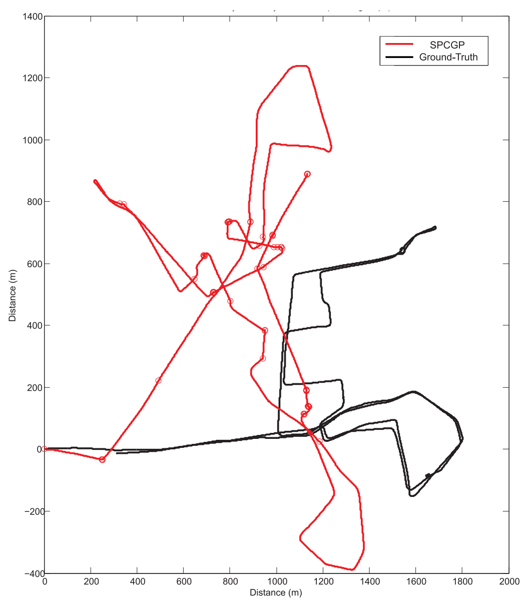
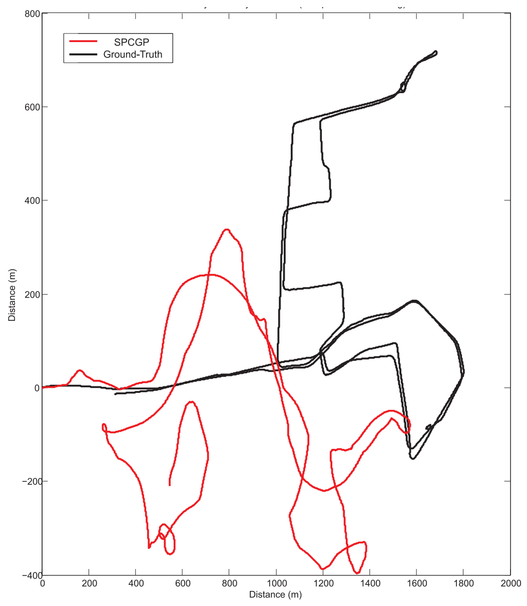


Figure 4.14 – Examples of images collected for the generalization tests with different vehicles.



(a)



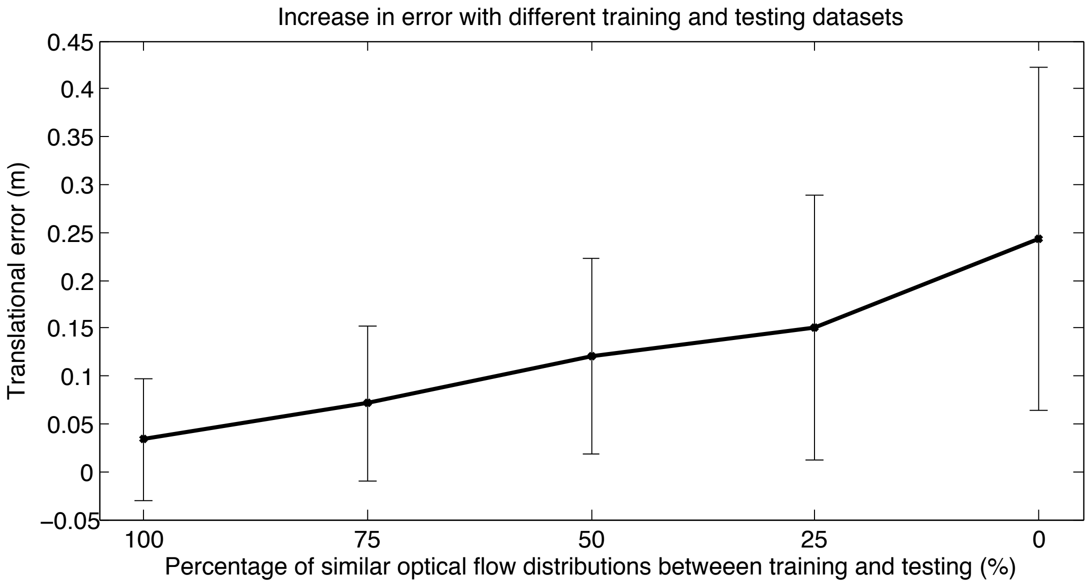
(b)

Figure 4.15 – Localization results in tests with different vehicles. (a) Similar training and testing datasets (the red circles indicate places in which the vehicle is not moving). (b) Vastly dissimilar training and testing datasets.

dataset for training. A percentage of 100% indicates that the training and testing datasets were obtained entirely under the same conditions, and a percentage of 0% indicates that the training and testing datasets were obtained entirely under different conditions (different vehicle, camera configuration and environment structures).

The results of such tests are presented in Fig. 4.16, where it is possible to see that the translational error increases roughly linearly with the percentage of dissimilarity between training and testing datasets. The rotational error also increases monotonically with the percentage of dissimilarity, however the rate in which this error increases has a significant jump at around 50%, indicating the point in which the algorithm starts to fail. When the training and datasets are completely dissimilar, translational error is roughly 5 times higher than when they are obtained under similar conditions, whereas rotational error is roughly 10 times higher. The error bars of such measures also increase monotonically with the dissimilarity percentage, indicating that measurements become more irregular and sensitive to noise.

Another series of tests was designed to evaluate the impact of dissimilarities between training and testing datasets, but now this dissimilarity was generated by skipping frames in the testing dataset, thus creating an artificial change in scale. This frame skipping technique creates optical flow distributions that were not learned during



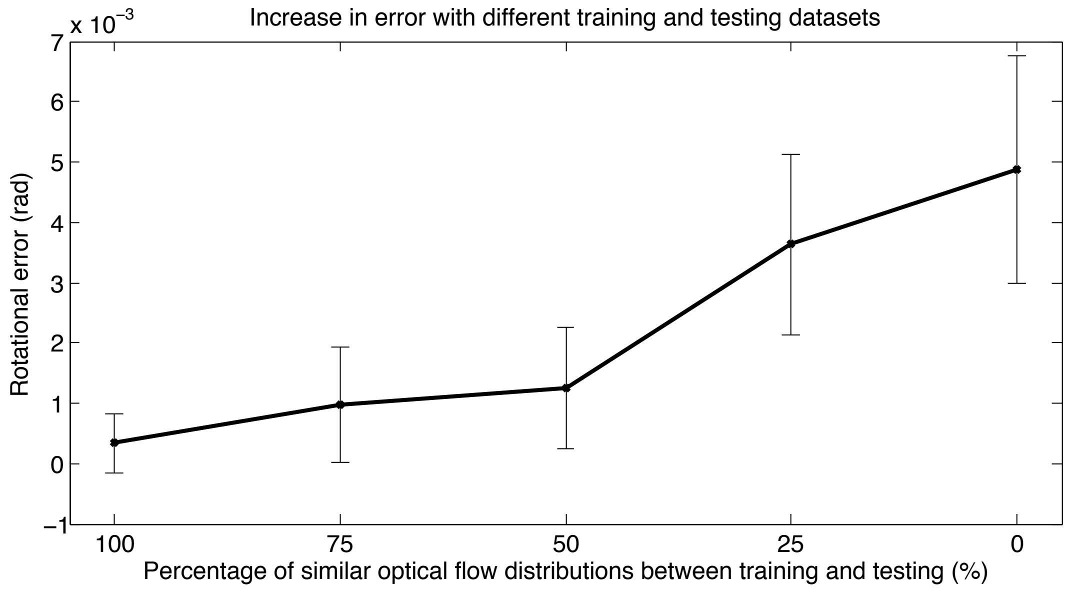
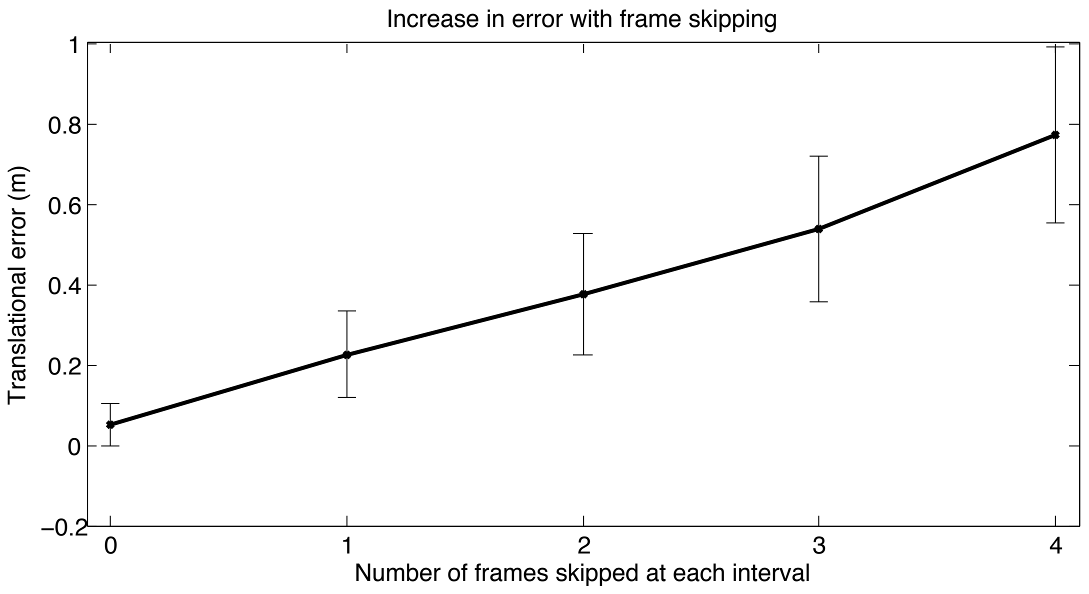


Figure 4.16 – Decrease in performance as the percentage of dissimilarities between training and testing datasets increases.

training, as now the vehicle traverses a longer distance between frames (essentially, the constant frame rate assumption no longer holds true). The results of such tests are presented in Fig. 4.17, where we can see a similar trend in how translational and rotational errors increase with the number of frames skipped. Translational error



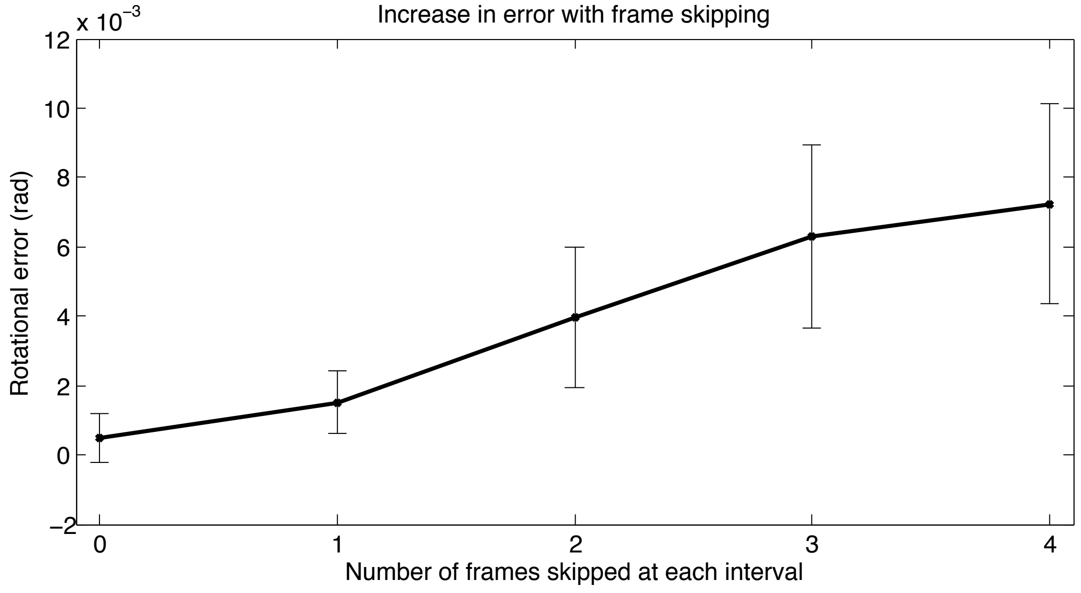


Figure 4.17 – Decrease in performance as number of skipped frames in the training dataset increases.

increases roughly linearly with the number of frames skipped, whereas rotational error initially increases slowly and has a significant jump midway through the tests (we attribute this jump to the increasingly smaller overlapping area between frames). When 4 frames are being skipped (only one for every 5 are used), translational error is roughly 40 times higher than in the initial configuration, and rotational errors are roughly 15 times higher, an indication that the algorithm has failed and no more useful information can be obtained.

4.4 Summary

This chapter improved on the visual odometry algorithm proposed in Chapter 3, addressing its various shortcomings. Five different extensions to the GP framework were proposed: 1) The modelling of cross-dependencies between different outputs (Coupled GPs, Section 4.1.1), that allows the full recovery of the covariance matrix and improves estimates in under-represented portions of the input space; 2) The introduction of temporal dependencies between outputs of subsequent frames (Section 4.1.2), that increases the amount of information available for inference and further constrains the solution space to remove ambiguities; 3) The incremental update of the covariance matrix (Section 4.1.3), that allows the non-parametric model to gradually adapt to new environments and learn useful optical flow distributions as they become available; 4) The incorporation of a geometric model as the mean function for the GP framework (Semi-Parametric CGPs, Section 4.1.4), that provides an initial estimate that is then further refined by the non-parametric model; 5) and an extension to SLAM (Section 4.2), in which all vehicle poses are tracked over time and a loop-closure algorithm is used to detect revisited areas, with this information being used to globally decrease uncertainty.

Experiments are conducted to evaluate the improvements provided by such extensions over the basic visual odometry algorithm, both in 2D and 3D environments. Qualitative and quantitative comparisons between different visual odometry techniques are presented and discussed, and they testify to the proposed algorithm's ability to generate accurate estimates even in challenging situations to traditional approaches. Generalization experiments are conducted to verify the algorithm's sensitivity to dissimilarities between training an testing dataset, a common trait in machine-learning based techniques, that require a well-represented input space in order to generate accurate results. These generalization experiments include changes in environment structure, camera configuration and vehicle dynamics, and serve to delineate the point in which the proposed algorithm is no longer capable of providing useful estimates.

However, during these generalization experiments another issue was detected: the presence of dynamic objects in the environment. Visual odometry algorithms assume a static environment, meaning that any optical flow detected is due to the camera's own translation and rotation between frames. Any dynamic object will, therefore, introduce a component of error in the optical flow distribution, essentially tricking the algorithm into thinking that the camera has moved in a different way (see Fig. 4.15a). Sporadic and localized dynamic objects are removed by the outlier detection algorithm and averaged out by the optical flow parametrization process, however if the scene is highly dynamic these techniques are not enough and these objects must be actively detected and removed before the image is deemed useful for visual odometry purposes. The next chapter is devoted to a novel technique for automatic segmentation and removal of dynamic objects, where each pixel in the image receives a probability of being either static or dynamic, along with a corresponding uncertainty measurement in regards to this classification. Features that are classified as dynamic are removed before the optical flow parametrization process, thus maintaining the assumption of a static environment around the vehicle during navigation.

Chapter 5

Automatic Segmentation of Dynamic Objects

The previous chapter described several techniques capable of addressing the various shortcomings in using a non-parametric machine learning approach to visual odometry. Cross-dependencies between outputs were modelled using a new extension called Coupled GPs, capable of recovering a full covariance matrix that quantifies the correlation between different outputs. Temporal dependencies were introduced as a way to increase the amount of information available for inference, especially on the particular case of narrow field of view. Incremental updates of the covariance matrix provide means for the non-parametric model to gradually adapt to new environments and learn new and useful optical flow distributions. The introduction of a geometric model as the mean function for the GP framework reduces the need for comprehensive training datasets, since an initial estimate is already available that is then further refined by the non-parametric model. Finally, the SLAM extension allows the algorithm to exploit the full covariance matrix recovered by the CGP framework to provide absolute motion estimates, when there is a loop-closure.

