Support Vector Machines in a real time tracking architecture

Benjamin Castaneda

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Support Vector Machines in a Real Time Tracking Architecture

by

Benjamin Castaneda

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Approved by:

Juan C. Cockburn
Thesis Advisor
Dr. Juan C. Cockburn, Associate Professor of Computer Engineering

Andreas Savakis
Committee Member
Dr. Andreas Savakis, Associate Professor of Computer Engineering

M. Shaaban
Committee Member
Dr. Muhammad Shaaban, Assistant Professor of Computer Engineering

Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
June 2004
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: Support Vector Machines in a Real Time Tracking Architecture

I, Benjamin Castañeda, hereby grant permission to the Wallace Memorial Library reproduce my thesis in whole or part.

Benjamin Castañeda
Benjamin Castañeda

06/30/2004
Date
Dedication

To my parents Benjamin and Georgina, and my goddaughter Micaela
I would like to take this opportunity to thank several people who helped me in the creation process of this Thesis. Dr. Juan Carlos Cockburn, my advisor and my friend who supported and guided me through all the studies for my Master's degree. Dr. Muhammad Shaaban and Dr. Andreas Savakis for the advice and motivation they gave me as my professors and members of my committee. Dr. Moises Sudit for the fruitful discussions we had. Staff and Faculty of the Computer Engineering Department at RIT, for their everyday help. Dr. Roger Gaborski and Jeffrey Keller from the Laboratory for Applied Computing who allowed me to run my simulations on their computing cluster. Matthew Woitaszek for providing the initial code in which I based my parallel implementations. Elisa, Juan Carlos, Thomas, Luisa and Roberto, for being my family in Rochester. Finally, I thank my friends who patiently supported me these years away from home.
Abstract

Support Vector Machines in a Real Time Tracking Architecture

Benjamin Castañeda

Supervising Professor: Dr. Juan C. Cockburn

The standard approach to tracking an object of interest in a video stream is to use an object detector, a classifier and a tracker in sequential order. This work investigates the use of Support Vector Machines (SVM) as classifiers for real-time tracking systems, combining them with Kalman Filter predictors. Support Vector Machines have been proved successful in a variety of classification tasks such as recognizing faces, cars, handwriting and others. However their use has been hampered by the complexity and computational time involved in the training and classification stages. In recent years new methods and techniques for training and classification of Support Vector Machines have been discovered making possible their utilization in real-time applications. These methods have been explored and improved resulting in a framework for fast prototyping and development of real-time tracking systems. New optimal and sub-optimal methods for parallel SVM training based on biased and unbiased versions of the Sequential Minimal Optimization algorithm are presented. They provide a trade-off between time performance and accuracy. Time performance in the classification stage is significantly improved by reducing the number of support vectors with almost no loss in accuracy. New methods to allow the reduction with different kernels are presented. The effectiveness of the approach developed is demonstrated in a face tracking problem where the objective is to track the lips and eyes of a subject in a video stream in real-time.
Contents

Dedication ................................................................. iv

Acknowledgments ......................................................... v

Abstract ................................................................. vi

Glossary ........................................................................ xii

1 Introduction ............................................................... 1

2 Background Theory ...................................................... 7
   2.1 Support Vector Machines ........................................... 7
      2.1.1 Support Vector Machines as Classifiers .................... 7
      2.1.2 Training Support Vector Machines ......................... 11
      2.1.3 Output Evaluation ............................................. 12
   2.2 A Real-time Face-tracking System ............................... 13
      2.2.1 Tracking Architecture .......................................... 14
      2.2.2 Face Tracking Implementation ............................... 15
      2.2.3 SVMs in the Face Tracking System ......................... 16

3 Parallel SVM Training .................................................. 19
   3.1 Support Vector Machine Training ............................... 20
      3.1.1 SMO advantages .............................................. 22
   3.2 Sequential Minimal Optimization ............................... 23
      3.2.1 Analytical optimization of two Lagrange multipliers .... 24
      3.2.2 Selecting two multipliers ................................... 26
      3.2.3 Unbiased SMO ............................................... 27
   3.3 Previous Work on SMO Parallelization ......................... 27
      3.3.1 Interleaved Parallelization (IP) ............................ 28
      3.3.2 Biased Sub-Optimal Blocked Parallelization (BSOBP) .... 30
      3.3.3 Biased Optimal Blocked Parallelization (BOBP) ........... 31
   3.4 New Approaches .................................................... 32
      3.4.1 Unbiased Sub-Optimal Blocked Parallelization (USOBP) ... 33
      3.4.2 Modified Biased Sub-Optimal Blocked Parallelization (MBSOBP) 34
      3.4.3 Chunking - SMO (ChSMO) ................................ 35
3.5 Results ................................................................. 37
  3.5.1 Results on Sub-Optimal Solutions (BSOBP, MBSOBP and USOBP) 38
  3.5.2 Results on Optimal Solutions (BOBP and ChSMO) .............. 42
3.6 Summary and Discussion ........................................ 43

4 Support Vector Reduction ........................................ 46
  4.1 The Reduced Support Vector Problem .......................... 47
  4.2 Current Approaches ............................................ 49
    4.2.1 Exact Simplification ..................................... 50
    4.2.2 Gaussian Kernel Reduction Method ....................... 52
  4.3 Proposed Methods ............................................. 57
    4.3.1 Data Sets used for testing .............................. 58
    4.3.2 Explicit Mapping ......................................... 60
    4.3.3 Modified Exact Simplification ......................... 64
    4.3.4 Simulated Annealing Method ............................. 70
    4.3.5 Polynomial Kernel Reduction ........................... 75
    4.3.6 Standard Optimization Method .......................... 82
  4.4 Summary and Discussion ...................................... 86

5 Tracking ............................................................. 89
  5.1 Kalman filter Theory ......................................... 92
  5.2 Kalman filter Parameter Selection ........................... 94
  5.3 Results .......................................................... 98
  5.4 Summary and Discussion ...................................... 101

6 Conclusions and Future Work .................................. 104
  6.1 Future Work .................................................... 107

Bibliography .......................................................... 108
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Results of the tuning experiments</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>SVM Classification Time</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of SMO, SVMlight and Chunking algorithms in time and scaling.</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Source [30]</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Speed-up for the Adult UCI database using Sub-Optimal Approaches</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>Training time in seconds using sub-optimal approaches for the facial feature data set. * Predicted for all approaches, + predicted for BSOBP and MBSOBP</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Increment in the number of support vectors for the sub-optimal approaches</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>Loss in Accuracy for sub-optimal approaches</td>
<td>41</td>
</tr>
<tr>
<td>3.6</td>
<td>Speed Up in the Adult UCI Database for the optimal approaches</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Modified Exact Solution: Results for the lip classification problem</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>Modified Exact Solution results using sorted support vectors</td>
<td>68</td>
</tr>
<tr>
<td>4.3</td>
<td>Simulated Annealing Method: Results for the Simple Data Set</td>
<td>73</td>
</tr>
<tr>
<td>4.4</td>
<td>Polynomial Kernel Reduction: Comparison between random initialization and support vector initialization</td>
<td>80</td>
</tr>
<tr>
<td>4.5</td>
<td>Polynomial Kernel Reduction: Results for the Simple Data Set</td>
<td>82</td>
</tr>
<tr>
<td>5.1</td>
<td>SVM Classification Time</td>
<td>98</td>
</tr>
<tr>
<td>5.2</td>
<td>SVM and Kalman Filters Tracking Time</td>
<td>99</td>
</tr>
<tr>
<td>5.3</td>
<td>Results of reducing the number of Support Vectors</td>
<td>101</td>
</tr>
<tr>
<td>5.4</td>
<td>Reduced Support Vectors Classification Time</td>
<td>101</td>
</tr>
<tr>
<td>5.5</td>
<td>RSV and Kalman Filters</td>
<td>101</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Standard Tracking Architecture ........................................... 2  
2.1 SVM decision boundary ......................................................... 8  
2.2 Example of non-linear mapping ............................................ 10  
2.3 Feature Based Tracking Architecture .................................... 14  
3.1 SMO Main Routine ............................................................. 24  
3.2 Training Constraints for 2 Lagrange multipliers ....................... 25  
3.3 Division of the training set in $n$ segments ................................ 29  
3.4 Three processor Interleaved Parallelization ............................. 29  
3.5 Biased Sub-Optimal Blocked Parallelization ............................ 31  
3.6 Biased Optimal Blocked Parallelization ................................ 32  
3.7 Chunking SMO approach ....................................................... 36  
3.8 Sub-optimal approaches: Speed-up for the Adult UCI database ........ 38  
3.9 Increment in the number of support vectors for sub-optimal approaches ........................................... 41  
3.10 Loss in accuracy for the sub-optimal approaches ....................... 42  
3.11 Optimal Approaches: Speed-up for the Adult UCI database ........... 43  
4.1 Exact Simplification Algorithm .............................................. 51  
4.2 Gaussian Kernel Reduction Method: Iterative Algorithm ............. 54  
4.3 Simple Data Set ................................................................. 59  
4.4 Explicit Mapping ............................................................... 63  
4.5 Linearly Independent Support Vectors in Feature Space for the Simple Data Set ........................................... 64  
4.6 Angle $\theta$ ...................................................................... 65  
4.7 Modified Exact Simplification Method ..................................... 66  
4.8 Modified Exact Simplification Method: Results for the Lip Classification Problem ........................................... 69  
4.9 Simulated Annealing Method ................................................. 71  
4.10 Simulated Annealing Method: Results for the Simple Data Set ....... 74  
4.11 Polynomial Kernel Reduction ............................................... 77  
4.12 Polynomial Kernel Reduction: Subroutine to find $z$ .................. 79  
4.13 Comparison between support vectors and random vectors as starting points in the Lip Classification Problem ........................................... 80  
4.14 Polynomial Kernel Reduction: Performance Results for the Lip Classification Problem ........................................... 83
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.15</td>
<td>Standard Optimization Method Algorithm</td>
<td>84</td>
</tr>
<tr>
<td>4.16</td>
<td>SOM: Results for the Lip Classification Problem using the quasi Newton Optimization Method</td>
<td>85</td>
</tr>
<tr>
<td>4.17</td>
<td>Comparison of the Polynomial Kernel Reduction and the Standard Optimization Method. Results for the Lip Classification Problem</td>
<td>86</td>
</tr>
<tr>
<td>5.1</td>
<td>Feature Based Tracking Architecture</td>
<td>90</td>
</tr>
<tr>
<td>5.2</td>
<td>Kalman filter recursive implementation</td>
<td>94</td>
</tr>
<tr>
<td>5.3</td>
<td>Distribution of Computation</td>
<td>100</td>
</tr>
<tr>
<td>5.4</td>
<td>Distribution of Computation using Reduced Support Vectors</td>
<td>102</td>
</tr>
</tbody>
</table>
Glossary

**Biased Optimal Blocked Parallelization (BOBP)**  A parallel algorithm for training SVMs based on the biased version of Platt’s SMO algorithm. The result of the training process is the global minimum of the SVM training problem. It was developed by Etin and Elias in 2001.

**Biased Sub-Optimal Blocked Parallelization (BSOBP)**  A parallel algorithm for training SVMs based on the biased version of Platt’s SMO algorithm. The result of the training process is a feasible point of the SVM training problem but it is not the global minimum. It was developed by Etin and Elias in 2001.

**Chunking - SMO (ChSMO)**  A new parallel algorithm for training SVMs based on the biased version of Platt’s SMO algorithm and the Chunking algorithm. The result of the training process is the global minimum of the SVM training problem.

**Chunking**  A sequential algorithm for training Support Vector Machines developed by Vapnik. It is based on the idea of discarding non-support vectors early in the training process.

**Kalman Filter**  The Kalman filter is an algorithm for sequentially updating a linear projection for a dynamic system that is in state-space representation. In this work, Kalman Filters are used as optimal estimators for tracking purposes.

**Karush-Kuhn-Tucker conditions (KKT conditions)**  Necessary and sufficient conditions for the termination of a SVM training process with an optimal solution.

**Modified Biased Sub-Optimal Blocked Parallelization (MBSOBP)**  A new parallel algorithm for training SVMs based on Etin’s BSOBP algorithm. The result of the training process is a feasible point of the SVM training problem but it is not the global minimum.

**Osuna’s Algorithm**  A method for training SVMs by dividing the QP training problem in a series of smaller QP sub-problems which size is fixed. It was developed by Osuna in 1997.
Quadratic Programming Problem (QP problem)  An optimization problem in which the objective function is convex and quadratic and it is subject to linear constraints. The solution of the optimization problem is unique. Training Support Vector Machines amounts to solving a Quadratic Programming Problem.

Reduced Support Vector Problem (RSVP)  The problem of approximating a Support Vector Machine Classifier with another one which has less number of support vectors and similar performance in accuracy.

Sequential Minimal Optimization (SMO)  A method for training SVMs by analytically optimizing two vectors at a time instead of using a numeric quadratic problem solver. It was developed by Platt in 1998.

Support Vector (SV)  A feature vector selected through optimization for inclusion in a Support Vector Machine model.

Support Vector Machine (SVM)  A machine learning algorithm based statistical learning theory developed by Vapnik and Chervonenkis.

SVMLight  A method for training SVMs by dividing the QP training problem in a sequence of smaller QP sub-problems which size is determined by heuristics. It was developed by Joachims in 1999.


Unbiased Sub-Optimal Blocked Parallelization (USOBP)  A new parallel algorithm for training Support Vector Machines based on the unbiased version of Platt's SMO algorithm. The result of the training process is a feasible point of the QP training problem but it is not the global minimum.
Chapter 1

Introduction

Human-Computer Interaction has received lately renewed interest due to advances in technology. Applications, which could have been considered fiction a decade ago, are now possible due to smaller, cheaper and more powerful computing devices. For example, smart rooms could have smart cameras in charge of tracking the user so his/her gestures and movements are evaluated as input commands. It is apparent that visual tracking should be a fundamental capability of the system, and since the interaction between the room and the user is required, this should be accomplished in real-time.

A common approach to tracking an object in a video stream is to use an object detector, a classifier and a motion estimator or tracker in sequential order. The object detector scans a frame from the video stream and selects the candidates to be analyzed by the classifier. The classifier evaluates every candidate assigning it a measure that indicates the likeliness of the candidate to be the object searched. The candidate with the best score is then locked and the tracker is used to follow it through the field of view. This standard tracking architecture, represented in Figure 1.1, has been successfully implemented in a variety of applications [20, 39, 25, 38, 23, 17, 2].
Figure 1.1: Standard Tracking Architecture

The classifier plays an important role in the overall performance of the tracking system. A poor classifier will lead to bad detection accuracy and false locks.

In recent years, Support Vector Machines have been a breakthrough in the machine learning community. They have been proved effective in a variety of tasks related to classification such as Character Recognition, Image Rotation Detection, Gesture Recognition, Face Recognition, Bioinformatics, and Spam Classification (See [43, 13, 8]).

Support Vector Machines’ performance in accuracy and generalization makes them an ideal candidate for this standard tracking architecture. However, in order to obtain their best performance, different issues should be addressed: feature selection, tuning, training and
classification complexity. The first two influence directly the classifier accuracy and generalization properties. The last two have an impact on the implementation of SVMs.

The features given as input to the classifier are of extreme importance to its accuracy. If the wrong set of features are used, not even the best classifier would be able to perform correctly. Selecting the correct input is problem dependent and the overall performance of the system depends on this step.

Support Vector Machines’ performance is also affected by the selection of its two parameters: the kernel and the limiting value. The meaning and influence of these two parameters over the classifier accuracy is explained in Chapter 2. Tuning or finding the correct parameters is also problem dependent.

As a supervised learning machine, Support Vector Machines need to be trained. The training process amounts to solving a Quadratic Programming (QP) problem. Due to the size of the problem, standard QP solvers cannot be used. This training complexity has limited the widespread usage of support vector machines. Even though new training algorithms have been developed recently, their time performance is problem dependent and the computational time needed is still large.

The training process gives as a result a set of vectors, called support vectors, which characterize the decision function of the classifier. The computational time involved in the evaluation of the classifier is directly proportional to the number of support vectors. This number is problem dependent but empirical results have shown that it is approximately between 10% and 20% of the number of training vectors in the training set. Since the usual training set contains large amounts of data, the number of support vectors is also large.
Therefore, SVMs’ time performance in test phase is less than other classifiers such as Neural Networks.

This work explores the use of Support Vector Machines (SVMs) as classifiers in real-time tracking systems. From the issues presented above, it addresses the ones which can be applied to any visual tracking problem, namely reducing the training time and computational time involved in classification. As a result, this work presents a framework for real-time visual tracking applications using SVMs.

Training SVMs involves solving a quadratic optimization problem. Since the training sets are usually very large, the computational complexity involved in training has limited its application to practical problems, specially in the area of image processing. Current advances in parallel computing are bringing to the user cost effective tools to attack complex problems such as this one. In this work the use of parallel computing techniques to reduce the training time was investigated.

New optimal and sub-optimal Parallel trainers based on Platt’s Sequential Minimal Optimization (SMO) algorithm are proposed and compared to previous work on the subject. The Unbiased Sub-Optimal Blocked Parallelization introduces the use of the unbiased version of the SMO algorithm in parallel trainers. It presents a trade-off between accuracy and time performance and it is specially useful for large training sets. The Modified Biased Sub-Optimal Blocked Parallelization modifies the miscalculation of a SVM parameter in a previous algorithm improving its accuracy. Finally, the SMO-Chunking algorithm is introduced. It combines previous sequential training methods in order to create an Optimal parallel trainer.
The computational time in test phase of a Support Vector Machine is improved by reducing the number of support vectors. This work examines the two main approaches to reducing the number of support vectors: 1) Subset methods and 2) Reduced Set methods. The former aims at selecting a subset of support vectors \( \{ z_i \} \) from the set of support vectors \( \{ x_i \} \) that describe the SVM after training. The latter aims at finding a set of vectors, not necessarily support vectors, that approximate, in feature space, the decision boundary of the original SVM.

In this work, three new approaches which belong to the class of Subset methods are presented: 1) The Modified Exact Simplification method which allows a trade-off between accuracy and the number of support vectors by introducing a tolerance parameter into the algorithm presented in [9]; 2) the Simulated Annealing method which allows to specify a priori the number of support vectors \( N_z \) in the reduced set; and 3) the Explicit Mapping method which uses standard linear algebra techniques to reduce the number of support vectors in a high dimensional space.

In addition to the above three algorithms, two approaches based on the Reduced Set methods are introduced: 1) The Polynomial Kernel Reduction method which implements the application of the iterative algorithm of [36, 34] to polynomial kernels, and 2) the Standard Optimization Method which provides a general algorithm to find a reduced set of vectors, regardless of the kernel used.

This work also presents the implementation of a real-time face tracker to study the integration of Support Vector Machines into a visual real-time tracking architecture. Face-tracking was selected among other tracking applications because it represents a prototype visual tracking problem and it has a large number of applications, specially in the fields of
surveillance and human computer interaction. This application gives a test-bed to assess the feasibility of using SVMs in a real-time tracking architecture and to measure the impact of reducing the number of support vectors.

Chapter 2 presents the basic ideas behind Support Vector Machines and the necessary theoretical background for Chapters 3, 4 and 5. It also describes the face tracking problem and the tracking architecture used in this application. Chapter 3 addresses the SVM training problem. It presents parallel optimal and sub-optimal training alternatives in order to reduce the time involved. Chapter 4 explores techniques to reduce the number of support vectors of an already trained support vector machines improving classification time performance. Chapter 5 shows the application of support vector machines in a face tracking system. It also presents the combination of SVMs and Kalman filters in a tracking architecture. Finally, Chapter 6 presents the conclusions for this work.
Chapter 2

Background Theory

In this chapter the fundamental background of Support Vector Machines necessary for this work will be introduced. Then, an experimental system for the evaluation of real-time face-tracking applications will be described. This real-time face-tracking system will be used to study experimentally the integration of Support Vector Machines to real-time tracking systems, one of the primary contributions of this work.

2.1 Support Vector Machines

2.1.1 Support Vector Machines as Classifiers

Support Vector Machines (SVM) are learning systems that classify a given input in two classes, trained with a learning algorithm from optimization theory that implements a learning bias. One of its advantages over other classifiers is that it has a strong theoretical foundation on statistical learning theory. In this section, a brief description of SVM as a classifier will be given. For further details see [13, 7, 8].

To introduce the basic ideas behind SVM consider the problem of classifying two linearly
separable classes (x and o) as shown in Figure 2.1.

Figure 2.1: SVM decision boundary

Given a data set \( \{x_i, y_i\} \) where \( x_i \in \mathcal{X} \) is a vector in a given input space and \( y_i \in \{1, -1\} \) is the associated label for the classification. The objective is to find a hyperplane, \( \langle W, x \rangle - b \), where \( W \) and \( b \) are parameters to be determined by the learning (optimization) algorithm and \( \langle W, x \rangle = W^T x \), that is, \( \langle \cdot, \cdot \rangle \) denotes inner products. For this classifier, the decision rule is:

\[
f(x) = \text{sgn} \left( \langle W, x \rangle - b \right)
\]  

(2.1)

A geometric illustration of the linear classification problem is shown Figure 2.1. This figure shows that the problem is ill-posed; there can be infinitely many solutions to this problem. To make the problem well posed one could impose additional constraints. For example, we would like to choose the hyperplane that minimizes the generalization error, that is, the one that would do a good job classifying new samples. In [41] it was shown that such hyperplane is the one which maximizes the distance between itself and the closest points (vectors) from both classes. These vectors are called support vectors (the bold markers in Figure 2.1).

The mathematical foundation for SVMs was developed by Vapnik [41]. He proved that
the best hyperplane is the one that minimizes the norm of $W$ and can be found solving the following quadratic optimization problem:

$$\min_{W} \frac{1}{2} \langle W, W \rangle \text{ subject to } y_i (\langle W, x_i \rangle - b) \geq 1; \quad i = 1, \ldots, N$$  \hspace{1cm} (2.2)

where $N$ is the number of vectors in the data set.

Using the Lagrange multipliers technique [8], $W$ is found to be a linear combination of the training vectors $(x_i)$:

$$W = \sum_{i=1}^{N} \alpha_i y_i x_i,$$

where $\alpha_i$ is the Lagrange multiplier associated to the $i^{th}$ vector. Typically in the optimization process, the majority of the Lagrange multipliers are usually zero. The non-zero multipliers are associated to the support vectors. Therefore, $W$ can be written as:

$$W = \sum_{i=1}^{N_s} \alpha_i y_i x_i$$  \hspace{1cm} (2.3)

where $N_s$ is the number of support vectors $(x_i)$. Using (2.3) the decision rule (2.1) becomes

$$f(x) = \text{sgn} \left( \sum_{i=1}^{N_s} \alpha_i y_i \langle x_i, x \rangle - b \right) = \text{sgn} \left( \sum_{i=1}^{N_s} \alpha_i y_i \langle x_i, x \rangle - b \right)$$  \hspace{1cm} (2.4)

The real power of Support Vector Machines is that they can also be applied to nonlinear classification problems. To deal with non-linearly separable classes, a non-linear mapping $\Phi : \mathcal{X} \mapsto \mathcal{F}$ is used to map the input space ($\mathcal{X}$) into a higher dimensional space, called the feature space ($\mathcal{F}$), where the data becomes linearly separable (see Figure 2.2). Under this
mapping the decision rule (2.4) transforms into:

\[ f(x) = \text{sgn} \left( \sum_{i=1}^{N_s} \alpha_i y_i K(x_i, x) - b \right) \]

(2.5)

where

\[ K(x_i, x) = \langle \Phi(x_i), \Phi(x) \rangle = \Phi(x_i)^T \Phi(x), \]

is the kernel induced by the mapping \( \Phi \). This kernel encapsulates the inner product in the feature space. As a consequence, the classification problem resembles a linear problem and the computation involved in it is not affected by the dimension of the feature space.

\[ \Phi = \mathcal{X} \rightarrow \mathcal{F} \]

\( \Phi \) is the mapping function between input space and feature space

Figure 2.2: Example of non-linear mapping

In practical applications the success of the classification depends on the kernel chosen which becomes a design parameter. The most commonly used kernels are:

- Gaussian: \( K(a, b) = \exp\left(-\frac{||a - b||^2}{2\sigma^2}\right) \)

- Polynomial: \( K(a, b) = (a^T b + d)^p \)

where \( a, b \in \mathcal{X} \) are vectors in the input space and \( d \in \mathbb{R}, \sigma \in \mathbb{R} \) and \( p \in \mathbb{N} \) are parameters.
2.1.2 Training Support Vector Machines

Part of the design of a SVM involves a training stage where learning takes place. In the case of linearly separable problems training amounts to finding the bias $b$ and weight vector $W$ in (2.2). In practice training samples are seldom linearly separable. Therefore (2.2) is modified as follows:

$$\min_{W,b,\xi} \frac{1}{2} \langle W, W \rangle + C \sum_{i=1}^{N} \xi_{i}, \text{ subject to } y_i(\langle W, x_i \rangle - b) \geq 1 - \xi_{i}; \ i = 1, \ldots, N \quad (2.6)$$

where $\xi_{i}$ are slack variables that allow margin failure and $C$ is the limiting value parameter which trades off between wide margin and a small number of margin failures. The introduction of a limiting value imposes the following additional constraint on the Lagrange multipliers:

$$0 \leq \alpha_{i} \leq C, \ i = 1, \ldots, N \quad (2.7)$$

For the general non-linear classification the training problem becomes

$$\min_{\alpha,b,\xi} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_{i}y_{j} K(x_i, x_j) \alpha_i \alpha_j + C \sum_{i=1}^{N} \xi_{i} \quad \text{subject to } \ y_i(\sum_{j=1}^{N} \alpha_j K(x_i, x_j) - b) \geq 1 - \xi_{i}, \ i = 1, \ldots, N \quad (2.8)$$

$$0 \leq \alpha_{i} \leq C, \ i = 1, \ldots, N$$

The dual form of the minimization problem in (2.8) is preferred since it does not show the slack variables:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_{i}y_{j} K(x_i, x_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_{i} \quad \text{subject to } \ \sum_{i=1}^{N} y_i \alpha_i = 0, \ i = 1, \ldots, N$$

$$0 \leq \alpha_{i} \leq C, \ i = 1, \ldots, N \quad (2.9)$$
Solving (2.9) gives the support vectors and their associated Lagrange multipliers $\alpha_i$. Finally, the bias $b$ is found as:

$$b = \sum_{i=1}^{N_s} \alpha_i y_i K(x_i, x_k) - y_k$$  \hspace{1cm} (2.10)

where $x_k$ is any support vector such that $0 < \alpha_k < C$.

### 2.1.3 Output Evaluation

Once the support vectors $(x_i)$, their associated Lagrange multipliers $(\alpha_i)$ and the bias parameter $(b)$ have been found through the training process, the classification of a new sample is achieved by evaluating the decision function (2.5). However, in most practical applications, the score function is used instead:

$$u(x) = \sum_{i=1}^{N_s} \alpha_i y_i K(x_i, x) - b$$  \hspace{1cm} (2.11)

The absolute value of the score function provides a notion of distance to the decision boundary, which in turn can be used as a confidence measure. The bigger the distance the larger the confidence that the sample has been classified correctly.

From (2.11), it can be seen that the time complexity involved in classifying a new sample is directly proportional to the number of support vectors $(N_s)$. Other factors that influence the evaluation time of a new sample is the dimension of the input space ($\mathcal{X}$) and the kernel of choice.
2.2 A Real-time Face-tracking System

In this section a face-tracking system based on Support Vector Machines will be described. The objective is to investigate how SVMs can be used for real-time visual tracking and evaluate experimentally the impact of this work. Among all the possible applications, real-time face tracking was selected for two reasons: firstly, it represents a good prototype for visual tracking problems and secondly, it has a large number of practical applications, including surveillance and human computer interfacing.