However, during the experiments a general issue was raised, that is not related to the GP framework used in this thesis but could compromise the results of any visual odometry algorithm. This issue is the presence of dynamic objects in the environment, that generate optical flow that is not consistent with the camera's own rotation and translation. An example of the impact of such dynamic objects is depicted in Fig. 4.15a, where we can see that most of the overall shapes throughout the trajectory are maintained, however there is a systematic drift component that skews the localization results and also some localized sharp turns that do not exist in reality. The systematic component of drift occurs due to the constant flux of cars in the same portion of the image (the right one, in Sydney), that generates an optical flow contrary to vehicle movement and tricks it into thinking that it is moving faster in that portion of the image than in the other side, and therefore it is slowly turning left. The sharp turns occur mostly when the vehicle is not moving (i.e. in a traffic light), when there are pedestrians and other vehicles crossing its path and generating optical flow in different directions randomly. This random optical flow tricks the algorithm into thinking it is rotating on its axis, and when it starts moving again its orientation is compromised.

The visual odometry algorithm proposed in this thesis has some components that allow the filtering of dynamic objects, both by removing outliers using RANSAC and averaging out dynamic features by dividing the image into equal-sized grids and combining its optical flow values into a single component. However, if a substantial portion of the image is dynamic these two techniques are no longer valid, and erro-

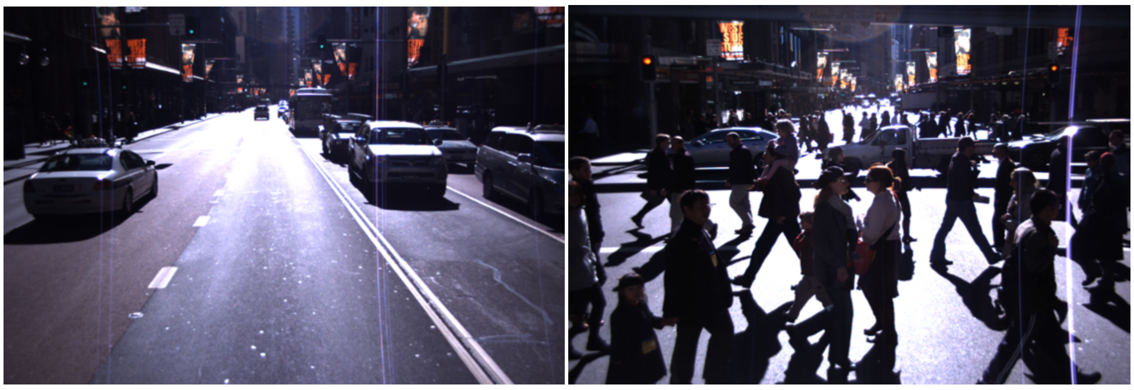

(a) **(b)**

Figure 5.1 – Examples of dynamic objects in visual odometry. (a) Constant flux of cars in one portion of the image, generating a contrary optical flow that creates systematic drift. (b) Pedestrians crossing the vehicle's path while it is not moving, generating random optical flow that compromises orientation.

neous information will be used both for training and testing. A solution would be to actively segment and remove the dynamic objects from the image, and the remaining (static) optical flow information is then used normally for visual odometry. This chapter presents and describes a novel technique for the automatic segmentation of dynamic objects that is based on the same GP principles used throughout this thesis, thus creating an elegant solution that can be incorporated seamlessly into the proposed visual odometry framework[1].

The assumption that no prior knowledge of the environment and/or camera calibration is maintained, and no ground-truth (i.e. manual labelling of images or expensive 3D laser sensors for direct distance measurements) is necessary during the segmentation process. Instead, the ground-truth is composed of the initial classification between inliers/outliers provided by the RANSAC algorithm, and the output is the probability that each pixel in the image belongs to a dynamic object (in a range from 0 to 1, zero meaning static and 1 meaning dynamic). The continuous function generated by the GP framework allows the dense classification of the entire image, even though sparse optical flow information is used as input, and a measure of uncertainty for each pixel classification is also obtained, due to the probabilistic nature of Gaussian processes. New information is incorporated online during navigation, thus allowing the algorithm to learn the characteristics of new dynamic objects and react accordingly to gradual changes in the environment, while redundant information is discarded to maintain computational cost roughly constant.

The remainder of this chapter is dedicated to introducing and describing the proposed algorithm for the automatic segmentation of dynamic objects. It starts by providing a brief overview of the currently available techniques for dynamic object segmentation, and how the proposed algorithm fits within the related literature. Afterwards, its various components are presented and discussed in detail, including the initial RANSAC classification, the novel method for parameterizing the optical flow information that serves as input for the GP framework, and the probabilistic least-squares classification method used to transform a regression technique such as GPs into a

[1]This technique was presented at the International Conference on Robotics and Automation (ICRA) 2013, under the title *Online Self-Supervised Segmentation of Dynamic Objects* [44].

binary classification technique. The tools for incremental update of the covariance matrix, that serve to decide which information is useful and should be incorporated into the non-parametric model and which is redundant and/or erroneous and should be discard, are also presented and discussed, along with a method for obtaining dense pixel-by-pixel classification of the entire image based on sparse optical flow information. Finally, experimental tests are conducted and the results are shown as a way to validate the proposed algorithm both as a tool for dynamic object segmentation and as a way of improving visual odometry results. Initial results on dynamic object clustering are also presented, and the chapter concludes with a summary of its contributions.

5.1 Related Work

Several applications of dynamic object detection assume a static camera, which implies that any non-dynamic object will maintain its position over time. In this scenario it is possible to statistically model the background, essentially "filtering it out", and treat any change in pixel intensity as a potential dynamic object. In [143] each pixel of the image is modelled as a Gaussian distribution, whose parameters are learned from observations in consecutive frames, and in [62] a Kalman Filter is used in a similar fashion. When a uni-modal solution is ill-suited (i.e. when the background changes in a predictable manner, such as trees swaying, fans rotating or water flowing), a mixture of Gaussian models has been applied with satisfactory results [30, 37, 122], and in [121, 124] a Hidden Markov Model (HMM) is used to model the background while exploiting spatial dependencies between pixels. Other approaches forego pixel-wise locality in favour of regional models of intensity, such as eigenvalue decomposition [95] and autoregressive moving averages [84, 145]. A mixture of local and regional models is employed in [132], and in [115] a foreground model is explicitly maintained in order to improve the detection of dynamic objects without using tracking information.

In other applications, however, the visual sensor is mobile, usually mounted on top of a robotic platform. In this scenario it is impossible to separate background and foreground solely by tracking pixel intensity changes, as static objects will also experience relative motion due to camera rotation and translation between frames. A straightforward way of segmenting this sort of image is to model the ground plane and treat everything else as an object [91, 119, 144], however this approach tends to fail in crowded environments where the ground plane is not readily visible. A weaker ground plane constraint is presented in [32], where a coupling between object detection and scene geometry is maintained using a Bayesian network.

If a significant portion of the environment is assumed static, the relative motion of static objects can be filtered out by calculating the optical flow [50] of the image and using a voting method, such as RANSAC [35], to elect the most probable motion hypothesis. Any region that does not comply to this constraint is assumed to be dynamic, and can be tracked using classical approaches such as Extended Kalman Filters [139] or Particle Filters [133] (robust data association algorithms [19] and occlusion-handling techniques [30, 32] are necessary to deal with very cluttered environments). If more than one camera are available, a stereo triangulation can provide a 3D position estimate for matched features [47], incorporating extra information that could be used to facilitate and improve object clustering and tracking [4, 127]. A more accurate detection can also be achieved by applying category-specific models to separate the static background from already established dynamic objects, either on a 3D point-cloud [5, 53], directly on the camera images [23, 64] or in a combination of both [31, 120]. The static background information can also be readily used to improve visual odometry applications [6, 43, 93], since its optical flow values now reflect solely the camera's own rotation and translation.

5.2 Algorithm Overview

A diagram of the proposed algorithm for the segmentation of dynamic object is shown in Fig. 5.2, where we can see several similarities with the simplified version of the proposed visual odometry algorithm as depicted in Fig. 3.1. First of all, the same input data, IMG_1 and IMG_2, is used by both algorithms, which is important since it does not create any need for additional processing. The same stages of feature extraction and matching are also conducted, and the resulting matching set MTC_{12} is filtered according to RANSAC to generate the inlier set INL_{12} that serves as the input information for the visual odometry GP framework. Here, the matching set is used directly as input for the *Initial RANSAC classification*, that provides the ground-truth data R_{12} for training and covariance matrix update. This classification is performed based on the fundamental matrix F_{12}, that encodes the geometrical

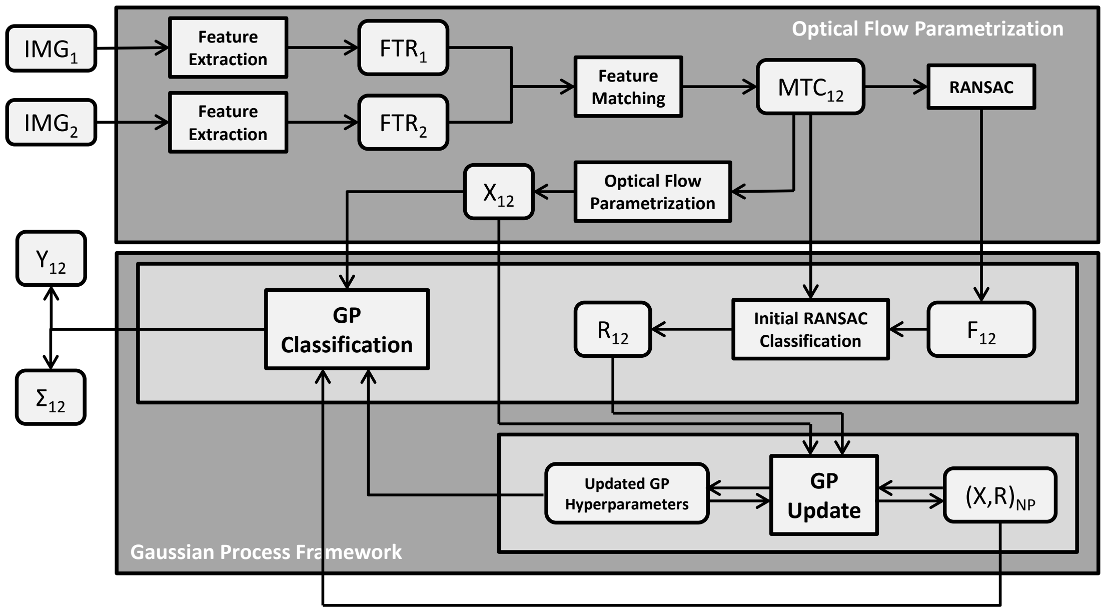

Optical Flow Parametrization
IMG1
IMG2
Feature Extraction
Feature Extraction
FTR1
FTR2
Feature Matching
MTC12
RANSAC
X12
Optical Flow Parametrization
Y12
Σ12
GP Classification
R12
Initial RANSAC Classification
F12
Updated GP Hyperparameters
GP Update
(X,R)NP
Gaussian Process Framework

Figure 5.2 – Diagram of the proposed algorithm for the automatic segmentation of dynamic objects. Again, it is divided into two stages, the *Optical Flow Parametrization* and the *Gaussian Process Framework*. At the beginning of navigation, the non-parametric model $(X, R)_{NP}$ is empty, and random hyperparameters are selected. At each timestep, a matching set MTC_{12} containing optical flow information is extracted from the current frame IMG_2 and the previous one, IMG_1. An initial classification R_{12} is performed using RANSAC, assuming outliers as dynamic objects and inliers as static objects, and each matching pair is parametrized to generate the input vector X_{12} for the GP framework. Both X_{12} and R_{12} are used as input for the *GP Update* stage, where it is decided whether or not this information is relevant, and should therefore be incorporated into the non-parametric model $(X, R)_{NP}$, or if it is redundant and should therefore be discarded. The GP hyperparameters are updated according to this information and used by the *GP Classification* to provide a final classification Y_{12} and variance Σ_{12} estimates for the input vector X_{12}.

constraints of the camera between frames and serves as the geometric model used in the proposed visual odometry algorithm.

Additionally, the matching set also undergoes a parametrization process, which generates the input vector X_{12} for the *GP Classification* and *GP Update* stages. It is important to note that, although the same information is used to generate this input vector, it is not the same as the one used by the proposed visual odometry algorithm since different properties are relevant in each application. Visual odometry requires optical flow information between frames, whereas object segmentation utilizes image coordinates and pixel intensity values. The resulting pair $(X, R)_{12}$ is used as input for the GP update stage, where it is decided whether or not this information is relevant (and should be incorporated into the non-parametric model $(X, R)_{NP}$), or redundant (and should be discarded). This non-parametric model is empty at the beginning of navigation and the hyperparameters are selected randomly, indicating that no prior knowledge of the environment is necessary. If these hyperparameters are available (i.e. from a previous run) they can be incorporated seamlessly.

The GP hyperparameters are updated according to this new information, and the resulting optimized GP hyperparameters are used by the GP framework to generate the final classification Y_{12} for each matching pair X_{12}, along with the corresponding uncertainty estimate Σ_{12}. Since this final classification does not require an initial classification R_{12}, or even a successful match (the input vector is based solely on image coordinates and colour intensities), it can be performed equally in any image pixel, thus allowing a pixel-by-pixel dense classification of the entire image. For speed purposes, this dense classification is performed at every 5th pixel of the image, both horizontally and vertically, and the spaces between pixels are classified using linear interpolation for both the mean Y_{12} and variance Σ_{12} values. The result is a $2 \times h \times w$ structure (where (h, w) are respectively the image's height and width), containing a value ranging from $[0, 1]$ that indicates the probability that each pixel in the image belongs to a dynamic object, along with the corresponding uncertainty in regards to that measurement.

5.2.1 Initial RANSAC Classification

The initial RANSAC classification provides the ground-truth used by the training and covariance matrix update processes, thus eliminating the need for manual labelling or extra sensors that are capable of providing such information directly. As explained in Section 2.3.3, the RANSAC is an iterative algorithm used to estimate the parameters of a mathematical model from a set of observed data which contains outliers. For the application at hand, the mathematical model is the fundamental matrix F, that encodes the geometric constraints correlating the visual system between frames, and the outliers are the dynamic objects in the environment, that generate optical flow that cannot be explained away by the camera's own translation and rotation. The RANSAC algorithm basically elects a random sample from the available data, builds a model and then tests all remaining data points against this model, and the model with the highest number of inliers is determined to be correct. If most of the environment is assumed static, it is natural to expect that RANSAC should elect the model that represents a static environment, and therefore any matching pair that does not comply to these constraints should belong to a dynamic object.

The same techniques for feature extraction and matching used for the proposed visual odometry algorithm are used here, thus eliminating the need for further processing of visual information. Examples of such feature extraction and matching processes are depicted in Fig. 5.3, in the particular cases where the camera is moving (left column) and static (right column). The first row shows the initial sets of features extracted from each image, and the second row shows the corresponding matching sets with the immediately subsequent frames. An average of 7000 features are extracted per frame, and it is immediately possible to see a substantial amount of obviously false matches throughout the image, that do not belong to either dynamic or static objects and should therefore be removed.