A practical face tracking application should detect and track faces on complex backgrounds, be insensitive to head orientation, scale changes, illumination changes, partial occlusions and shadows [32]. The real-time requirement adds a constraint on the algorithms and methods to use. Since the objective of this work is to focus on the use of SVMs as classifiers in a tracking architecture, the problem was limited to tracking faces of people looking into the camera at a known fixed distance from the camera, without occlusion and with controlled illumination.

There is a vast literature on face tracking and face detection. A recent assessment of the state of the art in this topic can be found in [32] and [44]. The latter divides the face detection techniques in four groups: Knowledge-based methods, Feature invariant approaches, Template matching methods and Appearance-based methods. The feature invariant methods detect features such as skin color or face aspect ratio that tend to be similar [45, 35, 16, 38]. The appearance based methods deal with recognizing important characteristics of faces such as mouth and eyes and is based on Machine Learning and Training [39, 34, 20, 33].
Recent results reported in [14] show that face detection with a component-based approach is superior to other approaches. They have also been working with SVMs as classifiers. Inspired by these results a face tracking system that detects and tracks two facial features: eyes and lips, from a color video stream was designed and implemented. It uses a combination of Feature Invariant and Appearance Based methods. The architecture and implementation details are described next.

2.2.1 Tracking Architecture

The architecture consists of several tracking modules, one per each feature, and a Data Fusion Stage (see Figure 2.3). Each of the modules can be configured independently in order to take advantage of the characteristics of a specific feature. The data fusion algorithm exploits the relationships among different features to improve the overall (face) detection, tracking and reacquisition.

Figure 2.3: Feature Based Tracking Architecture
A visual tracking module is composed of pre-processing, classification and motion estimation stages, and has two operational modes: Detection and Tracking. In detection mode, the pre-processing and classifier combination is used to obtain an initial position of the feature of interest with high confidence. Then the module is switched to tracking mode, where a motion estimator is used to track the feature detected by the classifier. Both modes of operation give as a result a list of vectors \((f_i, s_i, x_i)\) containing a candidate identifier \((f_i)\), a score indicating the likelihood of the candidate to be the actual feature \((s_i)\) and its position \((x_i)\).

The data fusion stage combines temporal and spatial information from different modules to determine the position of the tracked features and to decide whether the detection mode or the tracking mode of each module should be used. The current and previous data from the list of vectors given by the modules is used to restrict the number of candidates to be classified, and to weight the candidates in order to have the most accurate classification and to choose the mode of operation (detection or tracking). Once the final position of the individual features has been determined, they are used to update the motion estimators from the different modules.

### 2.2.2 Face Tracking Implementation

The pre-processing stage consists of two steps. The first step uses a combination of skin color segmentation [35], density maps [19] and geometric filtering to extract face candidate regions from a frame. The second step extracts facial features candidates from each face candidate region. The facial feature candidates are obtained by using a combination of skin color information and the fact that facial features are of lower intensity than the rest of the
face [38]. More details of the pre-processing stage can be found in [24].

A Kalman Filter was designed for motion estimation and combined with a template matching technique. For each feature, two Kalman Filters were used, one per coordinate axis. For example, each eye has its own pair of filters. Its implementation is discussed with more detail in Chapter 5. The data fusion stage was designed on biometric heuristics based on the geometric relationships between eyes and lips.

The focus of this work on using Support Vector Machine classifiers as feature detectors. In this particular case, the features are eyes and lips. Therefore, two SVMs were designed: one to recognize eyes, and another to recognize lips.

2.2.3 SVMs in the Face Tracking System

As mentioned in the introduction, several issues have to be addressed in order to obtain the best performance out of a SVM. This work is centered on speeding up training and classification. Improving time performance in these two aspects would enable the use of SVM in real–time applications. To tackle these problems it is necessary to solve the tuning (parameter selection) and feature selection (input space selection) problem for the particular application at hand, in this case, eye and lip detection. It is also necessary to build a data set as input for the classifier.

Feature selection is a problem which deserves deeper analysis. Raw intensity images [2] as well as transformed images (e.g. using Haar wavelet transform [20]) have been used. In order to keep the problem simple, the raw intensity image was selected as the input space.
Once this decision was made, we proceeded to build the training and test sets.

Since the classifier needed to distinguish among facial features, we created a Facial Feature Data set (FFDS). It consisted of $10 \times 20$ pixel images grouped in five classes: eyes, lips, eyebrows, nostrils and hair. Two programs were written to perform this task. The first one followed the algorithm described in the pre-processing stage to obtain facial feature candidates. For each facial feature candidate, a 10x20 image was extracted and classified by a human operator. The second program was designed to guarantee the integrity of the data sets; the already classified features grouped by class were shown to an operator who verifies the classification. 13 subjects from different ethnicity went through this process, in which they were asked to move while looking at a video camera. Only small movements such as yaw, roll and pitch of $\pm 5$ degrees were allowed. The FFDS was then divided in a training set consisting of 15,308 images and a test set with 5,347 images.

The parameters for the lip SVM and eyes SVM classifiers still needed to be tuned. To choose the kernel ($K$) and the limiting value ($C$), experiments were performed with linear, polynomial and radial basis function (RBF) kernels using a smaller training set (5,000 samples) and varying $C$ from 0.01 to 100. A summary of the results is given in Table 2.1. The best performance was obtained using a polynomial kernel of second degree and $C = 0.01$.

Finally, training the two classifiers using the whole training set from the FFDS led to 2,415 support vectors in the case of eyes and 2,261 support vectors in the case of lips (see Table 2.2). The *Matlab support vector machines toolbox* [6] was used in all the experiments.
Table 2.1: Results of the tuning experiments

<table>
<thead>
<tr>
<th>SVM Kernel</th>
<th>C</th>
<th>#SV</th>
<th>%Tr. Set</th>
<th>%Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>1</td>
<td>692</td>
<td>100.0</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>775</td>
<td>99.4</td>
<td>97.4</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>694</td>
<td>100.0</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>753</td>
<td>100.0</td>
<td>97.1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>753</td>
<td>100.0</td>
<td>97.1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>754</td>
<td>100.0</td>
<td>97.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1417</td>
<td>91.3</td>
<td>92.3</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>2271</td>
<td>87.1</td>
<td>89.0</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1254</td>
<td>91.6</td>
<td>92.4</td>
</tr>
<tr>
<td>Lips</td>
<td>1</td>
<td>855</td>
<td>100.0</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>998</td>
<td>99.0</td>
<td>96.6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>854</td>
<td>100.0</td>
<td>96.1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>916</td>
<td>100.0</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>916</td>
<td>100.0</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>915</td>
<td>100.0</td>
<td>96.8</td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td>2344</td>
<td>85.3</td>
<td>83.8</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>2889</td>
<td>78.7</td>
<td>80.7</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1955</td>
<td>86.3</td>
<td>85.4</td>
</tr>
</tbody>
</table>

Table 2.2: SVM Classification Time

<table>
<thead>
<tr>
<th></th>
<th>#SV</th>
<th>Accuracy Training Set</th>
<th>Accuracy Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>2415</td>
<td>98.80%</td>
<td>91.10%</td>
</tr>
<tr>
<td>Lips</td>
<td>2261</td>
<td>98.65%</td>
<td>91.10%</td>
</tr>
</tbody>
</table>
Chapter 3

Parallel SVM Training

Training SVMs involves solving a quadratic optimization problem. Since the training sets are usually very large, the computational complexity involved in training has limited its application to practical problems, specially in the area of image processing. Several training algorithms that address this issue have been proposed. However, their application to large and dense training data sets with high dimensionality is still a challenge and may take several days of training depending on the hardware available.

Current advances in parallel computing are bringing to the user cost effective tools to attack complex problems such as this one. In this work, the use of parallel computing techniques to reduce the training time was investigated.

In this chapter the development of parallel algorithms for training SVMs will be presented. Of all the current approaches to training, the Sequential Minimal Optimization (SMO) algorithm [29] was selected for parallelization due to its good computational time performance and the use of a closed form solution for the Quadratic Programming (QP) optimization. The chapter is organized as follows: Section 3.1 gives an overview of the current training algorithms. Section 3.2 provides the description of the SMO algorithm. Section 3.3
analyzes previous work on parallelizing the SMO algorithm, while Section 3.4 introduces new approaches to the problem. Section 3.5 presents experimental results and Section 3.6 closes the chapter with a discussion.

3.1 Support Vector Machine Training

From (2.9) in Chapter 2, the training problem consists of finding the Lagrange multipliers $\alpha_i$ such that:

$$\min_{\alpha} \Psi(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j K(x_i, x_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i,$$  \hspace{1cm} (3.1)

subject to

$$0 \leq \alpha_i \leq C,$$ \hspace{1cm} (3.2)

$$\sum_{i=1}^{N} y_i \alpha_i = 0$$ \hspace{1cm} (3.3)

where $y_i \in \{1, -1\}$ is the classification label for the $i^{th}$ training sample and $C$ is the limiting value parameter which trades off between wide margin and a small number of margin failures.

Equation (3.1) describes a quadratic programming (QP) optimization problem. QP problems are NP-Complete with a computational complexity of order $O(N^k)$ where $N$ is the size of the training set and $k$ is a positive real number.

An optimal solution to the problem described by equation (3.1) is characterized by the necessary and sufficient Karush-Kuhn-Tucker (KKT) conditions given below:
\[ \alpha_i = 0 \iff y_i u_i \geq 1, \]
\[ 0 < \alpha_i < C \iff y_i u_i = 1, \]
\[ \alpha_i = C \iff y_i u_i \leq 1 \]

where \( u_i \) is the score function \((2.11)\) of the SVM for the \( i^{th} \) sample given by:

\[ u_i = u(x_i) = \sum_{j=1}^{N_s} \alpha_j y_j K(x_j, x_i) - b \]

In practice, standard QP solvers cannot be applied to SVM training due to the large size of the problem. Equation (3.1) involves a matrix that has \( N^2 \) elements, where \( N \) is the number of elements in the training set. Furthermore, the training elements usually belong to a high dimensional input space \( \mathcal{X} \). The most important approaches to solve this problem are described next.

In [40], Vapnik presented the chunking algorithm. It is based on the idea that the minimization problem (3.1) remains the same if the vectors associated to a zero Lagrange multiplier (non-support vectors) are removed. The algorithm divides the QP problem in a sequence of sub-problems of smaller size whose goal is to identify the non-support vectors and discard them. At each step, the algorithm solves a QP problem consisting of the support vectors from the previous step and a pre-defined number \( M \) of new elements until all the vectors from the training set have been optimized. Therefore, the size of the sub-problem increases in each step. This algorithm reduces the time complexity of the problem to almost \( O(N_s^2) \) where \( N_s \) is the final number of support vectors in the set. However, it still cannot handle large-scale training problems.
In 1997, Osuna et al. [27] presented an algorithm in which the QP problem was also divided in a sequence of sub-problems following Vapnik’s ideas. Osuna et al. demonstrated that if there is at least one sample that violates the KKT conditions in each of the sub-problems, then its optimization still reduces the overall objective function (3.1). In Osuna’s algorithm the size of the sub-problem to be solved is fixed. At each step, a number $M$ of samples are removed and $M$ samples that violate the KKT conditions are added to the previous sub-problem. This algorithm still requires a QP solver subroutine.

In 1999, Joachims [15] followed the same ideas presented by Osuna et al. and incorporated some heuristics for rapid convergence to create the $SVM^{light}$ algorithm. In this algorithm, the size of the sub-problem is not fixed but selected by heuristics. Computational improvements such as kernel caching and incremental updates of the gradient and termination criteria were also introduced.

Almost at the same time that Joachims presented $SVM^{light}$, Platt proposed the Sequential Minimal Optimization algorithm in [31]. Following Osuna’s ideas [27], Platt selects the size of the sub-problem as the smallest possible: 2. The main advantage of SMO is that the QP problem of each sub-problem can be solved analytically avoiding the use of numerical QP optimization.

### 3.1.1 SMO advantages

The Sequential Minimal Optimization (SMO) algorithm is one of the fastest algorithms for SVM training (see Table 3.1) and therefore a natural candidate for parallelization. SMO
can solve large QP problems without any matrix storage and without using a numerical QP optimization subroutine. These characteristics make the SMO algorithm ideal for large size problems. It is also less susceptible to numerical precision problems since it does not utilize any matrix computations. Furthermore, the algorithm is simple and can be easily implemented following the pseudo-code given by Platt [31].

3.2 Sequential Minimal Optimization

SMO iterates over all the training data, until the Karush-Kuhn-Tucker optimality conditions of the QP problem are satisfied. The problem is solved by analytically optimizing sub-sets of two Lagrange multipliers at a time. Figure 3.1 shows SMO’s main routine. The subroutine examineExample determines if the example being examined violates the KKT conditions. If it does, it uses a set of heuristics to obtain a second multiplier to optimize.
Main Routine
numChanged = 0
examineAll = 1
While (numChanged > 0) | examineAll)
{
    numChanged = 0
    if (examineAll)
    {
        loop I over all training examples
        numChanged += examineExample(I)
    }
    else
    {
        loop I over examples where alpha is not 0 & not C
        numChanged += examineExample(I)
    }
    If (examineAll == 1)
    examineAll = 0
    else if (numChanged == 0)
    examineAll = 1
}

Figure 3.1: SMO Main Routine

The two main contributions introduced by SMO are: 1) the analytic method for optimizing two Lagrange multipliers, and 2) the heuristics for selecting which multipliers to optimize.

3.2.1 Analytical optimization of two Lagrange multipliers

Given two Lagrange multipliers $\alpha_1$ and $\alpha_2$, they both have to satisfy the constraints (3.2) and (3.3). A geometric interpretation of these constraints is shown in Figure 3.2. The bound constraint (3.2) limits the multipliers to be inside the box while equation (3.3) forces...
them to be on a diagonal.

\[ \alpha_1 = 0 \quad \alpha_2 = C \]
\[ \alpha_1 = C \quad \alpha_2 = 0 \]

\[ y_1 \neq y_2 \Rightarrow \alpha_1 - \alpha_2 = \gamma \]
\[ y_1 = y_2 \Rightarrow \alpha_1 + \alpha_2 = \gamma \]

Figure 3.2: Training Constraints for 2 Lagrange multipliers

The algorithm uses the second derivative of the objective function along the diagonal line,
\[ \eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2), \]
to compute \( \alpha_2^{\text{new}} \):

\[ \alpha_2^{\text{new}} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{3.5} \]

where \( E_i = u_i - y_i \) is the error on the \( i^{th} \) training example.