This outlier removal is done using RANSAC, however now a distinction has to be made between outliers (false matches) and dynamic features, that were correctly matched but have an optical flow different than that generated by static features. We can see in Fig. 5.3 that, even though a large portion of the environment is dynamic

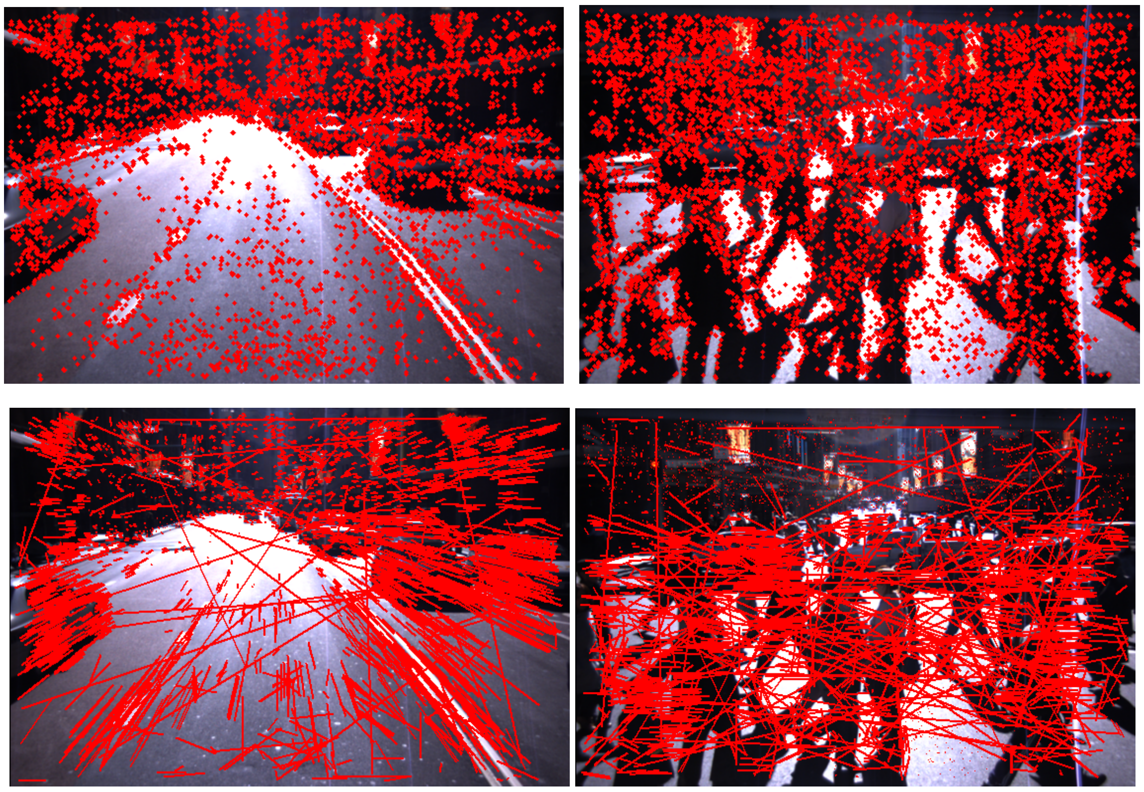


Figure 5.3 – Examples of the feature extraction and matching processes for automatic dynamic object segmentation. The first row shows the features obtained from two different images, and the bottom row shows the resulting matches with their immediately subsequent frames. In the left column the vehicle is moving, whereas in the right column the vehicle is not moving (all optical flow comes from dynamic objects).

(especially in the right column), there are still enough static features to ensure that RANSAC will converge to a model that represents a static environment, mostly in the street and in the upper portions of the image. Once this model is obtained, each matching pair is tested against it and a measurement of error is calculated, based on the Euclidean distance between the match and its corresponding epipolar line. If the match falls in the epipolar line, the projection error is zero and its optical flow is consistent to that of a static object, and the higher the error is the further away the match is from the epipolar line. This error measurement provides a metric for the determination of which features are dynamic and which are static, and matching pairs with a projection error above a certain threshold are discarded as outliers.

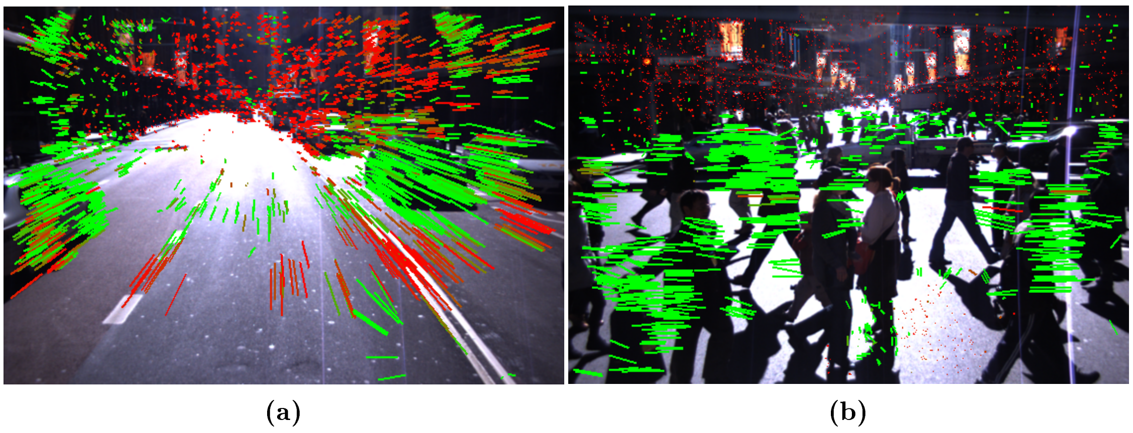


Figure 5.4 – Initial RANSAC classification, based on the image information presented in Fig. 5.3. Each line segment has a colour ranging from red (static object) to green (dynamic object), and matching pairs considered outliers were discarded.

This threshold is also used to normalize the remaining projection errors to values ranging from $[0, 1]$, which indicate the probability that each matching pair belongs to a dynamic object.

Examples of this initial RANSAC classification are shown in Fig. 5.4, where the projection error values ranging from $[0, 1]$ were converted to colours ranging from red (static objects) to green (dynamic objects). In Fig. 5.4a the vehicle is moving forward, generating a relative optical flow component that also influences static features. Because of that, even though most dynamic objects were correctly classified, there are several portions of the image that are wrongly classified as dynamic, especially towards the border where the relative optical flow component of camera motion is stronger. The street also contains several features that were wrongly classified, mostly due to the lack of texture that increases ambiguity during the matching process and increases the chances of false positives. In Fig. 5.4b the vehicle is not moving, and so any optical flow detected is solely due to the presence of dynamic objects. It is clear that this scenario greatly facilitates classification, as now any feature that experiences motion between frames can be safely classified as dynamic. However, we can also see several dynamic objects that are not represented by any features, and thus are not detected by the algorithm. We attribute this lack of representation to occlusion, the presence of deformable objects and local luminosity changes.

This is the visual information that is currently used as input for the visual odometry algorithm described in the previous chapter. However, this information is not ideal and contains a significant amount of dynamic objects that were incorrectly classified as static, and will therefore introduce a component of error to the final optical flow distribution that describes the image. Because of that, a second layer of classification is proposed here, that uses this initial classification as ground-truth and a novel descriptor to generate the corresponding input vector. The goal is to provide a more robust and reliable classification between static and dynamic objects by removing these errors, and also a dense classification capable of addressing the gaps in the image where there are no discernible features.

5.2.2 Optical Flow Parametrization

For a variety of reasons, the descriptor used to parametrize the optical flow information for dynamic object segmentation is different than the one used for visual odometry estimation, even though they are both generated from the same visual information. Dynamic object segmentation is binary, so there is no need to parametrize the magnitude and/or orientation of the optical flow, which now becomes the output given by the initial RANSAC classification. Also, the classification is now individual, rather than collective for the entire image, so each matching pair should have its own descriptor based on the area of the image in which it was detected. For these reasons, the new descriptor used to generate the input vector $\mathbf{x}_n$ for the GP classification is given by:

$$\mathbf{x}_n = \{u, v, m, r, g, b\}. \tag{5.1}$$

In the equation above, u and v are the pixel coordinates of the feature in the image (as a convention, the image coordinates of the feature in the first frame are used). This information is necessary in order to correctly model different motion patterns throughout the image (i.e. upper portions are mostly static, whereas right portions usually present motion contrary to the vehicle), and compose the spatial component of the descriptor. The four other components, $\{m, r, g, b\}$, are the colour components and are calculated by placing a w-by- w window centered on (u, v) and extracting the average of pixel intensities in this area (a 7-by-7 window is used in the experiments). The value of m is obtained by applying this method on a monochromatic version of the image, while $\{r, b, g\}$ are obtained by applying this method respectively on the red, blue and green channels of the coloured version of the image. This information is necessary in order to correctly model transitions between objects, delineating their borders and allowing the algorithm to "fill in the gaps" where no features were detected. The assumption that objects will have features of a similar colour is made, however the division of colour information into four components reduce the impact of such assumption, and since the goal is the removal of dynamic objects the division of any one single object into several sub-objects is not an issue. Other possible parametrizations, such as HSV, could lead to better results by removing redundancies in the colour spectrum, and future work will focus on exploring such variations.

It is important to note that, since this descriptor does not require matching information between features in different frames, it is possible to used it to parametrize any pixel in the image. These pixels will not have a corresponding ground-truth (obtained using RANSAC, that requires matching information), and therefore cannot be used to update the non-parametric model, however they can still be used for the inference process that provides the final classification between static and dynamic objects. By performing this inference on all image pixels, a dense classification of the entire image can be obtained, even though only sparse optical flow information is available for training.

5.2.3 Gaussian Process Classification

While the predictive mean $\bar{f}$ provided by the GP framework is useful in determining the most likely hypothesis, it can also be misleading if considered in isolation, because it does not provide any insight in the accuracy of such estimate. One of the key advantages in using Gaussian processes is the ability to calculate the variance $\mathcal{V}(\bar{f})$ of each prediction, that acts not only as a way of identifying areas with a high uncer-

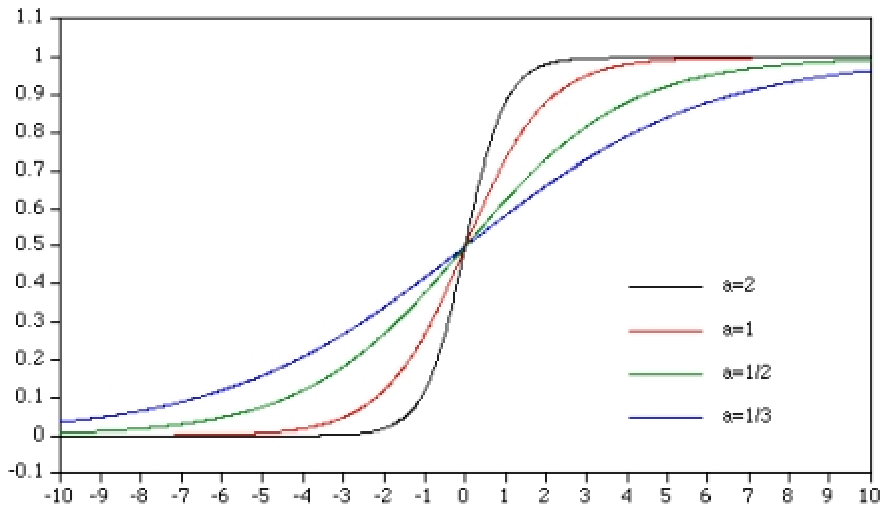


Figure 5.5 – Examples of the sigmoid function used to bound the GP regression results between $[0, 1]$, for different values of α.

tainty measurement but also can be combined with the predictive mean to generate a probability distribution that acts as a classifier for the entire input space. One of the possible methods for performing such classification is the *Probabilistic Least-Squares Classification* (PLSC) [101], which "squashes" the prediction using a bounded function in order to obtain a regression output ranging from $[0, 1]$. Other methods, such as the Laplace Approximation [140] and Expectation Propagation [83], are equally valid, however they require multiple iterations before convergence to a single query unit, which generates an undesirable computational cost that PLSC manages to avoid by directly modifying the regression model to perform classification.

The bounded function used here is the sigmoid function (as introduced in [100]), a special case of the logistic function that is real-valued, differentiable and has a pair of asymptotes at 0 and 1 as $x \to \infty$. Its two parameters, α and β, define the sharpness of the transition between classes (Fig. 5.5) and are determined using the cross-validation technique, as described in Section 2.2.5. The implemented version for parameter training is given by:

$$p(y_i|X, y_{-i}, \theta) = \Phi\left(\frac{y_i(\alpha\mu_i + \beta)}{1 + \alpha^2\sigma_i^2}\right), \tag{5.2}$$

where $\Phi(.)$ is the cumulative unit Gaussian, y_{-i} refers to the output values of all training data excluding the pair $(\mathbf{x}_i, y_i)$, μ_i and σ_i are the predictive mean $\bar{f}$ and variance $\mathcal{V}(\bar{f})$, and θ represents the optimized hyperparameters of the covariance function. The training of α and β can be performed by partitioning the original matrix K^{-1} to eliminate the influence of $\mathbf{x}_i$, thus eliminating the need of recalculating the entire covariance matrix for each training point [135]. The new expressions for the predictive mean and variance are presented in Eq. 5.3, and they allow the classification of each pixel in the image as a static, dynamic or unsure object, according to user-defined thresholds.

$$\mu_i = y_i - \frac{[K^{-1}y]_i}{[K^{-1}]_{ii}} \qquad \sigma_i^2 = \frac{1}{[K^{-1}]_{ii}} \tag{5.3}$$

5.3 Incremental Updates

The proposed algorithm for the automatic segmentation of dynamic objects does not make any assumptions about the environment and/or visual system utilized, which means that it should be able to provide robust estimates in any scenario where visual information is a valid way of obtaining information. There are two possible ways of doing so in the GP framework: either by maintaining a comprehensive training dataset that contains every possible sample of dynamic objects the vehicle might encounter during navigation, or by allowing it to learn and adapt to new circumstances as they are presented. Even so, the first solution fails due to the unpredictable nature of dynamic objects (they might start/stop moving suddenly) and a myriad of random phenomena that a single training dataset will never be able to truly address.

Hence, the only solution becomes to allow the algorithm to adapt to new circumstances as they are presented, gradually learning the behaviour of objects around the vehicle and constantly changing its beliefs to reflect the current environment state. With this approach, a dynamic object that stops moving would eventually become static and vice-versa, maximizing the information available for visual odometry without generating errors, and new objects that enter the camera's field of view would be incorporated seamlessly into the non-parametric model. The recalculation of the entire covariance matrix is of computational complexity $\mathcal{O}(n^3)$ and therefore infeasible for an online approach, so the solution proposed here involves the use of the covariance matrix update equations presented in Section 4.1.3, that allow the iterative incorporation of new information into the covariance matrix and the removal of redundant information.

The issue now becomes determining which information is relevant and which is redundant, thus incorporating enough data to provide an accurate classification while maintaining computational complexity at a reasonable level. The next section is dedicated to introducing and describing a novel method for information filtering that is able to maintain the number of data points in the non-parametric model stable even after long periods of navigation, thus keeping computational cost roughly constant. Afterwards, a technique for obtaining a dense classification of the entire image without performing inference in every pixel (again, a naive and very computationally costly approach) is discussed.

5.3.1 Information Filtering

The information filtering technique described here is composed of four individual steps, that are performed at every timestep when new information becomes available. The first step is to enforce a density constraint on the new data points, as a way to eliminate redundant information. Inference is then performed on the remaining data points in an attempt to discard those that lie on an already well-represented portion of the input space. The same inference process is then performed on the non-parametric model itself, which serves as an outlier removal and maintains consistency even after long periods of navigation. Lastly, the density constraint is now enforced on the non-parametric model itself, again to eliminate redundant information. Each of these four steps are described below in more detail:

Density Constraint on Input Data

The combination of SIFT and corner detector algorithms used in this thesis to extract image information produces an average of 7000 features per frame, which is a very large number for this particular application and prohibits the direct incorporation of all new data points into the non-parametric model. Fortunately, most of this information is redundant, since features tend to be clustered into specific portions of the image and may share a similar classification. This allows them to be safely discarded without compromising results by performing a nearest-neighbour search in the input space for each feature and discarding those whose classification is similar to their closest neighbours, according to a certain distance threshold. This process is repeated until only one feature of any given class remains in each portion of the input space determined by the distance threshold, and if any portion is represented by both classes then two features are maintained, one for each class.