Then the algorithm limits the value of \( \alpha_2^{\text{new}} \) to be between the ends of the diagonal, \( H \) and \( L \), as follows:

\[ \alpha_2^{\text{new,clipped}} = \begin{cases} H, & \text{if } \alpha_2^{\text{new}} \geq H \\ \alpha_2^{\text{new}}, & \text{if } L < \alpha_2^{\text{new}} < H \\ L, & \text{if } \alpha_2^{\text{new}} \leq L \end{cases} \tag{3.6} \]

If \( y_1 \neq y_2 \), the ends of the diagonal can be computed as described in (3.7), otherwise as
described in (3.8).

\[
L = \max(0, \alpha_2 - \alpha_1), \quad H = \min(C, C = \alpha_2 - \alpha_1)
\]  \hspace{1cm} (3.7)

\[
L = \max(0, \alpha_2 + \alpha_1 - C), \quad H = \min(C, \alpha_2 = \alpha_1)
\]  \hspace{1cm} (3.8)

Then, \(\alpha_1^{\text{new}}\) is computed as follows:

\[
\alpha_1^{\text{new}} = \alpha_1 + y_1 y_2 (\alpha_2 - \alpha_2^{\text{new,clipped}})
\]  \hspace{1cm} (3.9)

Once \(\alpha_1^{\text{new}}\) and \(\alpha_2^{\text{new}}\) have been calculated, they are used to compute the bias \(b\). If \(\alpha_1^{\text{new}}\) is not at the bound, \(b\) is computed using \((3.10)\). If \(\alpha_2^{\text{new}}\) is not at the bound, \(b\) is computed according to \((3.11)\). When both \(\alpha_1^{\text{new}}\) and \(\alpha_2^{\text{new}}\) are at bound then \((3.12)\) is used.

\[
b_{\text{new}} = b_1 = E_1 + y_1 (\alpha_1^{\text{new}} - \alpha_1) K(x_1, x_1) + y_2 (\alpha_2^{\text{new,clipped}} - \alpha_2) K(x_1, x_2) + b
\]  \hspace{1cm} (3.10)

\[
b_{\text{new}} = b_2 = E_2 + y_1 (\alpha_1^{\text{new}} - \alpha_1) K(x_1, x_2) + y_2 (\alpha_2^{\text{new,clipped}} - \alpha_2) K(x_2, x_2) + b
\]  \hspace{1cm} (3.11)

\[
b_{\text{new}} = \frac{(b_1 + b_2)}{2}
\]  \hspace{1cm} (3.12)

3.2.2 Selecting two multipliers

Once SMO has found one Lagrange multiplier \((\alpha_1)\) that violates the KKT conditions, it uses the following heuristics to choose the second multiplier to maximize the size of the step taken during the joint optimization. First, it tries to select a non-bound multiplier \(\alpha_2\) which maximizes \( | E_1 - E_2 | \). If the previous heuristic does not lead to positive progress, then SMO starts iterating through the non-bound multipliers until it finds one that leads to positive progress. If the previous heuristic fails, it iterates through all the lagrange multipliers until it finds one that leads to positive progress. If all the heuristics fail, then \(\alpha_1\) is
skipped. In normal conditions, only the first two heuristics are used.

### 3.2.3 Unbiased SMO

Platt also presented an unbiased SMO algorithm. In this case, the bias \((b)\) is fixed at zero. As a result, only one Lagrange multiplier needs to be updated at a time:

\[
\alpha^\text{new}_1 = \alpha_1 + \frac{y_1 E_1}{K(x_1, x_1)}
\]  

(3.13)

After \(\alpha^\text{new}_1\) is computed, it is clipped between 0 and \(C\). The Unbiased SMO algorithm would lead to a sub-optimal solution since the margin achieved will not necessarily be maximum. Details on implementation can be found in [31].

### 3.3 Previous Work on SMO Parallelization

Previous work on SMO parallelization can be divided in optimal and sub-optimal approaches. In the present context, a sub-optimal approach gives a solution which is a feasible point of the QP minimization problem, but it is not the global minimum. Sub-optimal approaches have been referred as non-optimal solutions in previous work. As opposed to sub-optimal approaches, an optimal one gives the global minimum of the QP training problem as the result of the training process.

In 2001, Etin et al. [10] attempted to parallelize the SMO algorithm. They primarily focused on the biased version of SMO and proposed optimal and sub-optimal approaches. Etin's optimal algorithm divided the training problem into sub-problems to be optimized
by each processor. The results from each sub-problem will then be used as a starting point for a final optimization in one processor. The sub-optimal approach replaces the final optimization step by a heuristic rule to combine the results from each sub-problem.

In 2002, Woitaszek [43] followed the ideas presented in [10] and implemented Etin’s sub-optimal approach. He also proposed an interleaved parallelization approach in which every processor optimized a subset of the training problem and then exchanged some elements of its subset with another processor. The details of Etin’s and Woitaszek’s approaches are presented next.

### 3.3.1 Interleaved Parallelization (IP)

This approach was presented in [43]. This algorithm divides the QP training problem in a series of steps. Each step involves solving $n$ QP sub-problems, where $n$ is the number of processors. The training set, containing $N$ samples, is logically subdivided into $2n$ segments. The division of the training set into segments is done as shown in Figure 3.3. The training set is separated into positive and negative samples, which in turn are divided in the number of segments needed ($2n$ for the IP approach). The final segments are composed of one sub-set of positive samples and one sub-set of negative samples. Therefore, each segment would approximately have the same ratio of positive vs. negative samples.

Each processor optimizes a sub-problem containing two of the segments in every step. After one step is finished, one of the segments is exchanged with a segment from another processor. This process leads to an all-to-all optimization pattern. Every training sample
Figure 3.3: Division of the training set in \( n \) segments has the opportunity to optimize with all the other vectors. Figure 3.4 shows a three processor interleaved parallelization.

Figure 3.4: Three processor Interleaved Parallelization

Initially, all processors have the same bias \((b)\) and error cache values \((E_i)\). At the end of
each step, the appropriate error cache values are sent with each segment to the corresponding processor, and $b$ is computed as the arithmetic mean of all the biases in the different processors. This new value of the bias might cause the previously verified samples to violate the KKT conditions, and therefore, one more step is taken for verification. If KKT violators are detected the optimization process starts from the beginning.

Even though this method of parallelization was aimed to obtain an optimal solution, the conclusions presented in [43] clearly indicate that it increases complexity and introduces communication overheads. The global bias parameter also represents a problem since it induces instability and prevents it from converging.

### 3.3.2 Biased Sub-Optimal Blocked Parallelization (BSOBP)

This approach was presented in [43] and [10]. The $N$ training samples are divided into $n$ segments as shown in figure 3.3. Each processor performs a local SMO over one segment obtaining a group of local support vectors and a local bias. The final solution is achieved by gathering all the local support vectors into the global set of support vectors. To obtain the global bias, each processor multiplies its local bias value by the number of support vectors on that processor. The result is added across all processors and then divided by the total number of support vectors. Figure 3.5 shows how the approach works with four processors.

This method intends to reduce the computation complexity of the previous one by compromising its accuracy and generalization. Some samples may violate the KKT conditions due to the change in bias from the local value to the global average. These violations are
ignored, decreasing the algorithm's accuracy but substantially increasing its time performance. This approach was evaluated on 4 processors in [43] and on 2 processors in [10]. Both of their results agreed on the fact that this method is useful if the overall accuracy can be compromised. The effect of this approach on the number of support vectors is not mentioned.

3.3.3 Biased Optimal Blocked Parallelization (BOBP)

This approach was presented in [10]. Like in the BSOBP method (Section 3.3.2), the $N$ training samples are divided into $n$ segments, one per processor. Each processor performs a local SMO over its segment obtaining a group of local support vectors and a local bias. After the weighted average bias is computed, the sub-optimal solution computed so far is utilized as an initial point for a final optimization step in one processor. The idea behind this method is that the sub-optimal solution will be closer to the optimal one, and therefore, the final optimization will require less time. Figure 3.6 shows the BOBP approach with
According to the results presenting in [10], computing the error cache function for all the samples to perform the final optimization step has a computational complexity of $O(N^2)$. This computation in big data-sets may result in a considerable delay before starting the final SMO optimization, introducing parallelization overhead. Without this overhead, Etin’s results showed an improvement in the time performance of the algorithm, however it was much less than the sub-optimal method improvement.

### 3.4 New Approaches

In this section, three new algorithms are proposed to improve the performance of SVM parallel training. The first one introduces the use of the unbiased version of SMO to obtain sub-optimal solutions. The second one improves the sub-optimal algorithm presented in Subsection 3.3.2. The third alternative explores the combination of the Chunking [40] and
SMO algorithms.

### 3.4.1 Unbiased Sub-Optimal Blocked Parallelization (USOBP)

In previous efforts, both [10] and [43] have only considered biased versions of SMO. Unbiased SMO has not been explored mainly because it leads to a sub-optimal solution. However, the BSOBP (Section 3.3.2) approach sacrifices accuracy and optimality in order to achieve a better time performance. In cases where a sub-optimal solution is acceptable, an unbiased SMO could be used.

For this approach, the $N$ training samples are divided into $n$ segments as shown in Figure 3.3. Each processor performs a local unbiased SMO over one segment obtaining a group of local support vectors. The final solution is achieved by gathering all the local support vectors into the global set of support vectors.

Since the value of $b$ is fixed at zero, this approach should present a better time performance than the BSOBP approach. First of all, it does not require to calculate a global bias value. Secondly, it can modify one Lagrange multiplier at a time, saving the time needed to select the second multiplier for the optimization process.

According to the experiment results (see Section 3.5), the Unbiased Sub-Optimal Blocked Parallelization performs better than BSOBP when the number of vectors processed by each processor is larger. This means that this approach is more suitable for large training data sets and for a small number of processors.
3.4.2 Modified Biased Sub-Optimal Blocked Parallelization (MBSOBP)

This method modifies the computation of the bias \( b \) in the BSOBP approach (Subsection 3.3.2). In [10, 43], a heuristic rule was introduced to calculate the final global bias value of the sub-optimal SVM. This heuristic may work when the gaussian or lineal kernels are used, but it performs poorly with a polynomial kernel.

Let us assume that the local results from two processors are as follows:

\[
\begin{align*}
    u_1 &= \sum_{i=1}^{N_{s1}} \alpha_{1i} y_{1i} K(x_{1i}, x) \\ 
    u_2 &= \sum_{i=1}^{N_{s2}} \alpha_{2i} y_{2i} K(x_{2i}, x)
\end{align*}
\]  

Where \( N_{s1} \) and \( N_{s2} \) are the local number of support vectors in processor 1 and 2 respectively. According to the BSOBP approach, the final bias should be computed as:

\[
b_{final} = \frac{b_1 N_{s1} + b_2 N_{s2}}{N_{s1} + N_{s2}}
\]  

Therefore the global solution has the form:

\[
u_{final} = u_1 + u_2 - b_{final}
\]

According to (3.16), \( b_{final} < b_1 + b_2 \) and \( b_{final} \in [b_1, b_2] \) assuming \( b_1 \leq b_2 \). Since the local solutions, (3.14) and (3.15), are independent, ideally, they both will show a classification output close to \( \pm 1 \) (depending if it is positive or negative). Using (3.16) as the \( b_{final} \) value would favor a positive result as the output. Furthermore, depending on local bias values and on the number of processors being used, the final output could always be forced to take
a positive value. To avoid these complications, MBSOBP computes the final bias value as the sum of all the local biases:

\[ b_{final} = \sum_{i=1}^{n} b_i \]  

(3.18)

The modification presented in this approach is expected to improve the results of a biased sub-optimal algorithm when used with a large number of processors and with other kernels, specially polynomials.

Accuracy results presented in Table 3.5, Section 3.5 show that the modified algorithm (MB-SOBP) outperforms the original one (BSOBP), while their time performances are similar.

### 3.4.3 Chunking - SMO (ChSMO)

The previous new approaches (USOBP, MBSOBP) are sub-optimal solutions to the SVM training problem. However, for some applications, optimality is a fundamental constraint. Therefore an optimal Chunking SMO approach is proposed. The basic idea behind this approach is to reduce the number of samples in the final optimization according to the decomposition ideas presented by Vapnik [40]. The Training problem is divided in stages. In the initial stage, the training set \((N)\) is divided in \(n\) segments according to the division process shown in Figure 3.3. For this approach, the number of processors \((n)\) needs to be a power of 2. Each processor performs a biased SMO. In the second stage, \(n/2\) processors participate by performing SMO over a new subset composed of the resulting support vectors from itself and from another processor from the previous stage as showed in Figure 3.7. Only the resulting support vectors are transferred to the following stages, until finally the last optimization is performed in only one processor.
Results presented in Section 3.5 show that the speed-up of this algorithm is much lower than the one achieved by sub-optimal algorithms. However, the final solution is optimal in correspondence with the SVM theory. The results also show that the time complexity involved in the algorithm depends on the number of support vectors ($N_s$) in the optimization; a value which cannot be known a priori.
3.5 Results

The code implemented by Woitaszek in [43] was modified to evaluate the three SMO parallel algorithms proposed in this chapter (USOBP, MBSOBP, ChSMO) as well as two previous efforts: BSOBP and BOBP. IP was not implemented because the conclusions presented in [43] clearly stated that the approach did not converge. The algorithms were written in C++ and run on a Beowulf Cluster at the Laboratory for Applied Computing [21] at the Rochester Institute of Technology. Each of the cluster nodes has two Pentium III 1.4 GHz with 512MB of RAM connected with Gigabit Ethernet. The algorithms were benchmarked using 1 to 32 processors.

The Adult UCI database (AUD) [3] was used as a training set and trained with a RBF kernel. The task of the classifier was to predict whether that household has an income greater than USD 50,000. In order to compare the results with those obtained by [43, 10] and [31], the database was modified as suggested in [31]. The database consists of 14 attributes; 8 are categorical and 6 are continuous. The continuous attributes were discretized yielding a total of 123 binary attributes. A variance value of 10 and the limiting value (3.2) of $C = 0.05$ were chosen for the RBF kernel. Using the standard implementation of SMO, the training led to 8,419 support vectors, achieving 86.60% of accuracy in 145,212.51 seconds.

For the sub-optimal approaches, the lip classification problem using the facial feature test set (FFTS, Subsection 2.2.3) was also used to assess their performance in an image processing application using a second degree polynomial kernel with $C = 0.01$. 2,261 support vectors with an accuracy of 98.60% was obtained by using the SMO implementation in the MATLAB support vector machine toolbox [6].
3.5.1 Results on Sub-Optimal Solutions (BSOBP, MBSOBP and USOBP)

The main advantage of the sub-optimal solutions is the impressive speed-up achieved when compared to the sequential version. Figure 3.8 and Table 3.2 show the speed-up for the Adult UCI database as compared to the training time of the standard SMO. It is important to note that this comparison is being made between two different algorithms. It cannot be considered as a true speed-up value since a sub-optimal and therefore different solution for the SVM training is obtained.

<table>
<thead>
<tr>
<th>n</th>
<th>BSOBP</th>
<th>MBSOBP</th>
<th>USOBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>5.93</td>
</tr>
<tr>
<td>2</td>
<td>3.93</td>
<td>3.93</td>
<td>17.33</td>
</tr>
<tr>
<td>4</td>
<td>18.10</td>
<td>18.10</td>
<td>71.02</td>
</tr>
<tr>
<td>8</td>
<td>102.93</td>
<td>102.93</td>
<td>223.39</td>
</tr>
<tr>
<td>16</td>
<td>392.46</td>
<td>392.46</td>
<td>660.19</td>
</tr>
<tr>
<td>32</td>
<td>1504.20</td>
<td>1504.20</td>
<td>1639.03</td>
</tr>
</tbody>
</table>

Table 3.2: Speed-up for the Adult UCI database using Sub-Optimal Approaches

Figure 3.8: Sub-optimal approaches: Speed-up for the Adult UCI database
Although in these results, the speed-up of the unbiased method is greater than the biased ones, the progression in speed-up indicates that with a higher number of processors the MBSOBP and BSOBP approaches would have better performance. The speed-up for the BSOBP and MBSOBP approaches scale as $O(n^{2.1})$, while the speed-up for the USOBP approach scales as $O(n^{1.6})$.

Incrementing the number of processors leads to reducing the computation time per processor and incrementing the time for communication. Therefore, the communication overheads will slow down the overall performance of these algorithms when $n$ increases. This effect starts to be noticeable in the three algorithms when using 32 processors.

<table>
<thead>
<tr>
<th>$n$</th>
<th>BSOBP</th>
<th>MBSOBP</th>
<th>USOBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>366029534.62</td>
<td>366029534.62</td>
<td>49489000.00</td>
</tr>
<tr>
<td>2*</td>
<td>38497505.60</td>
<td>38497505.60</td>
<td>8883547.98</td>
</tr>
<tr>
<td>4+</td>
<td>4046551.81</td>
<td>4046551.81</td>
<td>1282954.43</td>
</tr>
<tr>
<td>8</td>
<td>431086.68</td>
<td>431086.68</td>
<td>352233.74</td>
</tr>
<tr>
<td>16</td>
<td>48044.11</td>
<td>48044.11</td>
<td>65135.74</td>
</tr>
<tr>
<td>32</td>
<td>4613.87</td>
<td>4613.87</td>
<td>7346.80</td>
</tr>
</tbody>
</table>

Table 3.3: Training time in seconds using sub-optimal approaches for the facial feature data set. * Predicted for all approaches, + predicted for BSOBP and MBSOBP

Table 3.3 shows the training time in seconds for the FFDS. In this case, the training time for the biased approaches scale as $O(n^{-3.25})$ while the unbiased one scales $O(n^{-2.48})$. These results agree with ones from the Adult UCI database (see Table 3.2). The USOBP approach has a better time performance with less number of processors, while the biased methods are better with a higher $n$. This also suggests that the USOBP would have a better performance with larger data sets.

These results indicate that there is a relationship between the speed-up scaling factor of
the sub-optimal parallel algorithms with respect to the number of processors $n$: $O(n^{k_1})$, and the training time scaling factor of the sequential SMO with respect to the size of the training set $N$: $O(N^{k_2})$. According to the results, $k_1 \sim k_2$. This fact can be deduced from the way the sub-optimal algorithms are structured. These algorithms divide a QP problem of size $N$ in sub-problems of size $N/n$. Assuming SMO scales as $O(N^k)$ with respect to the size of the training set, the speed-up scales as: $O\left(\frac{N^k}{(N/n)^k}\right) = O(n^k)$. For the Adult database, the sequential biased SMO algorithm training time scaled as $O(N^2)$ depending on the size of the training set [31, 10]. Table 3.2 shows that the biased parallel algorithm scales as $O(n^{2.1})$.

<table>
<thead>
<tr>
<th></th>
<th>AUD</th>
<th>FFDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BSOBP</td>
<td>MBSOBP</td>
</tr>
<tr>
<td>1</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>3.16%</td>
<td>3.16%</td>
</tr>
<tr>
<td>4</td>
<td>7.47%</td>
<td>7.47%</td>
</tr>
<tr>
<td>8</td>
<td>12.33%</td>
<td>12.33%</td>
</tr>
<tr>
<td>16</td>
<td>18.57%</td>
<td>18.57%</td>
</tr>
<tr>
<td>32</td>
<td>25.56%</td>
<td>25.56%</td>
</tr>
</tbody>
</table>

Table 3.4: Increment in the number of support vectors for the sub-optimal approaches

One downside of these approaches is the increment in the number of support vectors. Figure 3.9 and Table 3.4 show this increment for Adult UCI Database and the Facial Feature Data Set. In the first database, the increment when using 32 processors is of approximately 25%. For the Facial Feature Data Set, the increment obtained goes up to 120%. This increment in the number of support vectors was not reported previously. It becomes an important issue when the goal is to use SVMs as a real-time classifier.

Table 3.5 and Figure 3.10 show the percentage of accuracy lost in comparison with the optimal solution. The results show a clear tendency to increase as the number of processors
Figure 3.9: Increment in the number of support vectors for sub-optimal approaches increases. The increase rate seems to be problem dependent, and it is bigger in the case of the polynomial kernel. The loss in accuracy of the BSOBP method for the lip classification data set was not illustrated in Figure 3.9 since the results show a loss over 60%. This is due to the way BSOBP calculates the final bias value.

<table>
<thead>
<tr>
<th>AUD</th>
<th>FFDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>BSOBP</td>
</tr>
<tr>
<td>1</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.35%</td>
</tr>
<tr>
<td>4</td>
<td>1.05%</td>
</tr>
<tr>
<td>8</td>
<td>1.68%</td>
</tr>
<tr>
<td>16</td>
<td>2.36%</td>
</tr>
<tr>
<td>32</td>
<td>3.09%</td>
</tr>
</tbody>
</table>

Table 3.5: Loss in Accuracy for sub-optimal approaches
Figure 3.10: Loss in accuracy for the sub-optimal approaches

3.5.2 Results on Optimal Solutions (BOBP and ChSMO)

The BOBP approach did not present any improvement in speed-up. The problem described in [10] related to the computation of the error cache function \( E_i \) results in a negative speed-up as shown in Figure 3.11 and in Table 3.6.

<table>
<thead>
<tr>
<th>n</th>
<th>ChSMO</th>
<th>BOBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.60</td>
<td>0.82</td>
</tr>
<tr>
<td>4</td>
<td>1.90</td>
<td>0.81</td>
</tr>
<tr>
<td>8</td>
<td>2.52</td>
<td>0.74</td>
</tr>
<tr>
<td>16</td>
<td>2.62</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 3.6: Speed Up in the Adult UCI Database for the optimal approaches

The proposed ChSMO algorithm presents an optimal solution with a better time performance than BOBP. It has a positive speed-up, although it is considerably less than the ones achieved by the sub-optimal approaches. It is important to note that ChSMO inherits its properties from the normal chunking algorithm. It performs better when the ratio (number
of support vectors) / (total number of samples) is smaller (< 10%). The modified adult training set used contained 22,697 samples, and from this number, almost 40% are support vectors (8,419). Therefore, it was expected that its time performance saturated quickly as shown in Figure 3.11.

3.6 Summary and Discussion

In this chapter, the parallelization of Platt’s Sequential Minimal Optimization algorithm was investigated as an answer to complex Support Vector Machine training. The parallel training methods presented were divided in two groups: Optimal and sub-optimal methods. The following remarks summarize this chapter:

1. The sub-optimal approaches showed great speed-up properties, however at the same
time, they increase the number of support vectors and lose accuracy and generalization properties. If training time is the main constraint in the problem to solve, then they are definitely the best algorithms to use. However, they are not suitable when the SVM classifier is intended for a real-time application. The loss in accuracy as well as the increment in the number of support vectors seem to be dependent on the particular problem and on the way the training set is sorted. It is also important to note that the sub-optimal solutions present a loss in generalization which is indirectly represented in the loss in accuracy. However, a better metric is needed to measure the generalization properties of the sub-optimal SVM solution.

2. A new sub-optimal parallel training algorithm has been presented. The Unbiased sub-optimal Blocked Parallelization performs better than previous biased algorithms (BSOBP) when dealing with small number of processors or with big training data sets.

3. An improvement in the computation of the bias parameter was introduced to a previous sub-optimal parallel training algorithm. Accuracy results presented in Table 3.5 suggest that the modified algorithm (MBSOBP) outperforms the original one (BSOBP).

4. In the case of the Optimal parallel approaches, a combination of SMO with the chunking algorithm was proposed. Results showed a poor speed-up of approximately 3 when 32 processors were used (three orders of magnitude less than the sub-optimal approaches). The ChSMO approach proved to be sensitive to the number of support vectors vs. number of total samples ratio.

5. Optimal Parallel training is still a challenge. Previous parallelization approaches and the new ones presented in this chapter addressed the parallel training problem at
a functional level. A deeper mathematical analysis of the problem is necessary in order to develop a SVM parallel trainer which should not necessarily be based on any current sequential one.
Chapter 4

Support Vector Reduction

Support Vector Machines are slower than neural networks and other classifiers during the test phase. This constitutes a major obstacle for the use of SVMs in applications that require real-time performance. The time complexity involved in the test phase is proportional to the number of Support Vectors ($N_s$). In turn, $N_s$ is also proportional to the number of samples used in the training phase ($N$). Furthermore, the expected value of the generalization rate is bounded by $\mathcal{E}_N\{\text{error}\} \leq \mathcal{E}_N\{N_s\}/N$, where the expectation is taken over all training sets of size $N$. Therefore, reducing $N$ to improve time performance is not acceptable, since having a bigger training data set will lead to a better classification performance.

In this work the aim is to improve the time performance of SVMs by reducing the number of support vectors of a SVM after training. There are two main approaches to the reduction of the number of support vectors: 1) Subset methods and 2) Reduced Set methods. The former aims at selecting a subset of support vectors $\{z_i\}$ from the set of support vectors $\{x_i\}$ that describe the SVM after training. The latter aims at finding a set of vectors, not necessarily support vectors, that approximate, in feature space, the decision boundary of the original SVM.
In this chapter three new approaches which belong to the class of Subset methods are presented: 1) The Modified Exact Simplification method which allows a trade-off between accuracy and the number of support vectors by introducing a tolerance parameter into the algorithm presented in [9]; 2) the Simulated Annealing method which allows to specify a priori the number of support vectors $N_z$ in the reduced set; and 3) the Explicit Mapping method which uses standard linear algebra techniques to reduce the number of support vectors in a high dimensional (feature) space.

In addition to the above three algorithms two approaches based on the Reduced Set methods are introduced: 1) The Polynomial Kernel Reduction method which applies the iterative algorithm of [36, 34] to polynomial kernels, and 2) the Standard Optimization Method which provides a general algorithm to find a reduced set of vectors, regardless of the kernel used.

The chapter is organized as follows. First, the problem of reducing the number of support vectors is defined. Current approaches and previous work on the reduction of support vectors are summarized next. Then, the new proposed methods are presented and evaluated experimentally. Finally the results are compared and discussed.

### 4.1 The Reduced Support Vector Problem

The evaluation of a new sample by a SVM classifier is given by the score function (2.11) as written below:

$$u(x) = \sum_{i=1}^{N_z} w_i K(x, x_i) - b$$

(4.1)
where $N_s$ is the number of support vectors $x_i \in \mathcal{X}$, $w_i \in \mathbb{R}$ are the weights associated to each vector, $b \in \mathbb{R}$ is the bias parameter and $K(\cdot, \cdot)$ is the kernel chosen.

The reduced support vector problem is to find a new score function:

$$v(x) = \sum_{i=1}^{N_z} \beta_i K(x, z_i) - \zeta$$  \hspace{1cm} (4.2)

where $N_z$ is the number of vectors $z_i \in \mathcal{X}$ in the reduced set, $\beta_i \in \mathbb{R}$ and $\zeta \in \mathbb{R}$, such that $N_z \ll N_s$ and the degradation in performance of the classifier in terms of accuracy and generalization is minimized.

This is an approximation problem that can be posed as a minimization problem as follows: The decision boundary of (4.1) is defined implicitly by:

$$\Psi = \sum_{i=1}^{N_s} w_i \Phi(x_i)$$  \hspace{1cm} (4.3)

where $\Psi \in \mathcal{F}$ is characterized by the support vectors $x_i \in \mathcal{X}$, $w_i \in \mathbb{R}$ and $\Phi(x) \in \mathcal{F}$ is the non-linear mapping implicitly defined by the kernel of choice.