Inference on Remaining Input Data

The next step consists of performing inference on the remaining input data points, that were not discarded in the previous step. This inference process provides a final GP classification for these data points based on the current non-parametric model (this step is skipped on the first iteration), which is then compared to the initial RANSAC classification. Data points that are *correctly* classified (the GP classification is the same as the RANSAC classification) are discarded, because their position in the input space is already well-represented and does not require more information to provide accurate estimates. Those that were *incorrectly* classified are assumed to be relevant and are incorporated into the non-parametric model, increasing the amount of information available for inference.

Inference on Non-Parametric Model

The next step is to perform inference on the non-parametric model itself. This time, data points that are *incorrectly* classified are removed, thus decreasing the amount of information available for inference. This step is important in eliminating RANSAC misclassification (outliers), as they are assumed to be a minority and therefore less representative of their position in the input space. The removal of such misclassifications allows the non-parametric model to maintain consistency even after long periods of navigation, essentially forgetting old environment behaviours and adapting to new ones.

Density Constraint on Non-Parametric Model

Lastly, the density constraint is enforced on the non-parametric model itself, by removing data points with similar classification that are close to each other according to a certain distance threshold. As in the first step, this serves as a way to decrease the amount of redundant information available for the non-parametric model while still maintaining its spatial distribution in the input space.

Fig. 5.7 shows the progression of the size of the non-parametric model during navigation. It starts empty, without any information, and at the first iterations a large number of data points are incorporated, because the algorithm is still learning the different static and dynamic structures of the environment. When the number of incorporated data points reaches roughly 4000 the size of the non-parametric model stabilizes, with roughly the same number of data points being incorporated and removed at each iteration. This is to be expected, since as a general rule the environment changes gradually with each frame, and the algorithm is capable of learning new behaviours at the same rate in which it is forgetting old ones. When there is a sudden change in the environment (i.e. the camera started/stopped moving, or a previously dynamic object became static or vice-versa) there is a spike on the number of data points incorporated, indicating that the environment suddenly became more complex and the algorithm is trying to learn this new configuration. Once it has managed to do so, the size of the non-parametric model stabilizes and returns to the 4000 level, which is maintained even after an undetermined long period of navigation. This level can be adjusted by fine-tuning the value for the distance threshold in the density constraint, according to requirements in performance and computational efficiency.

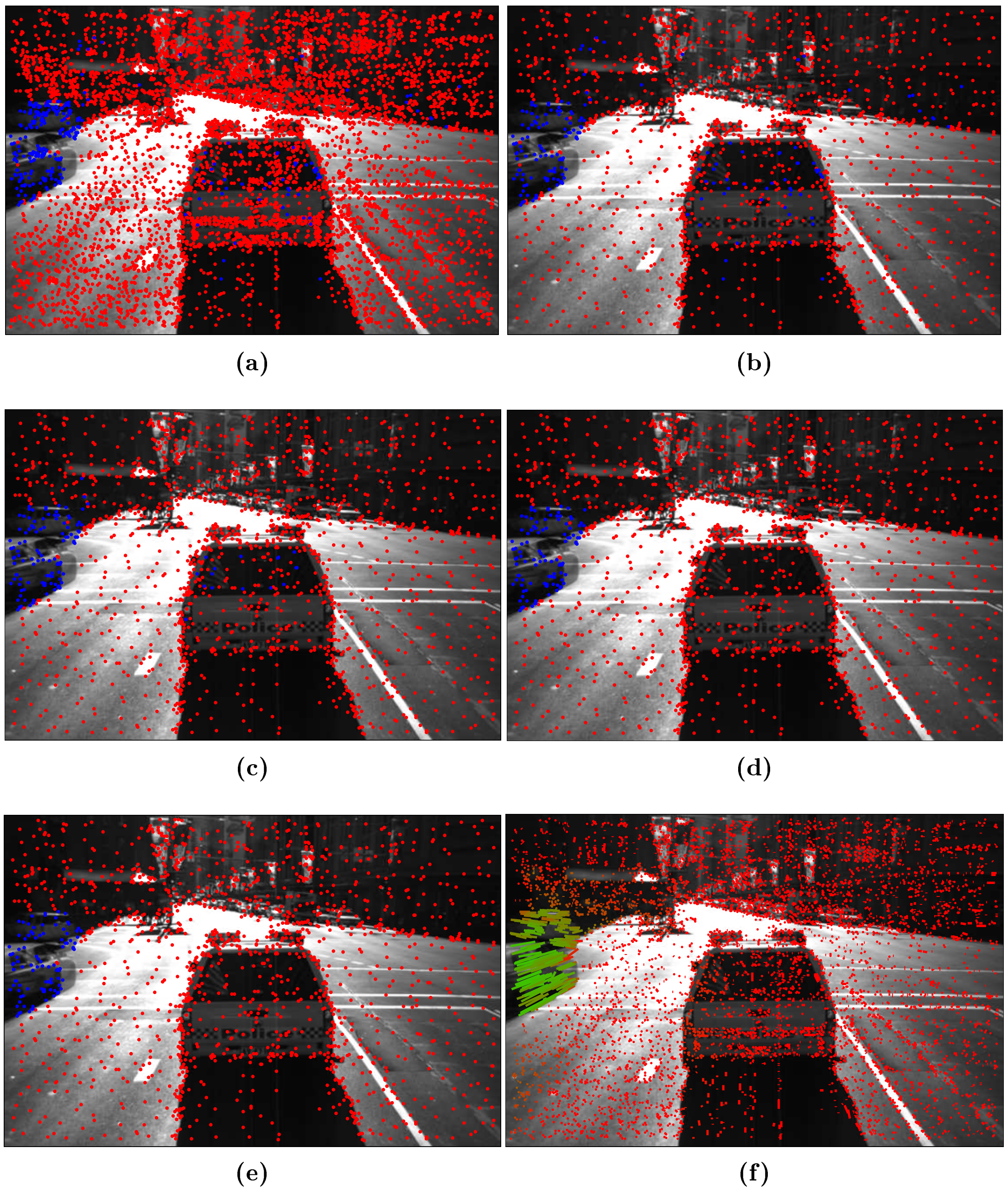


Figure 5.6 – Stages of information filtering (red dots are features classified as static, and blue dots are features classified as dynamic). (a) Initial input data points. (b) Input data points after density constraint. (c) Data points after inference that are incorporated into the non-parametric model. (d) Non-parametric model data points after inference. (e) Non-parametric model after density constraint. (f) Final GP classification of all input points, ranging from red (static) to green (dynamic).

5.3.2 Pixel by Pixel Classification

In principle, the entire image can be classified using the Probabilist Least-Squares Classification technique described in Section 5.2.3, by calculating the input vector for each pixel according to the descriptor introduced in Section 5.2.2 and performing GP inference based on the current non-parametric model. However, inference has a computational complexity of $\mathcal{O}(n^2)$, with n being the number of data points in the non-parametric model, and for a 600×400 image a total of 240000 inferences would have to be performed. Needless to say, these numbers make a dense classification of the entire image infeasible for an online algorithm.

Since each inference is performed independently, sharing the same covariance matrix K and ground-truth vector $\mathbf{y}$, one straightforward solution would be to parallelize the inference process, or to perform the matrix multiplications in a GPU (Graphic

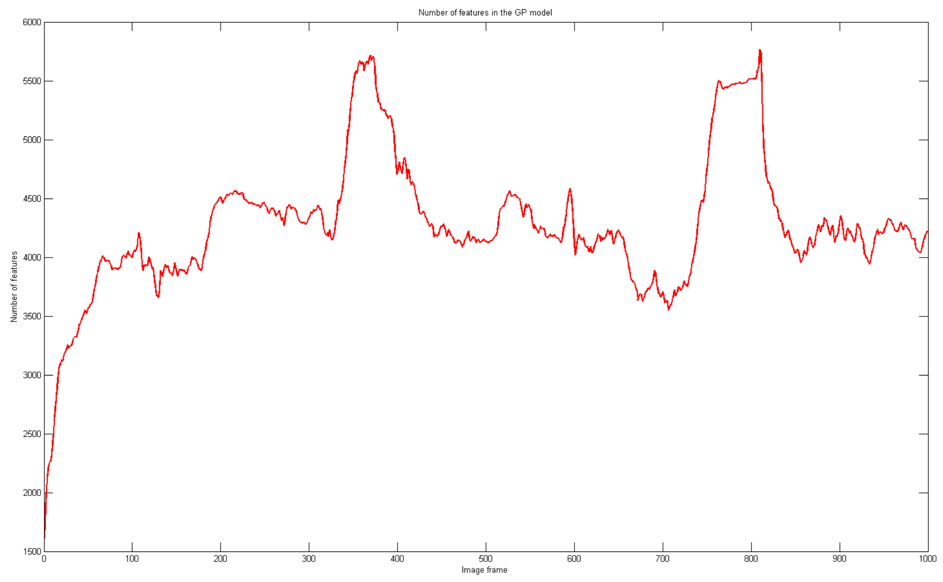
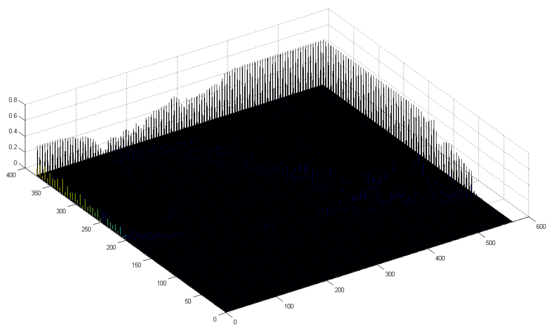


Figure 5.7 – Number of data points maintained by the non-parametric model at each iteration. Note that this number stabilizes at around 4000, with spikes indicating moments in which there was a radical change in the environment (i.e. the camera started/stopped moving, or a static object suddenly became dynamic or vice-versa). As the algorithm learns these new behaviours, the number of data points in the non-parametric model stabilizes and returns to the 4000 level.


(a)

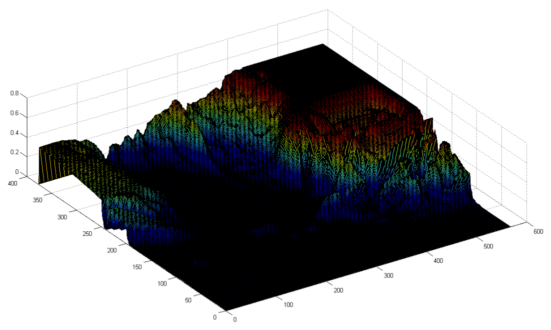

(b)

Figure 5.8 – Example of pixel subsampling. In (a), only every 5th pixel in each row and column is classified, providing an outline for the final classification that is then completed in (b) using linear interpolation.

Process Unit). Another solution proposed here is *pixel subsampling*, in which only certain pixels are selected for inference and the remaining are calculated based on their neighbours' properties. For this particular application, a predetermined number of rows and columns are skipped during inference (i.e. every other row and column are skipped), and their classification is calculated using linear interpolation based on their nearest neighbours both horizontally and vertically. This approach decreases the computational complexity by a factor of n^2, where n is the number of rows and columns skipped, and by fine-tuning the value of n it is possible to achieve the desired computational complexity. It also has the added benefit of blurring the borders of dynamic objects, thus providing a "safety zone" that minimizes the chances of using false information for visual odometry and filling in the gaps in small patches where no texture is available.

An example of such interpolation is shown in Fig. 5.8, in which each pixel in the image is depicted by its probability of belonging to a dynamic object. In Fig. 5.8a, only every 5th pixel in each row and column is classified, providing an outline of the final classification that is then completed via interpolation as shown in Fig. 5.8b. The resulting probabilistic classification can then be transformed into a discrete classification of static/dynamic/unsure objects, based on user-defined thresholds and the uncertainty estimates for each pixel. Matching pairs classified as dynamic or unsure are simply discarded and no longer used to generate the input vector $\mathbf{x}_n$ for the

SPCGP framework. It is important to note that the entire image does not need to be classified for visual odometry purposes, since only matched features provide the optical flow information necessary to estimate vehicle motion between frames.

5.4 Experimental Results

The proposed algorithm for the automatic segmentation of dynamic objects was tested using the same dataset used for the generalization experiments with different vehicles, as described in Section 4.3.3. This dataset is highly dynamic, with the vehicle interacting normally with cars, buses and pedestrians at speeds ranging from zero (traffic lights and traffic jams) to roughly 50 km/h. Initially, dense segmentation results are presented as a way to validate the algorithm, where the entire image is classified according to the pixel-by-pixel classification technique described in Section 5.3.2, using subsampling and linear interpolation for speed purposes. The proposed algorithm's performance is compared with traditional approaches for the segmentation of dynamic objects, and tests are conducted with a different dataset to show its ability to generalize to different camera configurations and vehicle dynamics.

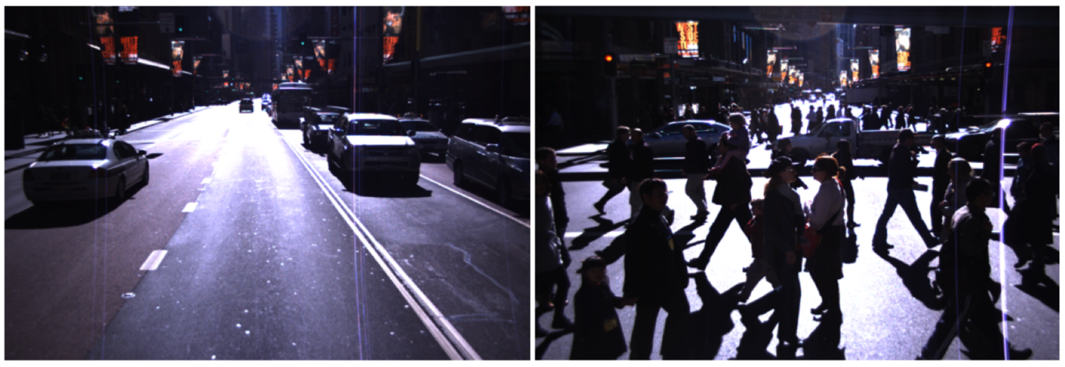
Afterwards, experimental results for the clustering of dynamic objects are presented and discussed. The algorithm for the automatic segmentation of dynamic objects as described in this chapter does not make any assumptions about the nature of the dynamic object, and also does not make any distinctions about different dynamic objects. This is not necessary for visual odometry purposes, where all dynamic objects are treated similarly and discarded, however by exploiting similarities in optical flow patterns and colour information it is possible to group different sets of dynamic features into separate objects, each with specific characteristics. These characteristics can then be used to perform a higher-level form of classification, in which dynamic objects are clustered into different semantic categories, such as cars, pedestrians, etc. This technique allows the creation of a library of dynamic objects during navigation, with no prior assumptions in regards to environment structures and/or camera configuration, that could serve a wide variety of purposes.