Similarly, the decision boundary described by (4.2) can be written as:

$$\Psi' = \sum_{i=1}^{N_z} \beta_i \Phi(z_i)$$  \hspace{1cm} (4.4)

with $N_z \ll N_s$, $\beta_i \in \mathbb{R}$, and the reduced set of vectors $z_i \in \mathcal{X}$. 
Then, the reduced support vector problem can be posed as:

\[ \min_{\Psi'} \| \Psi - \Psi' \|^2 \]  

(4.5)

where \( \| \cdot \| \) denotes the Euclidian Norm.

### 4.2 Current Approaches

There are two main approaches to solving the reduced support vector problem:

1. **Subset Methods.**

2. **Reduced Set Methods.**

The Subset methods select a reduced set of vectors \( \{ z_i \} \) from the current set of support vectors \( \{ x_i \} \), i.e., \( \{ z_i \} \subset \{ x_i \} \). The associated weights \( \beta_i \) to each vector \( z_i \) and the bias \( \zeta \) are chosen to minimize (4.5). One way to do this was presented in [9] where linearly dependent support vectors in the feature space are discarded without losing generalization nor accuracy in the trained support vector machine. The details of this method are presented in Section 4.2.1.

The main idea behind the Reduced Set methods is to find \( z_i \in \mathcal{X} \), not necessarily support vectors, which define an approximation to the decision boundary (4.4) that minimizes (4.5). In [36, 5], Burges et al. developed this idea and proposed an iterative algorithm to find the reduced set vectors. The details are presented in Section 4.2.2.
The rest of the section presents the previous work on the reduced support vector problem. Two methods are presented: 1) The Exact Simplification [9] which is a Subset method; and 2) the Gaussian Kernel Reduction which belongs to the second type of approaches.

4.2.1 Exact Simplification

This method was proposed in [9] and belongs to the Subset methods. It reduces the number of support vectors by discarding the ones which are linear dependent in feature space. The details of this method are presented next.

Assuming that the support vector $x_k$ is linearly dependent on the other support vectors in the feature space:

$$K(x, x_k) = \sum_{i=1, i \neq k}^{N_s} c_i K(x, x_i)$$

(4.6)

where $c_i$ are scalar constants. Then (4.1) can be written as:

$$u(x) = \sum_{i=1, i \neq k}^{N_s} w_i K(x, x_i) + w_k \sum_{i=1, i \neq k}^{N_s} c_i K(x, x_i) - b$$

$$u(x) = \sum_{i=1, i \neq k}^{N_s} w_i (1 + d_i) K(x, x_i) - b$$

$$u(x) = \sum_{i=1, i \neq k}^{N_s} \hat{w}_i K(x, x_i) - b$$

(4.7)

where $d_i = (w_k c_i)/w_i$ and $\hat{w}_i = w_i (1 + d_i)$. This approach suggests that all the linearly dependent support vectors in the feature space can be eliminated from the score function without losing the accuracy or generalization of the original Support Vector Machine.
Figure 4.1: Exact Simplification Algorithm

Figure 4.1 shows the flow diagram for the Exact Simplification method. The algorithm maintains a set of linearly independent support vectors \( I \) in the feature space. Given the set of support vectors arranged in a random order \( (SV) \), the algorithm initializes \( I \) with the first vector \( SV(1) \). Then the algorithm starts iterating through all the remaining support vectors (e.g. from 2 to \( N_s \)). At each iteration \( t \), the algorithm checks if \( SV(t) \) is linearly independent from the vectors in \( I \). If it is, then \( SV(t) \) is added to \( I \), otherwise, the weights associated to the vectors in \( I \) are changed according to (4.7).

In order to perform the linear independence check, the size of the matrix \( M \) is compared
to its rank. Given the extended set \( ISV = [ISV(t)] \) containing \( N_i + 1 \) vectors \( s_i \), \( M \) is a \((N_i + 1) \times (N_i + 1)\) matrix with elements: \( M_{i,j} = K(r_i, s_j) \) where \( r_i \in \mathcal{X} \) are random vectors in the input space. If the rank of \( M \) and its size are equal, then \( SV(t) \) is independent.

The advantage of this method is that the properties in terms of accuracy and generalization of the SVM classifier are maintained. The approximated decision boundary \( \Psi' (4.4) \) is equal to the original boundary \( \Psi (4.3) \).

According to the results presented in [9], the amount of reduction achievable by this method is both problem and kernel dependent. Therefore, there is no control over the number of vectors in the reduced set, which means that the number of reduced vectors \( N_z \) cannot be fixed a priori. Moreover, there may be no reduction if there is no linear dependence among the support vectors. For example, in the case of the lip classification for the facial feature database collected in [24], there was no reduction at all since the support vectors proved to be linearly independent in the feature space. These limitations are addressed in Section 4.3.

### 4.2.2 Gaussian Kernel Reduction Method

The idea of using a reduced set of vectors \( \{z_i\} \) to create a decision boundary \( \Psi' \) to minimize (4.5) was first introduced by Burges in 1996 [5]. These vectors are not part of the training samples, and therefore they are not support vectors. Burges proposed an analytical method to compute them in the case of homogenous polynomial kernels and an iterative unconstrained minimization method for other cases. Burges' idea was improved in [36] where Scholkpf et al. presented an iterative method to find the reduced set vectors emphasizing the simplification of gaussian kernels. The improved method was used in [34] for a
hierarchical face recognition system using SVMs. The details of this method are presented below.

Scholkpf et al. proposed an iterative solution for (4.5):

$$\min_{\Psi'} \|\Psi - \Psi'\|^2$$

where $\| \| \|$ denotes the Euclidian Norm.

First, they consider the case approximating $\Psi$ with one vector $z$. Then (4.5) will transform into:

$$\min_z \left\| \frac{\Psi \cdot \Phi(z)}{\Phi(z)} \Phi(z) - \Psi \right\|^2 = \min_z \|\Psi\|^2 - \frac{(\Psi \cdot \Phi(z))^2}{(\Phi(z) \cdot \Phi(z))} \quad (4.8)$$

which is equivalent to:

$$\max_z \frac{(\Psi \cdot \Phi(z))^2}{(\Phi(z) \cdot \Phi(z))} \quad (4.9)$$

Once $z$ is found through (4.9), the associated weight $\beta$ is computed by:

$$\beta = \frac{(\Psi \cdot \Phi(z))}{(\Phi(z) \cdot \Phi(z))} \quad (4.10)$$

In order to calculate higher order reduced set vectors $z_m$, $m > 1$, the algorithm in Figure 4.2 is followed. $\Psi_m$ is introduced as the decision boundary to minimize in every iteration of the algorithm. For the first one, $\Psi_m$ is initialized as the original decision boundary
Figure 4.2: Gaussian Kernel Reduction Method: Iterative Algorithm

The procedure will end when $\| \Psi - \Psi' \|^2$ is less than a pre-defined tolerance $\rho$, or $m = N_z$. 

where $m$ is the iteration number, $N_x = N_z + m - 1$, $(\omega_1, \ldots, \omega_{N_x}) = (\omega_1, \ldots, \omega_{N_z}, -\beta_1, \ldots, -\beta_{m-1})$, and $(\chi_1, \ldots, \chi_{N_x}) = (x_1, \ldots, x_{N_z}, z_1, \ldots, z_{m-1})$. 

In the next iterations, $\Psi_m$ is computed as:

$$
\Psi_m = \Psi - \sum_{i=1}^{m-1} \beta_i \Phi(z_i)
$$

$$
\Psi_m = \sum_{i=1}^{N_x} \omega_i \Phi(x_i) - \sum_{i=1}^{m-1} \beta_i \Phi(z_i)
$$

$$
\Psi_m = \sum_{i=1}^{N_x} \omega_i \Phi(x_i) 
$$

(4.11)
In [36, 34], Scholkpf et al. and Blake et al. applied this approach using gaussian kernels. For this type of kernels, the following procedure is presented to obtain one reduced vector.

Since gaussian kernels satisfy \( K(z, z) = (\Phi(z) \cdot \Phi(z)) = 1 \) for all \( z \in \mathcal{X} \), equation 4.9 reduces to:

\[
\max_{z} (\Psi \cdot \Phi(z))^2
\]  

(4.12)

In order to find the maximum, we have:

\[
0 = \nabla_z (\Psi \cdot \Phi(z))^2 \\
0 = 2(\Psi \cdot \Phi(z))\nabla_z (\Psi \cdot \Phi(z))
\]  

(4.13)

From (4.13), the following sufficient condition for a maximum is:

\[
0 = \nabla_z (\Psi \cdot \Phi(z)) \\
0 = \nabla_z \sum_{i=1}^{N_s} w_i K(x_i, z) \\
0 = \sum_{i=1}^{N_s} w_i \nabla_z K(x_i, z) \\
0 = \sum_{i=1}^{N_s} w_i \exp(-\|x_i - z\|^2 / 2\sigma^2) (x_i - z)
\]  

(4.14)
In (4.14), $z$ can be found by the following fixed point iteration:

$$
0 = \sum_{i=1}^{N_i} w_i \exp(-\|x_i - z\|^2/2\sigma^2)(x_i - z)
$$

$$
z = \frac{\sum_{i=1}^{N_i} w_i \exp(-\|x_i - z\|^2/2\sigma^2)x_i}{\sum_{i=1}^{N_i} w_i \exp(-\|x_i - z\|^2/2\sigma^2)}
$$

$$
z_{n+1} = \frac{\sum_{i=1}^{N_i} w_i \exp(-\|x_i - z_n\|^2/2\sigma^2)x_i}{\sum_{i=1}^{N_i} w_i \exp(-\|x_i - z_n\|^2/2\sigma^2)}
$$

(4.15)

Considering the right hand of (4.15) as a function $g(z_n)$, then (4.15) is only guaranteed to converge when $|g'(z_n)| < 1$. In the case of (4.15) this requirement is not always met. It will only converge when the initial guess for $z$ is within the neighborhood of the solution. For this reason, several initial guesses must be taken.

Results obtained with this method showed a significant reduction in the computational time in the test phase of the Support Vector Machine. In [36], the number of evaluated vectors was reduced to 10% of the original number, losing less than 1% in accuracy. Blake et al. [34] presented results for face recognition using 100 reduced set vectors while the original SVM had 8,291 support vectors with almost no loss in accuracy.

A limitation on the algorithm is that the number of vectors needed in the reduced set to avoid any significant loss in accuracy cannot be determined a priori, and it is problem dependent. This algorithm has only been implemented for gaussian kernels.
4.3 Proposed Methods

The main approaches to the reduced support vector problem are: Subset methods and Reduced Set methods. While the first group of solutions attempts to select a reduced set of vectors \( \{z_i\} \) from the current set of support vectors \( \{x_i\} \), the second group finds vectors, not necessarily support vectors, which define an approximation to the decision boundary (4.4) that minimizes (4.5).

In the Subset methods, Downs et al. [9] presented the Exact Simplification method. It analyzed the linear independence of the support vectors in the feature space. They proposed to discard the linearly dependent ones without losing generalization nor accuracy in the trained support vector machine. This method has no control over the number of vector in the reduced set, which means that the number of reduced vectors \( (N_z) \) cannot be fixed a priori. Moreover, there may not be any reduction if there is no linear dependence among the support vectors.

These limitations are addressed by the Modified Exact Simplification method and the Simulated Annealing method presented in this section. The Modified Exact Simplification extends the idea presented in the Exact Simplification method in order to allow some control over the final number of vectors in the subset \( (N_z) \), establishing a trade-off between accuracy and the number of support vectors. The Simulated Annealing Method allows to specify a priori the number of support vectors \( N_z \) in the reduced set, and then uses the simulated annealing algorithm to search for them. One more Subset method is presented: The Explicit mapping. It uses standard linear algebra techniques to reduce the number of support vectors in the feature space and helps to illustrate the issues of reducing the number of support vectors in a high dimensional space.
In the Reduced Set methods, an iterative algorithm to find a reduced set of vectors was presented in [36] and [34]. This algorithm has only been implemented with gaussian kernels. In this work, the Polynomial Kernel Reduction is presented. It applies this iterative algorithm to polynomial kernels. For this purpose, an iterative fixed-point equation to find one reduced vector for a polynomial kernel is deduced.

Even though, the polynomial and gaussian kernels are the most commonly used, there are other types of kernels, and even a kernel can be created for a particular application. For this reason, the Standard Optimization Method is presented. Its main purpose is to provide a general algorithm to find a reduced set of vectors, regardless of the kernel used.

A description of the data sets used to test the proposed methods is presented next. Then, the proposed methods are introduced and evaluated.

### 4.3.1 Data Sets used for testing

Two data sets are used to test the proposed methods. In order to continue with the face tracking application as explained in the introduction chapter, the facial feature database gathered in [24] is used for lip classification as a real image processing data set. This database contains $10 \times 20$ pixel images for eyes, lips, eyebrows, nostrils and hair. A Support Vector Machine was trained using lips as positive samples while the others were considered negative. The SVM training used a second degree polynomial kernel and resulted in 2,261 support vectors, reporting 98.60% accuracy for the training set and 90.30% accuracy for the test set.
A simple data set was also created to help in the proof of concept of the different methods. This data set was created following indications from [28]:

- One Thousand 2-dimensional points were created at random. The range for each dimension varied from -1 to 1. (e.g. \( x = (a, b)/a, b \in [-1, 1] \))

- Each point was classified as: \( \text{sign}(\sin(a + b)) \)

The simple data set size and dimensions are small enough to help us understand the issues of mapping the input space into a high dimensional feature space. The simple data set was classified using the SMO algorithm with a second degree polynomial kernel in order to be consistent with the lip classification problem. Figure 4.3 shows both classes of the data set and the 106 support vectors obtained.
4.3.2 Explicit Mapping

This approach is the first attempt to understand the problem of reducing the number of support vectors. Even though its usefulness is very limited, it helps to identify the issues of mapping into a higher dimensional space and to consolidate the basic concepts of vector spaces. The use of kernels encapsulates the inner product in the high dimensional space allowing the classification of non-linear problems as linear ones. However it hides the mapping $\phi$, and therefore it is not known explicitly.

The main idea in this approach is to define a mapping $\phi$ that satisfies the kernel of choice. This mapping depends on the dimension of the input space as well as on the selected kernel. Once $\phi$ is known, it is used to map every support vector into the feature space. Then, following the ideas presented in [9], it is possible to check for linear dependence of the vectors, allowing to discard the unnecessary ones.