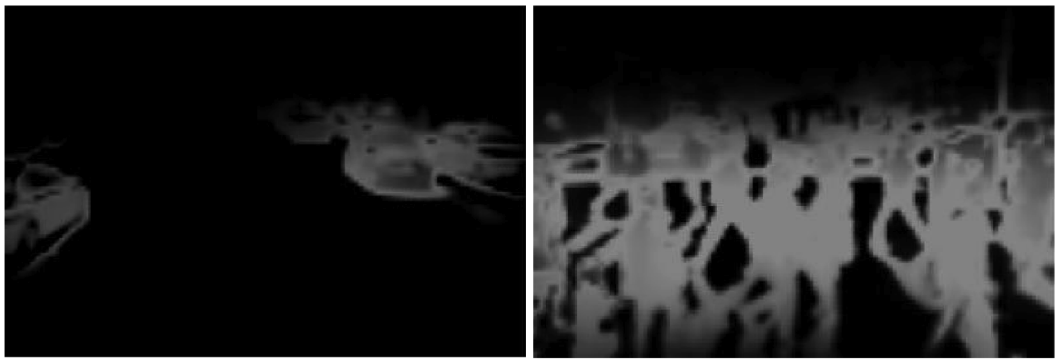
Afterwards, experiments conducted previously with the proposed visual odometry algorithm are repeated with the incorporation of the algorithm for the segmentation of dynamic objects described in this chapter, as a way to show the improvements in visual odometry estimation provided by this technique. For this particular application, no dense classification is necessary, since only the features matched between frames are capable of providing optical flow information and therefore contribute to the input vector used by the SPCGP framework. Results with different extensions of the proposed visual odometry algorithm are presented, including the extension to SLAM which generates absolute localization estimates as a way to remove drift accumulated during navigation. These results complete the experimental portion of this thesis, and testify to the proposed visual odometry algorithm's ability to provide accurate motion estimates using a GP framework based solely on visual information, with no prior knowledge of the environment, camera configuration/calibration or vehicle dynamics.

5.4.1 Dynamic Object Segmentation

Examples of the initial RANSAC classification used as ground-truth for the proposed algorithm were introduced in Fig. 5.4, and here the results of the final GP classification based on such information are presented (Fig. 5.9). A dense classification was performed, with inference being conducted using every 5th pixel in each row and column to provide an outline of the entire image, and the remaining pixels were classified using linear interpolation for speed purposes. This process was done for both the predictive mean (Fig. 5.9b) and variance (Fig. 5.9c) values, which indicate respectively the best hypothesis for each probability distribution and the confidence in regards to such hypothesis.

Right from the start, it is possible to see in Fig. 5.9b that the algorithm was able to correctly detect most of the dynamic objects in the environment, segmenting them from the static background according to their probability of being truly dynamic. Virtually all misclassifications given by the initial RANSAC classification in Fig. 5.4a


(a)


(b)

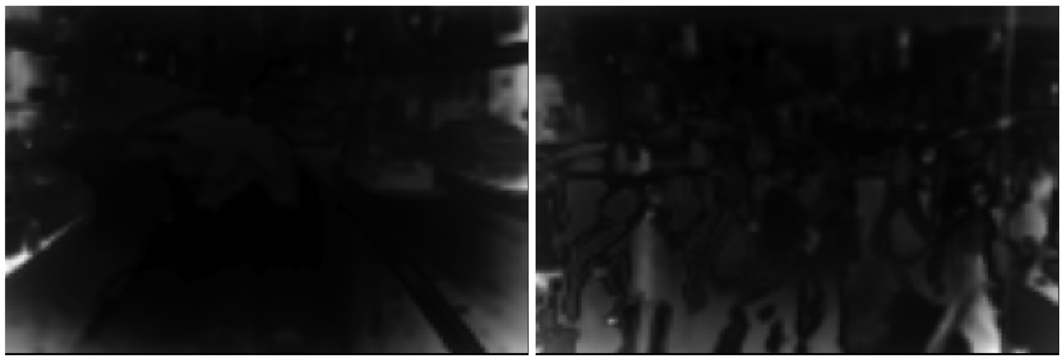

(c)

Figure 5.9 – Examples of the final GP classification results, based on the initial RANSAC information depicted in Fig. 5.4. (a) Original images. (b) Predictive mean, defined by a number ranging from 0 (black, static object) to 1 (white, dynamic object). (c) Predictive variance, normalized to values ranging from 0 (lowest) to 1 (highest).

were removed, especially on the top left and right corners of the image, and also in the areas where the street was represented. We attribute this to the filtering process, which is capable of removing outliers and detecting the correct tendency of each portion of the image even in the presence of significant noise. Also, virtually all the featureless regions in Fig. 5.4b were correctly filled by the dense classification process provided by the GP framework, allowing the complete delineation of all dynamic objects and their boundaries in relation to the static background. Because they are also moving, shadows were classified as dynamic objects as well, which is not a problem for visual odometry applications since their removal will not introduce any error to the final optical flow procedure.

Another key benefit of using the GP framework for segmentation is its ability to calculate the uncertainty inherent to each estimate, thus providing a measurement of variance for each pixel alongside the predictive mean. In the context of object detection, this variance can be used to determine which portions of the image are most likely to be correctly classified and which require more information before a final classification can be made, forming the basis for active learning [27, 114]. Examples of such variance are shown in Fig. 5.9c, with darker areas representing lower uncertainty values and lighter areas representing higher uncertainty values. It is clear that most of the variance is concentrated in the borders of the image, which is to be expected since this region is where the feature density is lowest (lots of features disappear and appear between frames due to vehicle motion) and also where new objects are detected for the first time. As we move to the central portions of the image new objects and features are gradually incorporated into the non-parametric model and the variance decreases.

More examples of the predictive mean obtained using the proposed algorithm are shown in Fig. 5.10. These results were obtained in the same run, meaning that the non-parametric model, starting empty and with random hyperparameters, had to constantly adapt to changes in the environment in order to learn the characteristics of each individual frame. The information filtering process described in Section 5.3.1 was used to keep computational complexity manageable during navigation, and the number of data points maintained at each iteration for the first 1000 frames is depicted in Fig. 5.7. From these images note that even though the vehicle experienced radical

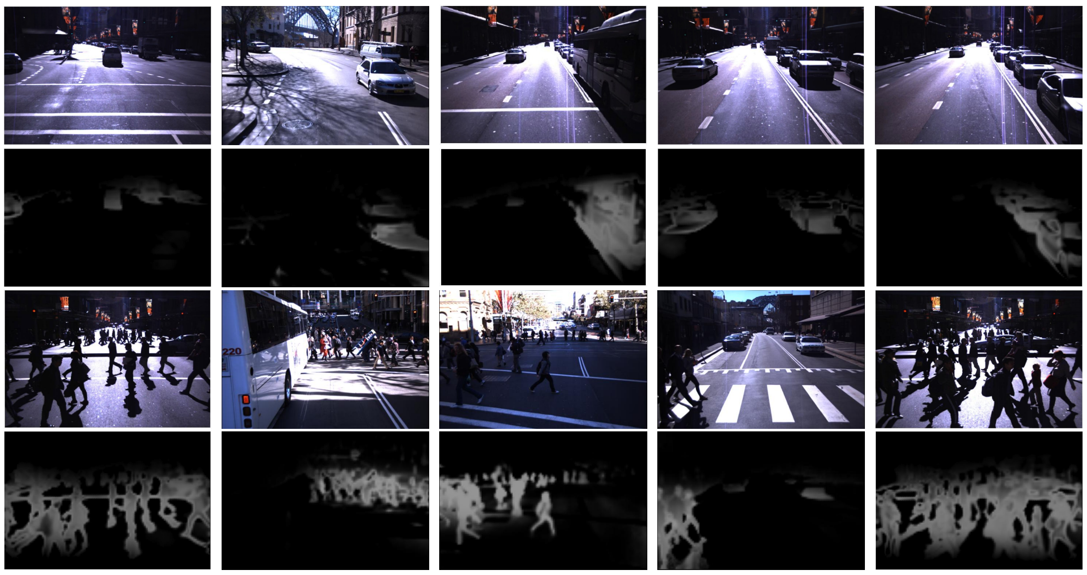
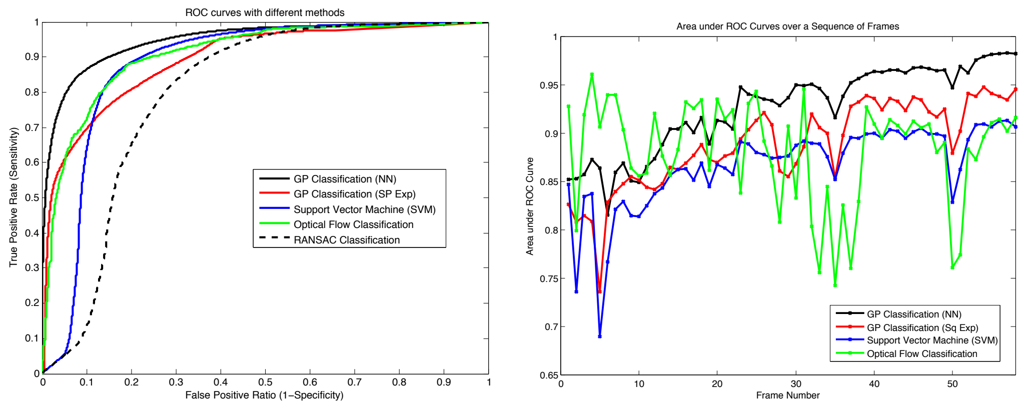


Figure 5.10 – Automatic dynamic object segmentation results. Each pixel is defined by a number ranging from 0 (black, static object) to 1 (white, dynamic object). These results were obtained in the same run as the non-parametric model constantly adapts to new environment characteristics, without any prior information and without any human intervention.

changes in both local and global luminosity, environment structures and a wide range of different dynamic objects, it was still capable of providing accurate segmentation results.

The proposed algorithm for the segmentation of dynamic objects was compared with other approaches to dynamic object segmentation, and the results are presented in Fig. 5.11. These results were obtained based on information from 200 hand-labelled images randomly selected from the 14500 images available for testing. The dotted line represents the initial RANSAC classification results, the black line indicates the proposed algorithm, and the red line indicates the proposed algorithm but with a square-exponential covariance function, instead of the neural network covariance function. The blue line indicates results obtained using *Support Vector Machine* (SVM) [21] as the self-supervised classification method instead of GPs, and the green line indicates the *Optical Flow Classification* (OFC) results obtained based on [88]. The OFC uses motion potentials based on geometry to build a graph-like structure from dense optical flow and feature tracking (the SLAM component was not implemented



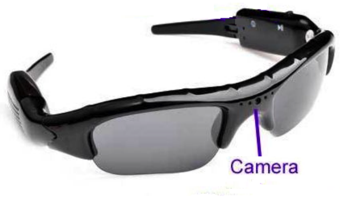
(a) **(b)**

Figure 5.11 – Comparison of the proposed algorithm for the segmentation of dynamic objects with other similar approaches. (a) ROC (Receiver Operating Characteristics) curves for each approach. (b) Area under the ROC curves for each approach in different frames.

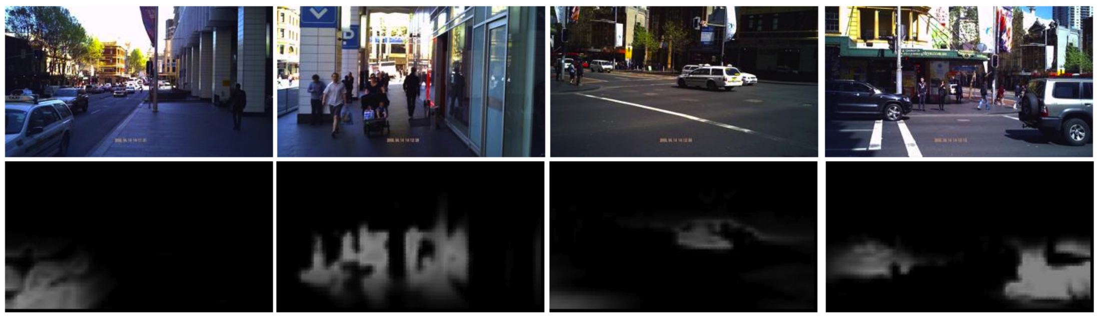
here, as it can be equally applied to any methodology). This graph is then clustered together and nodes with similar potentials become motion segments that generate a single structure. The ROC curves for each of these approaches are shown in Fig. 5.11a, where it is possible to see that the proposed algorithm outperforms the others in all threshold levels, and in particular that it improves over the initial RANSAC classification by a significant margin. It is also possible to see the importance of covariance function selection, since the same algorithm performed significantly worse when the squared exponential covariance function was used.

Fig. 5.11b depicts the area under the ROC curve for each subsequent frame at the beginning of navigation, indicating how accuracy changes as new data is incorporated into the non-parametric model. As expected, the OFC approach does not improve over time, since it is not based on learning techniques, using instead individual information from each frame. The accuracies of the three other approaches increase steadily over time, with occasional drops that indicate moments in which there was a significant change in the environment (i.e. the camera started/stopped moving, or a new object entered the camera's field of vision), and the proposed algorithm consistently outperforms the other two. It is interesting to note that, at the beginning of navigation, the OFC is the best solution, since there was no time for the non-parametric model to learn the environment characteristics. However, after a few frames the proposed algorithm improves and becomes the best solution, while the OFC oscillates heavily at each iteration.

The same algorithm was also tested, without any further modifications, using images obtained from a portable camera device (Fig. 5.12a), as a way to qualitatively explore its ability to generalize over different camera configurations and environments. Again, the non-parametric model was initialized empty from random hyperparameters, and the shakiness of the camera posed a challenge to the framework, since now the baseline between frames is much smaller and its motion is unconstrained by traditional two-dimensional vehicle dynamics. The results obtained using this configuration are shown in Fig. 5.12b, where it is possible to see that again there is a wide variation in luminosity and structures, and the proposed algorithm was still capable of correctly segmenting most of the dynamic objects in each frame (results were similar to the ones depicted in Fig. 5.11). It is also worth mentioning that these results were obtained without any human intervention, based solely on raw visual information collected from a single uncalibrated camera.



(a)



(b)

Figure 5.12 – Dynamic object segmentation results in different frames using a portable camera device.

5.4.2 Dynamic Object Clustering

The dynamic objects obtained during the segmentation process were subjected to a second processing stage, as a way to further segmenting them into different categories (i.e. pedestrians, cars). The segmentation algorithm, as described in this chapter, does not make any distinctions in regards to different dynamic objects, since its main purpose is to remove them and allow the use of a static background during visual odometry estimation. However, by exploiting discrepancies in optical flow patterns and colour information from different features it is possible to determine the boundaries between objects. It is natural to assume that an object would have features that share a similar optical flow distribution, and that this distribution changes gradually during navigation, which allows the tracking of different objects over time, as a way to increase robustness and decrease ambiguity in object segmentation. The dense classification of the entire image also allows the segmentation of dynamic objects according to their spatial coordinates, since large gaps between pixels classified as dynamic indicate the beginning of a new object.

An example of such segmentation is presented in Fig. 5.13, where a rectangle was placed over the original images delineating the boundaries of each individual dynamic object. This segmentation is conducted according to the following iterative process:

1. A random feature is selected on the image, forming the core of a new dynamic object cluster.
2. All its neighbours within a certain radius are checked. Features with an optical flow pattern whose magnitude/orientation are similar within a certain threshold are added as part of the same dynamic object cluster.
3. Step (2) is repeated for all newly added features, increasing the size of the current dynamic object cluster.
4. When there are no newly added features, the process stops and the current dynamic object cluster is determined. Step (1) is repeated for a new random feature that still does not belong to any cluster.