In the case of the simple data set (see Section 4.3.1, the dimension of the input space is two and the kernel chosen is a polynomial of second degree. Assuming two vectors $a, b \in \mathbb{R}^2$, $a = [a_1, a_2]^T$ and $b = [b_1, b_2]^T$:

$$K(a, b) = \Phi(a)^T \Phi(b)$$
$$K(a, b) = (a^T b + 1)^2$$
$$K(a, b) = (a_1 b_1 + a_2 b_2 + 1)^2$$
$$K(a, b) = (a_1 b_1 + a_2 b_2)^2 + 2(a_1 b_1 + a_2 b_2) + 1$$
$$K(a, b) = a_1^2 b_1^2 + 2a_1 a_2 b_1 b_2 + a_2^2 b_2^2 + 2a_1 b_1 + 2a_2 b_2 + 1 \quad (4.16)$$
Therefore, the following mapping ($\Phi$) is proposed:

$$\Phi(a) = [a_1^2, a_2, \sqrt{2}a_1, \sqrt{2}a_2, 2a_1a_2, 1]$$  \hspace{1cm} (4.17)

It can be extended to an input space of $M$ dimensions as follows:

$$\Phi(a) = [\Omega, \Upsilon, \Gamma, 1]$$  \hspace{1cm} (4.18)

where $\Omega = [a_1^2, \ldots, a_M^2]$, $\Upsilon = [\sqrt{2}a_1, \ldots, \sqrt{2}a_M]$, and $\Gamma = \sqrt{2}a_ia_j; i \neq j; i = 1, \ldots, M - 1, j = i, \ldots, M$.

The dimension of the feature space can be computed as:

$$x \in \mathbb{R}^M \Rightarrow \Phi(x) \in \mathbb{R}^P$$

$$P = 2M + \frac{M(M - 1)}{2} + 1$$  \hspace{1cm} (4.19)

Once $\Phi$ is defined, the support vectors are computed in the feature space and the algorithm shown in Figure 4.4 is followed. The set of independent support vectors in the feature space ($\mathcal{I}$) is initialized with the first support vector. Then, linear dependence is verified sequentially. Each vector is checked for linear dependence or independence against $\mathcal{I}$. For this purpose, the matrix $M$ is formed. Every vector in $\mathcal{I}$ and the vector to test $SV(t)$ constitutes a column in $M$. The rank of $M$ will show if the vector being checked is independent or dependent. If the rank is equal to the number of columns in $M$, then the vector being checked is independent and it is added to $\mathcal{I}$, otherwise the weights of the support vectors belonging to $\mathcal{I}$ are modified according to (4.20). These steps are repeated until all the vectors have
been evaluated.

\[
\Phi(x_k) = \sum_{i=1, i \neq k}^{N_i} c_i \Phi(x_i)
\]

\[
d_i = \left(\frac{w_k c_i}{w_i}\right), i = 1, \ldots, N_i
\]

\[
\hat{w}_i = w_i (1 + d_i), i = 1, \ldots, N_i
\]

where \(\hat{w}_i\) is the modified weight for the independent support vector \(x_i \in \mathcal{I}\) and \(N_i\) the number of vectors in \(\mathcal{I}\). Equation 4.20 describes a system of equations which can be solved to find \(c_i\).

In the case of the simple data set, the dimension of the feature space is 6, and therefore only 6 independent vectors in the feature space are necessary to represent the 106 support vectors. These vectors form the reduced set (see Figure 4.5 and compare to Figure 4.3).

Although this approach performed a significant reduction from 106 vectors to only 6 in the simple data set, its usefulness reduces when dealing with a more realistic data set. When applied to the lip classification problem with the facial feature database, the dimensionality of the feature space (20, 301) and the number of support vectors (2, 261) makes the process of checking linear dependence computationally expensive. The dimension of the feature space increases rapidly depending on the dimension of the input space. In the case of the second degree polynomial kernel, the increment has a complexity of \(O(N^2)\), where \(N\) is the dimensionality of the input space. Therefore, explicitly computing the vectors in the feature space limits the applicability of this method to simple problems (e.g. with low dimensional input space using a low degree polynomial kernel).

Another drawback of this method is that it requires to find an explicit mapping that fits to
the kernel of choice. This represents a relative straightforward task for the polynomial kernels, but it certainly represents a difficult one for other kernels such as gaussians.

Despite the limitations of this approach, it shows clearly the issues of reducing the number of Support vectors in high dimensional spaces. It also demonstrates that explicit computation of the vectors in feature space increases the computational complexity of the problem, and therefore it is not feasible for most applications.
4.3.3 Modified Exact Simplification

In [9], Downs et al. found the linearly dependent support vectors in the feature space and discarded them. Instead of using an explicit mapping $\phi$, they used the kernel to form a series of equation to find linear dependency as explained in (4.6). A limitation presented by the Exact Simplification algorithm is that it cannot control the amount of vectors reduced. Furthermore, if the support vectors being reduced are linearly independent, then there is no reduction at all. The Modified Exact Simplification (MES) extends the idea presented in the Exact Simplification method in order to allow some control over the final number of vectors in the reduced set ($N_z$), allowing a trade-off between accuracy and $N_z$.

The goal of the Modified Exact Simplification (MES) is to identify the vectors which are pseudo-dependent and discard them. To define the concept of pseudo-dependent, let’s consider Figure 4.6 where $SV_1, SV_2, SV_3 \in \mathbb{R}^3$ are support vectors in the feature space. $SV_1 = [1, 0, 0]$, $SV_2 = [0, 1, 0]$ and $SV_3 = [1, 1, 0.2]$. $SV_{12}$ is the projection of $SV_3$ in
the subspace spanned by \( SV_1 \) and \( SV_2 \), and \( \theta \in [0, \pi/2] \) is the angle between \( SV_3 \) and the projection \( SV_{12} \). \( SV_3 \) is pseudo-dependent on \( SV_1 \) and \( SV_2 \) if \( \theta \leq \rho \) where \( \rho \) is a pre-defined tolerance.

![Figure 4.6: Angle \( \theta \)](image)

The analysis of pseudo-dependence is done in the feature space indirectly by the use of the kernel as presented in [9]. The Exact Simplification algorithm can be considered a particular case of the Modified Exact Simplification in which the pre-defined tolerance has been set to 0.

MES uses the angle \( \theta \) to approximate the support vectors to a reduced (pseudo-independent) subset according to the procedure shown in Figure 4.7. The pseudo-independent subset \( (I) \) is initialized with the first support vector. Then, every support vector is checked for pseudo-dependence. If \( \theta < \rho \) then the approximated vector is considered close enough to the original one and the weights of vectors in \( I \) are modified accordingly (see (4.20)). Otherwise, the vector is added to \( I \).
The evaluation of linear dependence is done indirectly by using the kernel:

\[
\Phi(x_k) = \sum_{i=1, i \neq k}^{N_i} c_i \Phi(x_i)
\]

\[
\Phi(x)^T \Phi(x_k) = \Phi(x)^T \left( \sum_{i=1, i \neq k}^{N_i} c_i \Phi(x_i) \right)
\]

\[
K(x, x_k) = \sum_{i=1, i \neq k}^{N_i} c_i K(x, x_i)
\]  \hspace{1cm} (4.21)

In order to find the coefficients $c_i$, a random set of vector $\tau_i \in \mathcal{X}$ is used to form a system
of equations of the form:

\[ AC = B \]  \hspace{1cm} (4.22)

Where \( A \) is a \( 2N \times N \) matrix with elements \( A_{ij} = K(r_i, x_j), i = 1, \ldots, 2N; j = 1, \ldots, N \), \( B \) is a column vector with \( B_i = K(r_i, x_k), i = 1, \ldots, N \); and \( C \) is a column vector of size \( N \). \( N \) is the number of vectors in \( I \). Then, equation 4.22 is solved for \( C \) in the least squares sense defining the coefficients \( c \) for the best approximation spanned by vectors in \( I \). Note that the system of equations (4.22) has \( 2N \) equations to avoid the case in which some of the random vectors \( (r_i) \) are parallel in the feature space.

The angle \( \theta \) is calculated as:

\[ \theta = \arccos \left( \frac{\langle AC, B \rangle}{\|AC\| \|B\|} \right) \]  \hspace{1cm} (4.23)

This method was applied to the simple data set and the lip classification problem using the facial feature database. Because of the small dimension of the simple data set, the results are not much different from the Explicit Mapping method. Six vectors were found to be independent. This result did not change while varying the values of \( \rho \) between 0.1 and 0.001 degrees.

MES was applied to the lip classification problem using the facial feature data set. Results are reported in terms of accuracy and number of support vectors while varying the tolerance \( \rho \) from 0.005 to 0.04 degrees. Recalling from Chapter 2, the lip SVM classifier was composed of 2,261 support vectors and had a classification accuracy of 98.60% for the training set and 91.30% for the test set.
Table 4.1 shows the results when the support vectors were presented to the algorithm in the order they had after the training stage. Table 4.2 shows the results when the support vectors were sorted according to the absolute value of their associated weights ($w_i$) in a descending order. Figure 4.8 summarizes both tables.

<table>
<thead>
<tr>
<th>$\rho$ Degrees</th>
<th>Accuracy Training Set</th>
<th>Accuracy Test Set</th>
<th>Number of Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005</td>
<td>98.70%</td>
<td>90.27%</td>
<td>2155</td>
</tr>
<tr>
<td>0.010</td>
<td>96.56%</td>
<td>88.63%</td>
<td>1721</td>
</tr>
<tr>
<td>0.015</td>
<td>87.63%</td>
<td>82.79%</td>
<td>1335</td>
</tr>
<tr>
<td>0.020</td>
<td>77.94%</td>
<td>70.30%</td>
<td>1021</td>
</tr>
<tr>
<td>0.030</td>
<td>59.45%</td>
<td>59.57%</td>
<td>607</td>
</tr>
<tr>
<td>0.040</td>
<td>56.13%</td>
<td>58.57%</td>
<td>435</td>
</tr>
</tbody>
</table>

Table 4.1: Modified Exact Solution: Results for the lip classification problem

<table>
<thead>
<tr>
<th>$\rho$ Degrees</th>
<th>Accuracy Training Set</th>
<th>Accuracy Test Set</th>
<th>Number of vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005</td>
<td>98.68%</td>
<td>90.67%</td>
<td>2182</td>
</tr>
<tr>
<td>0.010</td>
<td>97.57%</td>
<td>89.19%</td>
<td>1721</td>
</tr>
<tr>
<td>0.015</td>
<td>84.43%</td>
<td>75.39%</td>
<td>1323</td>
</tr>
<tr>
<td>0.020</td>
<td>77.53%</td>
<td>67.50%</td>
<td>1003</td>
</tr>
</tbody>
</table>

Table 4.2: Modified Exact Solution results using sorted support vectors

As expected, increasing the value of $\rho$ leads to a lesser number of pseudo-independent vectors and to a lesser performance in accuracy. A value of $\rho$ less than 0.02 degrees loses all performance in classification. $\rho = 0.005$ degrees presents almost no loss in accuracy. However, it also maintains most of the support vectors.

This method is also sensitive to the order in which the support vectors are presented to the algorithm. To analyze this effect, MES is applied to a sorted set of support vectors. The support vectors were sorted according to the absolute value of their weights ($w_i$). Sorting
the support vectors does not seem to give an advantage per se. These results suggest that the accuracy achieved is related to the number of pseudo-independent vectors found, presenting clearly the trade-off between accuracy and reduction.

This method presents the advantage of computing linear pseudo-dependence indirectly through the kernels. Therefore, it can be used easily with any kernel of choice. An explicit mapping is not needed, avoiding expensive computations in the feature space. It also allows to establish a trade-off between accuracy and the number of support vectors based on the tolerance $\rho$. Among its limitations, MES cannot fix the number of vectors in the independent set $(I)$ a priori, and it cannot achieve a significant reduction without sacrificing performance in accuracy and generalization.
4.3.4 Simulated Annealing Method

In the last two approaches, the final number of vectors in the reduced set cannot be fixed a priori. As explained before, the time complexity of the test phase of a Support Vector Machine is proportional to the number of support vectors. For some applications (specially real-time ones), it might be of importance to fix the maximum number of support vectors to be used in the test phase.

The idea of the Simulated Annealing Method is to specify a priori the number of support vectors $N_z$ in the reduced set. From all the possible combinations of subsets containing $N_z$, the objective is to select the subset that characterizes a decision boundary (4.4) which minimizes (4.5). In order to find this subset, a search using the simulated annealing algorithm has been implemented and described in Figure 4.9.

The algorithm needs as parameters: the number of vectors in the reduced set $N_z$, an iteration limit: $t_{max}$, a relative large temperature: $q > 0$, the number of iterations the temperature would not change: $t_{still}$, the percentage in which the temperature decreases after $t_{still}$ iterations: $per$, an objective function $F$ and the threshold for the objective function: $tol$.

The algorithm starts by choosing a subset of $N_z$ vectors at random $S$ from all the support vectors and initializing the incumbent solution $S_f \leftarrow S$ and the solution index $t \leftarrow 0$. Then, the process iterates for $t_{max}$ times or until the value of the objective function becomes less than $tol$. At each iteration, the algorithm randomly interchanges a vector in the current $S$ with one outside $S$ to form a new set $S_{t+1}$. If $S_{t+1}$ gives a better value in the objective function or has a probability of $\exp(-\Delta Obj/q)$ then it is accepted as a successful change. If the objective function value of $S_{t+1}$ is superior to that of the incumbent solution, then
$S_f \leftarrow S_{t+1}$. If $t_{\text{still}}$ iterations has passed since the last temperature change, then a temperature reduction occurs: $q \leftarrow q \times \text{per}$

An important element of the this algorithm is the choice of an objective function. Computing the objective function should be done quickly since the overall time complexity of the Simulated Annealing method depends on it. The accuracy of the training set (or any other set) appears to be a logical option for an objective function:

$$\frac{\sum_{i=1}^{N_s} \Theta(x_i)}{N_s}$$

(4.24)
given

\[
\Theta(x_i) = \begin{cases} 
1 & \text{if } y_i = v(x_i) \\
0 & \text{otherwise}
\end{cases}
\] (4.25)

where \( y_i \) is the classification label and \( v(x_i) \) is the reduced score function \((4.2)\):

\[
v(x) = \sum_{i=1}^{N_z} \beta_i K(x, z_i) - \zeta
\]

where \( N_z \) is the number of reduced vectors, \( \beta_i \in \mathbb{R}, \zeta \in \mathbb{R} \) and the reduced set of support vectors \( z_i \in \{x_i\} \).