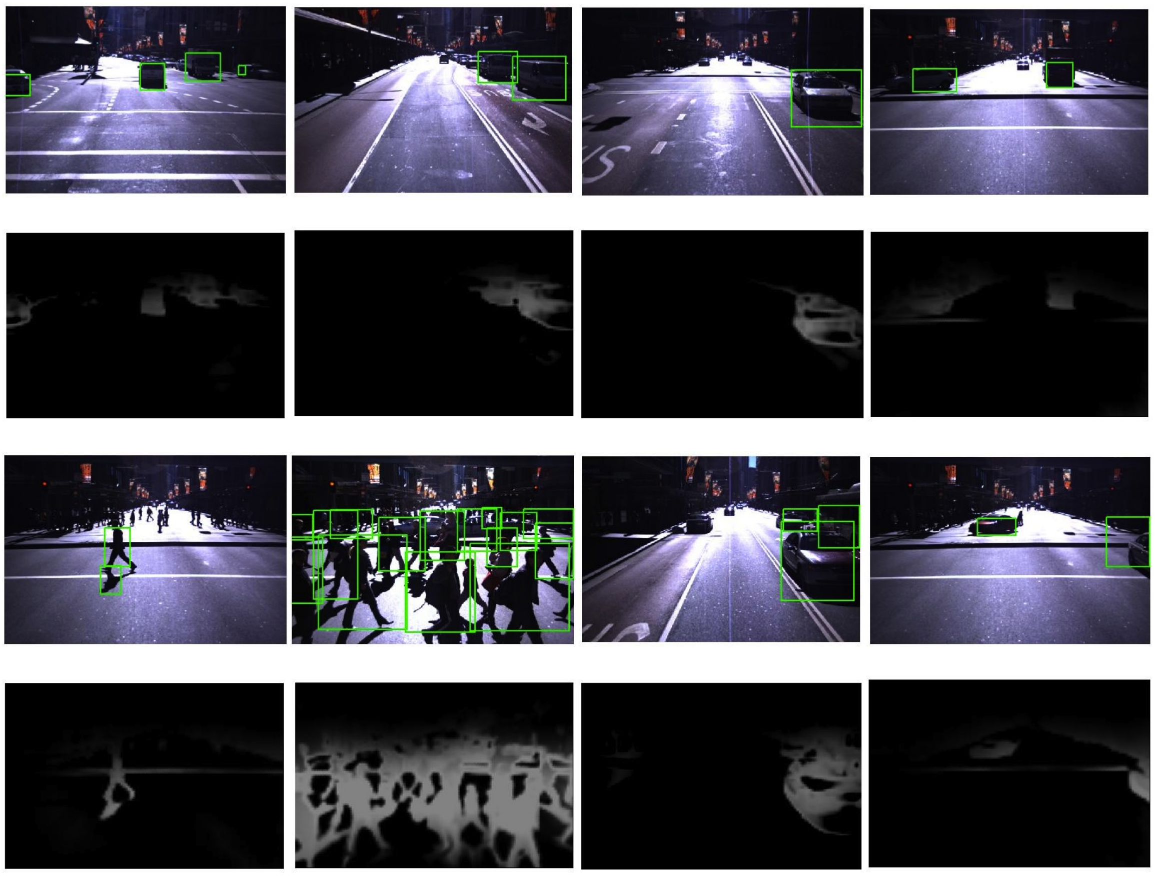


Figure 5.13 – Different objects obtained during the segmentation process.

5. When all features already belong to a cluster, a merging process is conducted to join clusters whose magnitude/orientation average are similar within a certain threshold, and their features share an overlapping area in the image. This step is important to reduce the number of sub-clusters of a single object, due to small variations in the optical flow distribution.

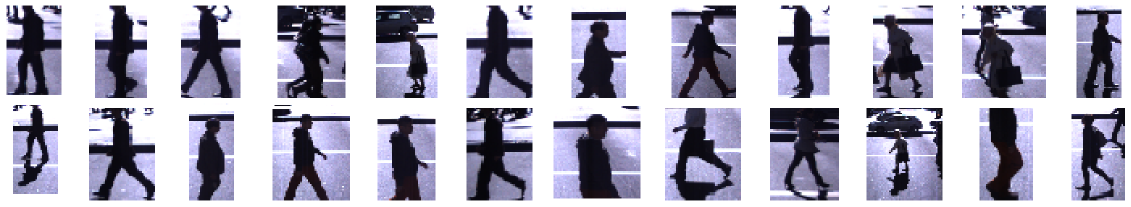
6. Once the merging process is done, filtering is conducted to remove clusters with a number of features that falls below a certain threshold. This step is important as it removes small clusters generated by noise in the segmentation algorithm.

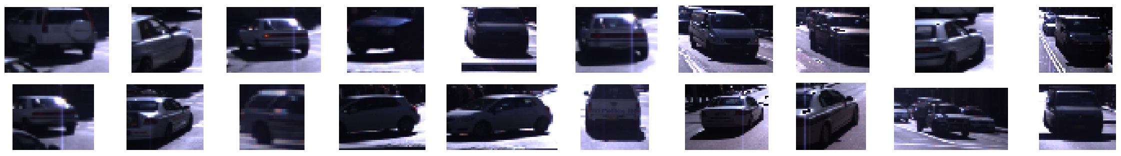
7. Each dynamic object cluster is expanded to include pixels that do not contain features with optical flow distributions, according to the dense classification created by the GP framework. Each feature in each cluster is expanded to

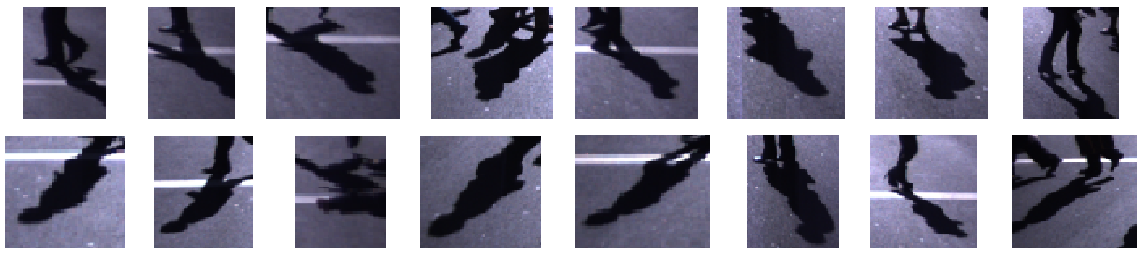
include its neighbouring dynamic pixels, in a process that is repeated until there are no more pixels to expand to. If the same pixel is neighbour to two different dynamic object clusters, a linear combination between colour information and distance to the nearest matched feature is used to determine which cluster it should be incorporated to.

Once all dynamic objects are obtained, the next step is to further divide them into different categories, according to the object each one represents (in an urban environment, mostly cars and pedestrians). This is done using data collected from 1000 consecutive images, all segmented according to the algorithm described previously, for a total of 2077 objects. Each object is described using a histogram for each one of its colour components (red, green and blue), divided into 6 bins. These histograms are normalized to have an unitary sum, to account for objects of different size. These descriptors are then categorized using k-means [54], a clustering method which aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean. For the experiments presented here, we selected 8 as the number of clusters to be formed, as a way to minimize the impact of random objects that do not fall within any other category.

Fig. 5.14 shows the results for the three clusters with the most samples, obtained using the method described above, along with some examples belonging to the other five clusters. It is important to note that these categories do not imply any knowledge of the environment, and were generated based solely on the dynamic objects collected during navigation. Nevertheless, we can see a clear pattern present in each one of them, indicating that k-means was able to correctly cluster these dynamic objects into semantically meaningful categories. Fig. 5.14a contains mostly pedestrians, Fig. 5.14b contains mostly cars and other vehicles, and Fig. 5.14c contains mostly shadows from pedestrians. Other clusters include mostly partial objects, that were not merged into a single group for some reason, shadows from static objects, that were classified as dynamic due to the learning nature of the self-supervised algorithm, and other sporadic misclassifications. A ground-truth dataset was generated by manually labelling 200 dynamic objects between these three main categories, and the proposed algorithm shows an accuracy of around 70% in object classification (a confusion matrix for these results is presented in Table 5.1). Tests were conducted using different


(a)


(b)


(c)

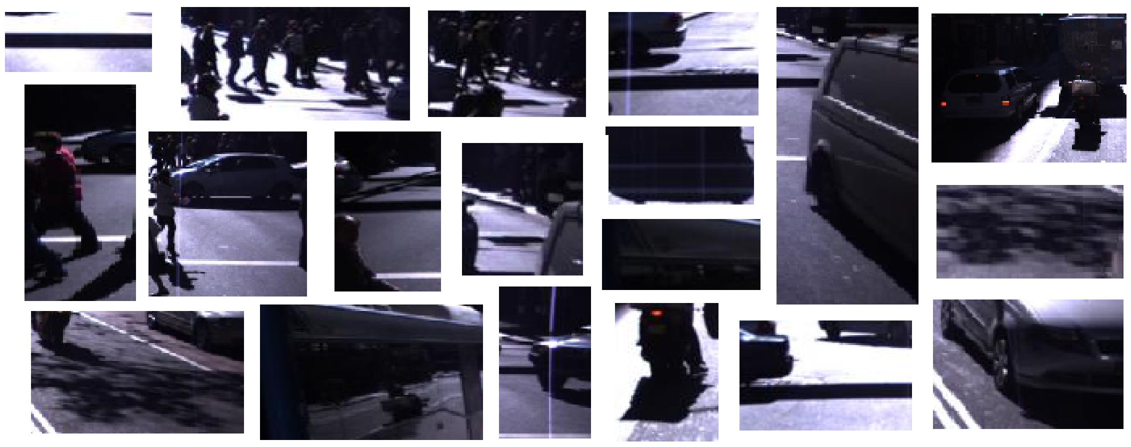

(d)

Figure 5.14 – Clustering results for different dynamic objects. (a) Pedestrians. (b) Vehicles. (c) Pedestrian shadows. (d) Example of Objects that do not belong to any of the three main categories.

numbers of clusters, and while a higher number did not show any significant improvement, smaller numbers show a merging between different clusters that affected results.

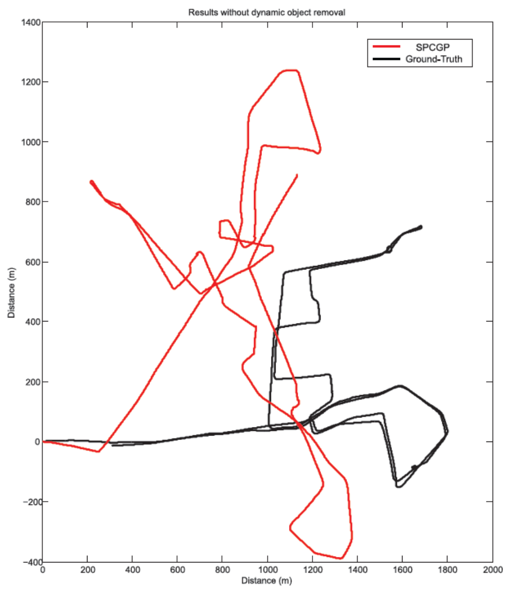
	Cars	Pedestrians	Shadows	Others
Cars	58	3	2	8
Pedestrians	4	43	11	6
Shadows	4	7	21	8
Others	2	3	5	15

Table 5.1 – Confusion matrix for the clustering results.

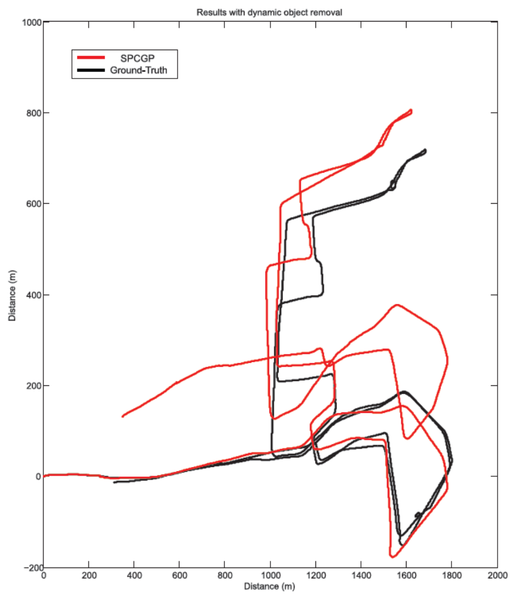
5.4.3 Visual Odometry

We show here the improvement generated by the proposed dynamic object segmentation algorithm in a visual odometry application. As shown in Sec. 4.3.3, the presence of dynamic objects in the environment creates optical flow information that is not generated by camera motion, and therefore will incorporate a component of error into the visual odometry estimation. The purpose of the dynamic object segmentation then becomes the removal of such dynamic objects, to allow the use of only optical flow information belonging to static objects during visual odometry estimation. The same localization results obtained in Sec. 4.3.3, using 14500 images obtained in a highly dynamic urban environment, are shown in Fig. 5.15a, where it is possible to see several sharp turns and systematic drift caused by the presence of dynamic objects. No training and/or model refinement would be able to deal with such shortcomings, so the automatic dynamic object segmentation algorithm described in this chapter was proposed as a way to solve such ambiguities.

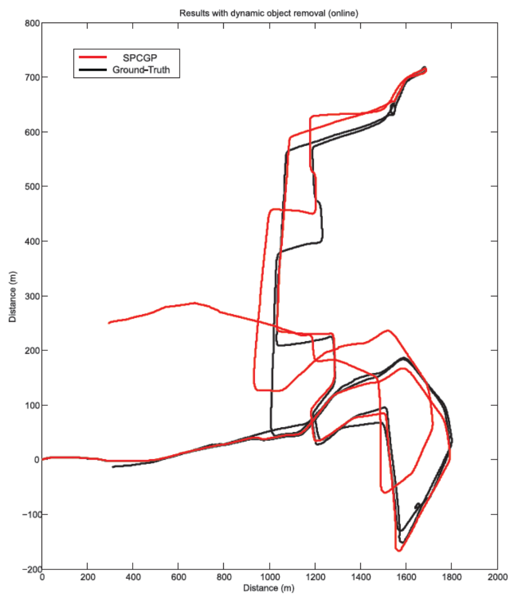
Localization results obtained after incorporating the proposed dynamic object segmentation algorithm are shown in Fig. 5.15b, using the SPCGP approach. Dynamic objects were removed during both the training and testing stages, and there was no hand-labelling or human intervention at any point. As expected, there is still some residual drift caused by the incremental nature of visual odometry estimates, however virtually all sharp turns are removed, allowing the system to recover the



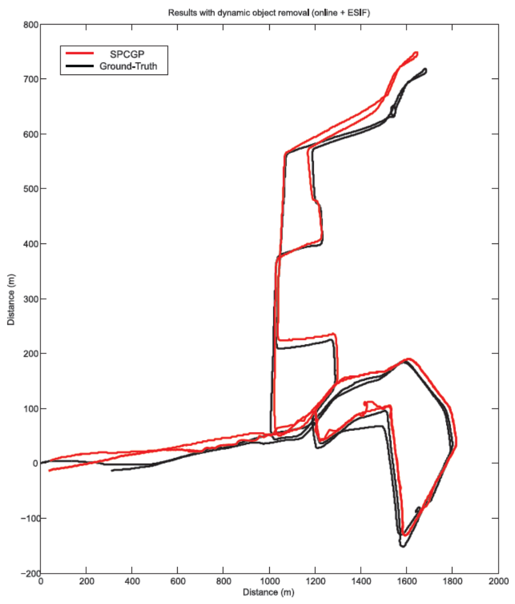
(a)



(b)



(c)



(d)

Figure 5.15 – Localization results in a highly dynamic environment using the SPCGP approach. (a) Without dynamic object removal. (b) With dynamic object removal. (c) With dynamic object removal and online information incorporation. (d) SLAM extension of the localization results obtained using dynamic object removal and online information incorporation.

overall trajectory shape in great detail. Moreover, when the online information incorporation extension is included (Fig. 5.15c), this residual drift decreases even more. We attribute this to the large variation in environment structures and behaviours throughout navigation, since the trajectory is long and comprises several different portions of the city. By gradually incorporating new information as it becomes available, the algorithm is capable of learning these new structures and react accordingly to produce accurate estimates. Finally, Fig. 5.15d shows the localization results obtained using these estimates in a SLAM framework, as described in Sec. 4.2. The loop-closures were mostly done in the lower right portion of the map (beginning and end of the trajectory), and thus most of the drift accumulated during navigation could be removed by globally decreasing uncertainty throughout the entire run.

A quantitative comparison between these methods is given in Table 5.2. It is interesting to note that, due to the sporadic and concentrated nature of errors generated by the presence of dynamic objects, there was not a significant improvement in *rmse* performance. This is to be expected, as these errors were diluted by the frames in which the environment was more conducive with visual odometry estimation. As before, translational error remained roughly the same regardless of which GP approach was used, and rotational error decreased steadily as more extensions were added to the SPCGP framework (which also accounts for the removal of systematic drift caused by dynamic objects), along with the uncertainty in regards to such errors. Also, a different performance metric was introduced, in which the distance between the ground-truth and estimated locations of the vehicle at each frame (in the xy plane) is calculated. This allows a quantitative comparison of these different approaches,

Method	**Trans. Error (rmse) (10^{-2} m)**	**Rot. Error (rmse) (10^{-2} m)**	**xy Eucl. Dist. (rmse)**
SPCGP	6.05 ± 7.91	0.08 ± 0.13	27.82 ± 14.58
SPCGP + DOR	5.94 ± 7.44	0.06 ± 0.09	10.34 ± 6.48
OSPCGP + DOR	5.57 ± 6.97	0.05 ± 0.08	8.57 ± 5.12
ESIF + OSPCGP + DOR	5.71 ± 6.58	0.04 ± 0.05	5.83 ± 3.14

Table 5.2 – Linear and angular errors per frame in experiments with dynamic object removal (DOR).

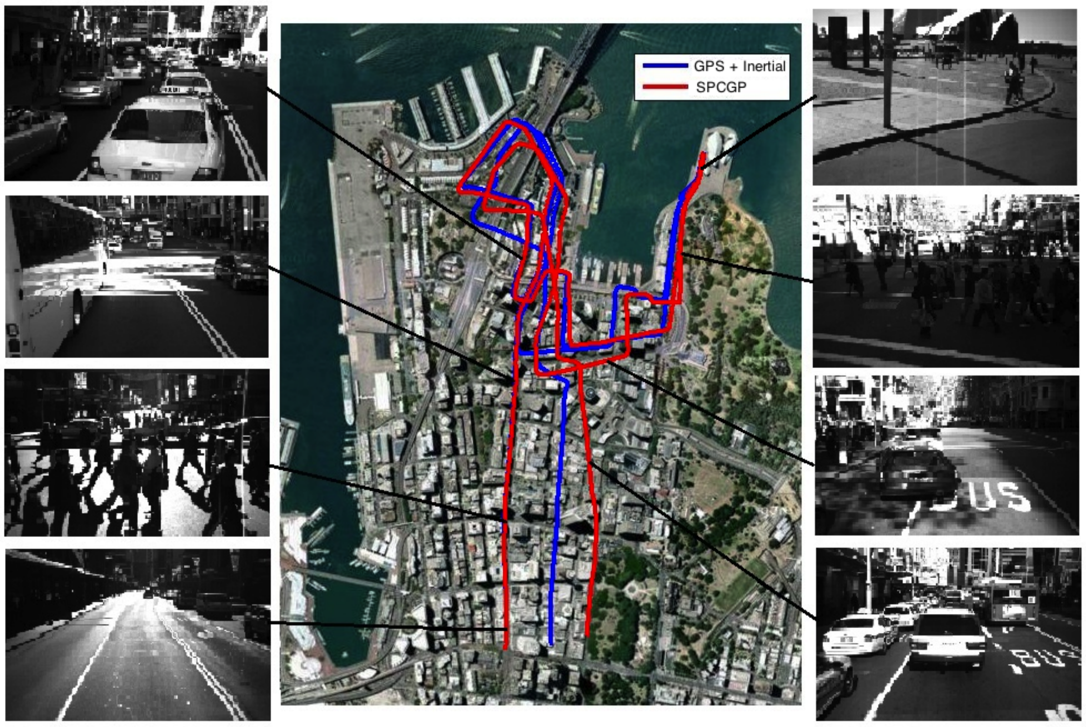


Figure 5.16 – Localization results obtained using the SPCGP approach and dynamic object removal.

since now the few sporadic drifts (especially in angular velocities) will generate a significant cumulative error over time. As expected, the error greatly decreases as new extensions are incorporated, which can be clearly seen in Fig. 5.15.

Finally, localization results obtained using the proposed SPCGP approach with all extensions (except the SLAM framework, since it could in principle be used in conjunction with any visual odometry application) are shown in Fig. 5.16. A map of the environment was superimposed (courtesy of Google Maps) to facilitate the comparison with ground-truth information, and sample frames encountered during navigation are depicted to show some of the challenges faced by the visual odometry algorithm.

5.5 Summary

This chapter provided a novel technique for the automatic segmentation of dynamic objects, conceived as a way to improve visual odometry results in highly dynamic environments. This is done in a non-supervised fashion, without any manual labelling or human intervention whatsoever. The RANSAC algorithm provides an initial classification that is then further refined by the GP framework, to remove outliers and fill in the gaps where there is not enough texture to generate robust features. The result is a dense probabilistic classification of every pixel in the image between static and dynamic, alongside a measurement of confidence in regards to such classification. A novel descriptor was introduced, that both preserves the spatial structure of features in the image and also their colour intensity, thus allowing the precise segmentation of object borders. The non-parametric training is conducted online, starting from an empty covariance matrix and empty hyperparameters. As new information becomes available, it is filtered to remove redundancies and information deemed useful is incorporated into the non-parametric model, which is then able to gradually learn new environment behaviours as it forgets old ones.

Experiments were conducted initially to validate the proposed algorithm's ability to correctly segment dynamic objects from a static background, both when the camera is static and when it is moving alongside the vehicle. Qualitative results were presented, and a quantitative comparison with other techniques was provided using manual labelling of random frames that served as ground-truth. The proposed algorithm outperformed every other technique tested, and also showed a significant improvement over the initial RANSAC classification. Experiments in dynamic object clustering were conducted, providing a further classification where each dynamic object is divided into possibly several categories, such as cars, pedestrians and so forth. Although this clustering in principle has no effect in visual odometry applications, it could lead to a library of dynamic objects that is generated online during navigation, again in an unsupervised manner. Finally, the proposed algorithm was incorporated into a visual odometry application, in which the images used for training and testing were filtered to remove features that are deemed dynamic. Results show a significant improvement over the standard approach of using the entire image to generate optical flow estimates, virtually removing all instances of sharp turns and systematic drifts.

Chapter 6

Conclusion and Future Work

This thesis addresses the problem of visual odometry from a machine learning standpoint, where the transformation between image information and vehicle motion estimates is treated as a regression problem with unknown parameters. By using a Gaussian process we eliminate the need of a known model, using instead a training dataset composed of pairs of images and their corresponding ground-truth information, obtained using a different and independent sensor. This framework eliminates the need for any prior knowledge in regards to the visual system used (including camera calibration parameters), and is capable of recovering scale even in a monocular configuration.

Initially, a basic version of the algorithm is introduced and described, and this basic algorithm is then extended to include several modules that comprise the core of this thesis' contributions. These extensions include: 1) The ability to simultaneously calculate all degrees of freedom in motion from a single pair of images, thus capturing the cross-dependencies between outputs; 2) The introduction of temporal dependency between frames, that increases the amount of information available for inference; 3) Incremental updates of the covariance matrix, which allows the system to iteratively learn new and useful behaviours; 4) A semi-parametric extension to visual odometry that benefits from both the non-parametric GP model and the traditional camera geometric model. An extension to the SLAM framework using information filters was also discussed, where loop-closure is used to globally decrease uncertainty and remove accumulated drift. Finally, a self-supervised technique for the automatic segmentation of dynamic objects is proposed, as a way to remove optical flow information generated by dynamic objects in the environment, which would introduce a component of error that could compromise results.

6.1 Summary of Contributions

1. **Monocular Non-Parametric Visual Odometry Algorithm**
   The problem of visual odometry is addressed from a machine learning perspective, using a Gaussian process framework to learn the transformation function from image information directly into vehicle motion. This approach eliminates the need for a geometric model or even traditional camera calibration, using instead a non-parametric model whose hyperparameters are optimized according to training data and a carefully chosen covariance function. The benefits of this approach are three-fold:

   - **Full Covariance Matrix Recovery**
     Since visual odometry is essentially a multiple-output problem (each image is mapped to more than one degree of freedom in motion), a Multiple-Output Gaussian Process (MOGP) derivation is used for inference. The traditional MOGP derivation is extended to allow the simultaneous estimation of all outputs (linear and angular velocities). This new methodology exploits dependencies between outputs, generated by constraints in vehicle dynamics, to improve accuracy on each individual estimate. These dependencies are quantified as the cross-terms in the resulting full covariance matrix.
   - **Scale Recovery in Monocular Configuration**
     The proposed framework uses training data obtained from a different and independent sensor as ground-truth. If this sensor is capable of scale estimation (i.e. range sensors), this information is encoded into the resulting

non-parametric model and can be recovered by exploiting structure similarities between training and testing images. As the dissimilarity between training and testing data increases, so does the uncertainty inherent to each measurement, indicating that the results, although less accurate, are still valid from a probabilistic standpoint.

- **Natural Treatment of Uncertainties**

  Because Gaussian processes are a Bayesian probabilistic technique, all inferences conducted using the proposed framework will naturally provide a measurement of uncertainty. This measurement of uncertainty is useful in determining the level of confidence that should be put into each estimate, and is of key importance in further treatments of the results, such as data fusion or incorporation into a SLAM scenario.

2. **Temporal Dependencies Between Frames**

   The standard spatial correlation between features in the image used by the Gaussian process framework is extended to include temporal dependencies between features in subsequent frames. Within this new framework, the outputs from each iteration are used as inputs in the next one, thus increasing the amount of information available for inference. This approach works under the assumption that vehicle velocity will change smoothly during navigation, and is especially useful in situations where optical flow information is ambiguous due to the various degrees of freedom available for camera rotation and translation (such as 3D unconstrained navigation).

3. **Semi-Parametric Gaussian Processes**

   The traditional zero-mean assumption in Gaussian processes is modified and a standard Structure from Motion (SFM) geometric model is incorporated into the proposed framework, as the new mean function. The camera calibration parameters are learned simultaneously with the GP hyperparameters, and if they are already available they can be incorporated seamlessly. The geometric model provides an initial estimate that is then further refined by the non-parametric model, and as training data deviates from testing data the geometric model results become more prominent. The result is a semi-parametric approach to visual odometry (and by extension to any other problem in which a previously defined model and training data is available) that benefits from both the information encoded into a parametric model and the flexibility of a non-parametric model.

4. Online Updates of the Covariance Matrix

The semi-parametric model obtained during training is constantly updated during navigation, incorporating new information as it becomes available and discarding redundant or old information, in order to maintain roughly constant computational time. The online update of the semi-parametric model allows the algorithm to gradually adapt to new environments, decreasing the sensitivity to similar training and testing conditions. All new information incorporated is obtained directly from the GP inference process, so there is still no need for a different sensor during navigation, and if another sensor is available (such as inertial data or wheel odometry) this information can be incorporated seamlessly as ground-truth during the update stage.

5. Vectorized Representation of Optical Flow Information

A novel method for representing optical flow information is proposed, as a single vector of fixed dimension. This representation retains the spatial structure of the image, a valuable characteristic since optical flow patterns vary radically and consistently throughout each frame. The clustering of features in different portions of the image, by averaging their optical flow components, also minimizes the impact of false matches and dynamic objects.

6. Automatic Segmentation of Dynamic Objects

A novel technique for the self-supervised segmentation of dynamic objects from a static background is proposed, in which the camera is moving alongside the vehicle. The RANSAC algorithm is used to provide an initial classification, and these results serve as ground-truth to iteratively train a Gaussian process during navigation. No hyperparameter optimization and/or environment knowledge is necessary prior to the beginning of navigation, and new information is incorporated and removed in an online fashion to allow the system to adapt to new circumstances while maintaining a roughly constant computational time. Features deemed dynamic are then removed before the optical flow vectorization stage, to ensure that only a static background is used for visual odometry estimation.

6.2 Future Research

This section briefly describes possible research directions that can be taken based on the work developed throughout this thesis. These research directions aim both to further develop the proposed solution to visual odometry and to take the framework introduced here to different areas of robotics, thus increasing its range of applications.

Covariance Matrix Sparsity

It is natural to assume that the covariance matrix produced during both visual odometry estimation and dynamic object segmentation will be sparse, since features from distant portions of the image will have a near-zero impact on predictions. By utilizing a covariance function with local support (Section 2.2.4) it is possible to exactly round these values to zero, and therefore generate a truly sparse matrix that can be exploited to generate a substantial increase in computational time. Preliminary tests show that, for the automatic dynamic object segmentation algorithm, up to 95% of the covariance matrix can be zeroed out using a sparse covariance function without significantly impacting classification, and similar results should be expected for the visual odometry algorithm as well.

GPU Processing

The use of graphics processing units (GPUs) is becoming increasingly popular as a way to rapidly solve basic computational calculations, such as matrix multiplications [58], that compose the core of more complex tasks. Most of the feature extraction and matching algorithms described in this thesis have already been ported to GPU, achieving real-time performance even in scenarios with thousands of features per frame. The GP framework, due to its marginalization properties, is also highly favourable to parallelization, and therefore could benefit immensely from the use of multiple GPU cores during training and inference.

Data Fusion with Other Sensors

All results obtained in this thesis rely solely on visual information, except during training when a different and independent sensor is used to obtain ground-truth data. During testing, this extra sensor can be removed without impacting the algorithm's performance whatsoever. However, if another sensor is still available during navigation (i.e. wheel encoders, inertial measurements or GPS data), it can be used to further refine the results by incorporating its information into the estimates. The probabilistic nature of Gaussian processes provides the perfect environment for such data fusion, especially within the SLAM framework described in Section 4.2.

Application to Other Areas

Even though the SPCGP framework developed in this thesis was used to address the visual odometry problem, it can in principle be extended to any scenario in which there is a parametric model and available training data. This parametric model can be seamlessly incorporated as the new mean function for the SPCGP framework, providing initial estimates that are then further refined by the non-parametric model, based on the chosen covariance function and training data. Since these semi-parametric estimates converge to the purely parametric estimates as training data deviates from testing data, results will in the worst case scenario be as accurate as the ones obtained without the non-parametric component, and therefore this approach can only improve results.

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover Publications, 1965.

[2] M. Agrawal and K. Konolige. Rough terrain visual odometry. *International Conference on Advanced Robotics (ICAR)*, August 2007.

[3] A. Aizerman, M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[4] D.-L. Almanza-Ojeda and M.-A. Ibarra-Manzano. *Recent Advances in Mobile Robotics*, chapter 3D Visual Information for Dynamic Objects Detection and Tracking During Mobile Robot Navigation. InTech Publishing, 2011.

[5] K. Arras, M. Mozos, and W. Burgard. Using boosted features for the detection of people in 2d range data. *International Conference on Robotics and Automation (ICRA)*, April 2007.

[6] A. Bak, S. Bouchafa, and D. Aubert. *Dynamic Objects Detection Through Visual Odometry and Stereo-Vision: A Study of Inaccuracy and Improvement Sources.* Machine Vision and Applications, special issue on Car Navigation and Vehicles Systems, 2011.

[7] H. Bay, T. Tuytelaars, and L. van Gool. Surf: Speeded up robust features. *European Conference on Computer Vision (ECCV)*, May 2006.

[8] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1992.

[9] S. Botelho, P. Drews, G. Oliveira, and M. Figueiredo. Visual odometry and mapping for underwater autonomous vehicles. *Latin American Robotics Symposium (LARS)*, October 2009.

[10] P. Boyle and M. Frean. Multiple output gaussian process regression. Technical report, University of Wellington, 2005.

[11] P. Boyle and M. Frean. Dependent gaussian processes. *Advances in Neural Information Processing Systems*, 17:217–224, 2005.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[13] M. Brown and D. G. Lowe. Recognising panoramas. *International Conference on Computer Vision (ICCV)*, 2:1218–1225, 2003.

[14] W. Buntine and A. Weigend. Bayesian backpropagation. *Complex Systems*, 5: 603–642, 1991.

[15] Y. Cheng, M. Maimone, and L. Matthies. Visual odometry on the mars exploration rovers. *International Conference on Systems, Man and Cybernetics*, October 2005.

[16] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, 2005.

[17] J. Civera, D. R. Bueno, A. J. Davison, and J. M. M. Montiel. Camera self-calibration for sequential bayesian structure from motion. *International Conference on Robotics and Automation (ICRA)*, May 2009.

[18] P. Corke, D. Strelow, and S. Singh. Omnidirectional visual odometry for a planetary rover. *International Conference on Intelligent Robots and Systems (IROS)*, September 2004.

[19] I. J. Cox. A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision (IJCV)*, 10:53–66, 1993.

[20] N. Cressie. *Statistics for Spatial Data.* Wiley Series in Probability and Statistics, 1993.

[21] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[22] Lehel Csato and Manfred Opper. Sparse on-line gaussian processes. Technical report, Massachusetts Institute of Technology, 2002.

[23] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 2005.

[24] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence (PAMI)*, 2007.

[25] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, August 2007.

[26] F. Dellaert. The expectation maximization algorithm. Technical report, Georgia Institute of Technology, 2002.

[27] C. Dima. Active learning for outdoor perception. Technical report, Carnegie Mellon University, 2006.

[28] J. J. Dongara, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK's User Guide*. Society for Industrial and Applied Mathematics, 1979.

[29] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation. *International Conference on Robotics and Automation (ICRA)*, May 2000.

[30] T. Ellis and M. Xu. Object detection and tracking in an open and dynamic world. *Workshop on Performance Evaluation of Tracking and Surveillance*, June 2001.

[31] A. Ess, B. Leibe, and L. van Gool. Depth and appearance for mobile scene analysis. *International Conference on Computer Vision (ICCV)*, October 2007.

[32] A. Ess, B. Leibe, K. Schindler, and L. Van Gool. Moving obstacle detection in highly dynamic scenes. *International Conference on Robotics and Automation (ICRA)*, May 2009.

[33] R. M. Eustice, H. Singh, and J. Leonard. Exactly sparse delayed-state filters for view-based slam. *IEEE Transactions on Robotics*, 22:1100–1114, 2006.

[34] O. D. Faugeras, Q.-T. Luong, and S. J. Maybank. Camera self-calibration: Theory and experiments. *European Conference on Computer Vision (ECCV)*, May 1992.

[35] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–395, 1981.

[36] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21: 196–207, 2004.

[37] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. *Conference on Uncertainty in Artificial Intelligence (UAI)*, August 1997.

[38] M. Gibbs. *Bayesian Gaussian Processes for Classification and Regression.* PhD thesis, Cambridge University, 1997.

[39] M. Gibbs and D. J. C. MacKay. Efficient implementation of gaussian processes. Technical report, Cambridge University, 1997.

[40] I. Gordon and D. G. Lowe. *Toward Category-Level Object Recognition*, chapter What and Where: 3D Object Recognition with Accurate Pose. Springer-Verlag Berlin Heidelberg, 2006.

[41] V. Guizilini and F. Ramos. Multi-task learning of visual odometry estimators. *12th International Symposium on Experimental Robotics (ISER)*, December 2010.

[42] V. Guizilini and F. Ramos. Visual odometry learning for unmanned aerial vehicles. *International Conference on Robotics and Automation (ICRA)*, May 2011.

[43] V. Guizilini and F. Ramos. Semi-parametric models for visual odometry. *International Conference on Robotics and Automation (ICRA)*, May 2012.

[44] V. Guizilini and F. Ramos. Online self-supervised segmentation of dynamic objects. *International Conference on Robotics and Automation (ICRA)*, May 2013.

[45] A. Hald. On the history of maximum likelihood in relation to inverse probability and least squares. *Statistical Science*, 14:214–222, 1999.

[46] C. Harris and M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, February 1988.

[47] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2004.

[48] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 1997.

[49] D. Higdon. *Quantitative Methods for Current Environmental Issues*, chapter Space and Space-Time Modeling Using Process Convolutions. Springer-Verlag London, 2002.

[50] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1980.

[51] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993.

[52] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. *International Conference on Intelligent Robots and Systems (IROS)*, September 2008.

[53] M. Keck Jr., J. Davis, and A. Tyagi. Tracking mean shift clustered point clouds for 3d surveillance. *International Workshop on Video Surveillance and Sensor Networks*, October 2006.

[54] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, and A. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2002.

[55] Y. Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 2004.

[56] J. Kelly and G. Sukhatme. An experimental study of aerial stereovisual odometry. *Symposium on Intelligent Autonomous Vehicles*, September 2007.

[57] J. Kelly, S. Saripalli, and G. Sukhatme. Combined visual and inertial navigation for an unmanned aerial vehicle. *International Conference on Field and Service Robotics (FSR)*, July 2007.

[58] J. T. Kider, M. Henserson, M. Likhachev, and A. Safonova. High-dimensional planning on the gpu. *International Conference on Robotics and Automation (ICRA)*, May 2010.

[59] S. Kim, K.-J. Yoon, and I. S. Kweon. Object recognition using a generalized robust invariant feature and gestalt's law of proximity and similarity. *Pattern Recognition*, 41:726–741, 2008.

[60] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[61] B. M. Kitt, J. R., A. D. Chambers, M. Schonbein, H. Lategahn, and S. Singh. Monocular visual odometry using a planar road model to solve scale ambiguity. *European Conference on Mobile Robots (ECMR)*, October 2011.

[62] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell. Towards robust automatic traffic scene analysis in real-time. *International Conference on Pattern Recognition (ICPR)*, October 1994.

[63] S. Lazebnik, C. Schmid, and J. Ponce. Semi-local affine parts for object recognition. *British Machine Vision Conference (BMCV)*, September 2004.

[64] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision (IJCV)*, 77:259–289, 2008.

[65] T. Lemaire, C. Berger, I.-K. Jung, and S. Lacroix. Vision-based slam: Stereo and monocular approaches. *International Journal of Computer Vision (IJCV)*, 74:343–364, 2007.

[66] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision (IJCV)*, 30:79–116, 1998.

[67] H. C. Longuet-Higgins. *A computer algorithm for reconstructing a scene from two projections.* Morgan Kaufmann Publishers Inc., 1987.

[68] D. G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision (ICCV)*, November 1999.

[69] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60:91–110, 2004.

[70] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 1994.

[71] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *DARPA Image Understanding Workshop*, April 1981.

[72] D. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, 1992.

[73] D. MacKay. Bayesian interpolation. *Neural Computation*, 1992.

[74] D. MacKay. Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–510, 1995.

[75] D. MacKay. Hyperparameters: Optimize or integrate out. *Maximum Entropy and Bayesian Methods*, 11:1035–1068, 1996.

[76] D. MacKay. Introduction to gaussian processes. *Neural Networks and Machine Learning*, 1998.

[77] D. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[78] B.S. Manjunath, P. Salembier, and T. Sikora. *Introduction to MPEG-7: Multimedia Content Descriptor Interface.* Wiley et. Sons, 2002.

[79] G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.

[80] MATLAB. *version 7.13 (R2011b).* The MathWorks Inc., 2011.

[81] A. Melkumyan and F. Ramos. A sparse covariance function for exact gaussian process inference in large datasets. *Iinternational Joint Conference on Artifical Intelligence (IJCAI)*, October 2009.

[82] M. Milford. Single camera vision-only slam on a suburban road network. *International Conference on Robotics and Automation (ICRA)*, May 2008.

[83] T. Minka. Expectation propagation for approximate bayesian inference. *Association for Uncertainty in Artificial Intelligence (AUAI)*, August 2001.

[84] A. Monnet, A. Mittal, N. Paragios, and V. Ramesh. Background modelling and subtraction of dynamic scenes. *International Conference on Computer Vision (ICCV)*, October 2003.

[85] A. Moore. An introductory tutorial on kd-trees. Technical report, Cambridge University, 1991.

[86] H. P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* PhD thesis, Stanford University, 1980.

[87] H. P. Moravec and D. B. Gennery. Cart project progress report. Technical report, Stanford University, July 1976.

[88] R. K. Namdev, A. Kundu, K. M. Krishna, and C. V. Jawahar. Motion segmentation of multiple objects from a freely moving monocular camera. *International Conference on Robotics and Automation (ICRA)*, May 2012.

[89] R. Neal. Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, University of Toronto, 1992.

[90] R. M. Neal. *Bayesian Learning for Neural Networks.* Lecture Notes in Statistics, Springer-Verlag, 1996.

[91] S. Nedevschi, R. Danescu, D. Frentiu, T. Graf, and R. Schmidt. High accuracy stereovision approach for obstacle detection on non-planar roads. *IEEE Proceedings on Intelligent Engineering Systems (INES)*, October 2004.

[92] D. Nister. Preemptive ransac for live structure and motion estimation. *Machine Vision and Applications*, 2005.

[93] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23:3–20, 2006.

[94] A. O'Hagan. Curve fitting and optimal design for prediction (with discussion). *Journal of the Royal Statistical Society*, 40:1–42, 1978.

[95] N. Oliver, B. Rosario, and A. Pentland. A bayesian computer vision system for modelling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, August 2000.

[96] M. A. Osborne, S. J. Roberts, A. Rogers, S. D. Ramchurn, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. *International Conference on Information Processing in Sensor Networks (IPSN)*, April 2008.

[97] C. Paciorek. *Nonstationary Gaussian Processes for Regression and Spatial Modelling.* PhD thesis, Carnegie Mellon University, 2003.

[98] A. Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill Companies, 3rd Edition, 1991.

[99] M. A. Paskin. Thin junction tree filters for simultaneous localization and mapping. *International Joint Conference on Artificial Intelligence (IJCAI)*, August 2003.

[100] J. C. Platt. *Probabilities for SV Machines.* Advances in Large Margin Classifiers, MIT Press, 2000.

[101] C. E. Rasmussen and K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[102] R. Roberts, H. Nguyen, N. Krishnamurthi, and T. Balch. Memory-based learning for visual odometry. *International Conference on Robotics and Automation (ICRA)*, May 2008.

[103] R. Roberts, C. Potthast, and F. Dellaert. Learning general optical flow subspaces for egomotion estimation and detection of motion anomalies. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 2009.

[104] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[105] H. Samet. *Foundations of Multidimensional and Metric Data Structures (Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).* Morgan Kaufmann Publishers Inc., 2005.

[106] D. Scaramuzza. Performance evaluation of 1-point-ransac visual odometry. *Journal of Field Robotics*, 2011.

[107] D. Scaramuzza and F. Fraundorfer. Visual odometry tutorial. *Robotics and Automation Magazine*, 18:80–92.

[108] D. Scaramuzza and R. Siegwart. Appearance guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE Transactions on Robotics, special issue on Visual SLAM*, October 2008.

[109] D. Scaramuzza, F. Fraundorfer, M. Pollefeys, and R. Siegwart. Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints. *International Conference on Computer Vision (ICCV)*, October 2009.

[110] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, 2001.

[111] B. Scholkopf and A. J. Smola. *Learning with Kernels.* The MIT Press, 2002.

[112] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. *International Conference on Robotics and Automation (ICRA)*, May 2001.

[113] M. Seeger. Gaussian processes for machine learning. *International Journal of Neural Systems*, 14:69–106, 2004.

[114] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison, 2010.

[115] Y. Sheikh and M. Shah. Bayesian modelling of dynamic scenes for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, February 2005.

[116] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[117] J. Shi and C. Tomasi. Good features to track. *Computer Vision and Pattern Recognition Conference (CVPR)*, June 1994.

[118] M. Smith, I. Posner, and P. Newman. Efficient non-parametric surface representations using active sampling for push broom laser data. *Robotics: Science and Systems (RSS)*, June 2010.

[119] M. Soga, T. Kato, M. Ohta, and Y. Ninomiya. Pedestrian detection with stereo vision. *International Conference on Data Engineering*, July 2005.

[120] L. Spinello, R. Triebel, and R. Siegwart. Multimodal people detection and tracking in crowded scenes. *International Conference on Artificial Intelligence (Physically Grounded AI Track)*, July 2008.

[121] C. Stauffer and W. Grimson. A probabilistic background model for tracking. *European Conference on Computer Vision (ECCV)*, August 2000.

[122] C. Stauffer and W. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, August 2000.

[123] M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics, 1999.

[124] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Buhmann. Topology free hidden markov models: Application to background modeling. *European Conference on Computer Vision (ECCV)*, June 2000.

[125] Z. Sun, J. van de Ven, F. Ramos, X. Mao, and H. Durrant-Whyte. Inferring laser-scan matching uncertainty with conditional random fields. *Robots and Autonomous Systems*, 60:83–94, 2012.

[126] N. Sunderhauf, K. Konolige, S. Lacroix, and P. Protzel. *Visual Odometry using Sparse Bundle Adjustment on an Autonomous Outdoor Vehicle*. Tagungsband Autonome Mobile Systeme. Springer Verlag, 2005.

[127] A. Taluker and L. Matthies. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. *Intelligent Robots and Systems (IROS)*, September 2004.

[128] J.-P. Tardif, Y. Pavlidis, and K. Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. *International Conference on Intelligent Robots and Systems (IROS)*, September 2008.

[129] S. Thrun. Particle filters in robotics. *Conference on Uncertainty in Artificial Intelligence (UAI)*, August 2002.

[130] S. Thrun, Y. Liu, A. Koller, Z. Ng, and H. Durrant-Whyte. Simultaneous mapping and localization with sparse extended information filters. *International Journal of Robotics Research (IJRR)*, 23:693–716, 2004.

[131] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999.

[132] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. *International Conference on Computer Vision (ICCV)*, September 1999.

[133] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particle filter. *Advances in Neural Information Processing Systems (NIPS)*, December 2001.

[134] S. Vasudevan, F. Ramos, E. Nettleton, and H. Durrant-whyte. Non-stationary dependent gaussian processes for data fusion in large-scale terrain modeling. *International Conference on Robotics and Automation (ICRA)*, May 2011.

[135] G. Wahba. Spline models for observational data. *Conference Series in Applied Mathematics*, January 1990.

[136] M. R. Walter, R. M. Eustice, and J. J. Leonard. Exactly sparse extended information filters for feature-based slam. *International Journal of Robotics Research (IJRR)*, 26:335–359, 2007.

[137] H. Wang, K. Yuan, W. Zou, and Q. Zhou. Visual odometry based on locally planar ground assumption. *International Conference on Information Acquisition*, 2005.

[138] L. Wasserman. *All of Nonparametric Statistics*. Springer Texts in Statistics, 2007.

[139] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, University of North Carolina, 1995.

[140] C. K. Williams and D. Barber. Bayesian classification with gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, November 1998.

[141] C. K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10:1203–1216, 1998.

[142] J. Wolberg. *Data Analysis Using the Method of Least Squares: Extracting the Most Information from Experiments*. Springer Verlag, 2005.

[143] C. Wren, A. Azarbayejani, T. Darrel, and A. Pentland. Pfinder: Real time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, July 1997.

[144] L. Zhao and C. Thorpe. Stereo and neural network-based pedestrian detection. *IEEE Transactions on Intelligent Transportation Systems*, pages 148–154, 2000.

[145] J. Zhong and S. Sclaroff. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. *International Conference on Computer Vision (ICCV)*, October 2003.

[146] Z. W. Zhu, T. Oskiper, O. Naroditsky, S. Samarasekera, H. S. Sawhney, and R. Kumar. An improved stereo-based visual odometry system. *Workshop of Performance Metrics for Intelligent Systems*, August 2006.