The computation of \( v(x) \) requires to have the weights \( \beta_i \) associated to each reduced vector \( z_i \) as well as the bias parameter \( \zeta \).

According to [36], the optimal weights can be computed by as:

\[
\beta = (K^z)^{-1} K^{xz} \alpha
\] (4.26)

where \( K^z_{ij} = K(z_i, z_j) \) and \( K^{xz}_{ii} = K(z_i, x_j) \).

The bias value can then be obtained by maximizing \((4.24)\). However, the computation of \( \zeta \) requires itself a search algorithm to find an optimal value. In order to avoid this last computation, \((4.5)\) is considered as the objective function:

\[
\|\Psi - \Psi'\|^2
\] (4.27)
where $\Psi$ is defined in (4.3) and $\Psi'$ is defined in (4.4). The weights $\beta_i$ are calculated according to (4.26) and $\zeta$ is no longer required.

The objective function (4.27) measures how close the approximated boundary is to the real one and correlates very well to the misclassification rate (100% - accuracy) obtained, as shown in Figure 4.10. However, it needs the classification of an entire training set. Therefore, depending on the training set, it may require a large amount of computation.

This method was tested with the simple data set, with the following initialization: $t_{\text{max}} = 1500$, $q = 250$, $\text{per} = 0.95$, $t_{\text{stil}} = 10$ and $\text{tol} = 1E - 20$. In this case, the tolerance has been considered practically zero to allow the algorithm to run $t_{\text{max}}$ iterations. Table 4.3 presents the results in terms of number of support vectors, value of the objective function and accuracy of the training set. Figure 4.10 presents the value of the objective function and the misclassification rate (i.e. 100% - Accuracy achieved).

<table>
<thead>
<tr>
<th>No Vectors</th>
<th>Objective Function</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.067E+01</td>
<td>80.70%</td>
</tr>
<tr>
<td>2</td>
<td>1.289E+00</td>
<td>99.00%</td>
</tr>
<tr>
<td>3</td>
<td>6.071E-01</td>
<td>99.10%</td>
</tr>
<tr>
<td>4</td>
<td>2.225E-03</td>
<td>99.20%</td>
</tr>
<tr>
<td>5</td>
<td>1.149E-10</td>
<td>99.20%</td>
</tr>
<tr>
<td>6</td>
<td>9.237E-14</td>
<td>99.20%</td>
</tr>
</tbody>
</table>

Table 4.3: Simulated Annealing Method: Results for the Simple Data Set

The results for 1, 2 and 3 vectors were checked by doing an extensive search of the simple data set obtaining the same vectors and weights.

The results show the correlation between the objective function and the misclassification
rate. In the case of 6 vectors, the approximated boundary and the real one are the same. This result agrees with the previous methods which indicated that the dimension of the feature space is 6 and therefore the boundary can be represented with any 6 independent vectors in the feature space. Note that even though the accuracy with 4, 5 and 6 vectors is the same, there is a difference between the approximated boundary and the real one, which may be translated into loss of generalization.

Even though, the simulated annealing method shows good results for the simple data set, this approach is not viable for practical applications such as the lip classification problem using the facial feature database. The computational cost of evaluating the objective function at each iteration as well as the dimensions of a practical problem would require too much computational power. A possible solution is to create a parallel version of the algorithm. The main advantage of this approach is that it fixes the number of reduced vectors a
4.3.5 Polynomial Kernel Reduction

The Polynomial Kernel Reduction (PKR) belongs to the group of reduced set methods. In contrast with the previous proposed methods, this approach uses an iterative algorithm to find a set of reduced vectors $z_i \in \mathcal{X}$ which are not necessarily support vectors. These reduced vectors define an approximation to the decision boundary (4.4) that minimizes (4.5). In [36] and [34] an iterative algorithm is presented to find a reduced set of vectors. In previous work, this algorithm has only been implemented with gaussian kernels. The Polynomial Kernel Reduction method applies this algorithm to polynomial kernels using an iterative fixed-point equation to find one reduced vector with this type of kernels. The deduction of this equation is presented in detail. This approach is tested with a second degree polynomial to be consistent with the lip classification problem presented in Chapter 2.

The derivation of the iterative fixed-point equation is presented next. Then, a review of the algorithm and issues about its implementation are addressed.

Following the ideas presented in Section 4.2.2, equations (4.3) to (4.11), the goal is to maximize (4.9) which can be expressed in kernel form as:

$$\max_{z} \frac{\left(\sum_{i=1}^{N_z} w_i K(x_i, z)\right)^2}{(K(z, z))}$$

(4.28)
In order to find the maximum, we have:

\[ 0 = \nabla_z \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right)^2 \left( K(z, z) \right) \]

\[ 2K(z, z) \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right) \nabla_z \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right) = \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right)^2 \nabla_z \left( K(z, z) \right) \]

Assuming a polynomial kernel: \( K(a, b) = (a^T b + d)^p \):

\[ 2K(z, z) \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right) \nabla_z \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right) = \left( \sum_{i=1}^{N_s} w_i K(x_i, z) \right)^2 \nabla_z \left( K(z, z) \right) \]

\[ 2(z^T z + d)^p \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right) \nabla_z \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right) = \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right)^2 \nabla_z ((z^T z + d)^p) \]

\[ 2(z^T z + d)^p \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right) \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p x_i \right) = \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right)^2 p ((z^T z + d)^p - 1) z \]

(4.30)

Using (4.30), a fixed-point iteration can be defined to find \( z \):

\[ z^n = \frac{(z^T z + d) \left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p x_i \right)}{\left( \sum_{i=1}^{N_s} w_i (x_i^T z + d)^p \right)} \]

\[ z_{n+1} = \frac{(z_n^T z_n + d) \left( \sum_{i=1}^{N_s} w_i (x_i^T z_n + d)^p x_i \right)}{\left( \sum_{i=1}^{N_s} w_i (x_i^T z_n + d)^p \right)} \]

(4.31)

For the particular case, being examined in the lip classification application, \( d = 1 \) and \( p = 2 \).

Provided a starting point, this equation finds a vector \( z \) which maps into a principal component of the decision boundary in the feature space. However, this equation presents the same problems as (4.15). Convergence is not guaranteed and depends on the chosen starting point.
In order to calculate higher order reduced vectors $z_m, m > 1$, the algorithm presented in Section 4.2.2 is followed. A high level view of the algorithm is presented in Figure 4.11. In each iteration, the algorithm finds the reduced vector $z$ which minimizes $|| \Psi_m - \beta \Phi(z) ||^2$. In the first iteration $\Psi_m = \Psi$. In the next iterations $\Psi_m$ is computed as in (4.11):

$$
\Psi_m = \Psi - \sum_{i=1}^{m-1} \beta_i \Phi(z_i)
$$

$$
\Psi_m = \sum_{i=1}^{N_x} \omega_i \Phi(x_i) - \sum_{i=1}^{m-1} \beta_i \Phi(z_i)
$$

$$
\Psi_m = \sum_{i=1}^{N_x} \omega_i \Phi(x_i)
$$

(4.32)
where \( N_x = N_s + m - 1 \), \((\omega_1, \ldots, \omega_{N_x}) = (w_1, \ldots, w_{N_s}, -\beta_1, \ldots, -\beta_{m-1})\), and \((\chi_1, \ldots, \chi_{N_x}) = (x_1, \ldots, x_{N_s}, z_1, \ldots, z_{m-1})\).

At each iteration, the algorithm uses (4.31) to find the reduced vector \( z \). The weight associated with this vector is computed according to (4.10):

\[
\beta = \frac{(\Psi \cdot \Phi(z))}{(\Phi(z) \cdot \Phi(z))}
\]

The procedure will end when \( \|\Psi - \Psi'\|^2 \) is less than a pre-defined tolerance \( \rho \), or \( m = N_z \).

The implementation of the iterative equation (4.31) is the most important part of the algorithm. Issues on its implementation are discussed next. Figure 4.12 shows the flow diagram of the subroutine to find \( z \).

This implementation presents the usual parameters of an iterative algorithm: the maximum number of iterations \( \text{Iter}_{\text{max}} \) and the tolerance \( \text{tol} \). The selected \( \text{tol} \) defines a radius in the input space in which the vector \( z \) is considered to have converged. In implementation terms, the convergence criterion is to compute the infinity norm of \( z_{n+1} - z_n \) less than \( \text{tol} \). After \( \text{Iter}_{\text{max}} \) iterations, the algorithm will reset and start with a new initial point.

The second block in the flow diagram deals with the selection of the starting points to find \( z \). Burges observed that the vectors of the reduced set are found in the neighborhood of the support vectors [5]. Following this observation, the first support vector is used as the first starting point. If the algorithm fails, then the second is used and so on until the algorithm converges or all the support vectors have been evaluated. This is a modification introduced to the Reduced Set Method which suggested the use of random vectors when the iterative
Figure 4.12: Polynomial Kernel Reduction: Subroutine to find $z$

In order to test this modification, the algorithm's performance with support vectors as starting points was compared to the algorithm's performance when using random vectors as starting points. The training and test set of the lip classification problem were used for the comparison. Results in terms of closeness to the original boundary ($\|\Psi - \Psi'\|^2$), and number of iterations, are presented in Table 4.4 and Figure 4.13.

While the results in performance are similar using both support vectors and random vectors as starting points, the number of iterations is consistently lower using support vectors. The use of support vectors instead of random vectors as starting points proves to be a good modification.
<table>
<thead>
<tr>
<th>Number</th>
<th>$(|\Psi - \Psi'|^2)_{SV}$</th>
<th>Iter SV</th>
<th>Iter Rand</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV</td>
<td>Rand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8.7066</td>
<td>70</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>8.0439</td>
<td>68</td>
<td>106</td>
</tr>
<tr>
<td>3</td>
<td>7.4336</td>
<td>190</td>
<td>834</td>
</tr>
<tr>
<td>4</td>
<td>6.9312</td>
<td>264</td>
<td>1784</td>
</tr>
<tr>
<td>5</td>
<td>6.4923</td>
<td>625</td>
<td>9112</td>
</tr>
<tr>
<td>6</td>
<td>6.0595</td>
<td>766</td>
<td>15047</td>
</tr>
<tr>
<td>7</td>
<td>5.6784</td>
<td>1324</td>
<td>60457</td>
</tr>
<tr>
<td>8</td>
<td>5.3420</td>
<td>8654</td>
<td>119765</td>
</tr>
<tr>
<td>9</td>
<td>5.0206</td>
<td>10211</td>
<td>534812</td>
</tr>
</tbody>
</table>

Table 4.4: Polynomial Kernel Reduction: Comparison between random initialization and support vector initialization

![Graph showing comparison between support vectors and random vectors](image)

Figure 4.13: Comparison between support vectors and random vectors as starting points in the Lip Classification Problem

This experiment gave another important result. The vectors found when using support vectors ($z_{sv}$) and random vectors ($z_{rv}$) as starting points represent parallel vectors in feature space: $\Phi(z_{rv}) = \gamma \Phi(z_{sv})$, where $\gamma \in \mathbb{R}$. Using (4.23) as a measure of closeness, the angle
formed by the vectors is less than 0.001 degrees. The difference can be attributed to the
tolerance used in convergence (托l). This explains the similar results in (||Ψ - Ψ'||^2) of both
approaches.

Another parameter introduced into the implementation of the subroutine to find z (see Fig-
ure 4.12) is the Maximum norm (NormMax). The algorithm starts with a given value for
NormMax. If the infinity norm of z is greater than NormMax, it fails and the algorithm
returns to select another starting point. Due to the maximum norm parameter, the algorithm
will find the reduced vectors which infinity norm is bounded by MaxNorm.

This modification is introduced because the infinity norm of z has a tendency to increase
according to the number of reduced vectors previously found. According to (4.31) and in
the particular case of K(a, b) = (a^T b + 1)^2, the norm of the vector z depends on the inverse
of ||Ψ_m|| (see (4.33)). Since the Polynomial Kernel Reduction method reduces ||Ψ_m|| with
every reduced vector found, then the infinity norm of z has a tendency to increase. How-
ever, from previous experiments, it was found that the algorithm gave as result a direction
in the feature space rather than a vector (e.g. the algorithm gave as a result parallel vectors
in the feature space). From the different z_i ∈ X, it is desirable to find the vector z with the
smallest infinity norm possible. Although having the reduced set vectors bound in norm
increases the number of iterations, it reduces numerical instabilities in the process.

\[
\begin{align*}
Z_{n+1} & = \frac{(z_n^T z_n + 1) (\sum_{i=1}^{N_z} \omega_i (x_i^T z_n + 1) x_i)}{\left( \sum_{i=1}^{N_z} \omega_i (x_i^T z_n + 1)^2 \right)} \\
Z_{n+1} & = \frac{(||\Phi(z)||) (\sum_{i=1}^{N_z} \omega_i (x_i^T z_n + 1) x_i)}{||\Psi_m|| ||\Phi(z)|| \cos(\theta)}
\end{align*}
\]

(4.33)

The results of the final version of the algorithm using the simple data set and the facial
feature data set are presented next. Table 4.5 shows the results of the Polynomial Kernel Reduction method on the simple data set. The results agree with the ones presented in the Modified Exact Simplification and the Simulated Annealing methods. The simple data set can be represented with two vectors with practically no loss in accuracy. The performance in terms of accuracy achieved by Polynomial Kernel Reduction with only one reduced vector outperforms the one obtained by the Subset methods.

<table>
<thead>
<tr>
<th>Number</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSV</td>
<td>Simple Data Set</td>
</tr>
<tr>
<td>1</td>
<td>98.1%</td>
</tr>
<tr>
<td>2</td>
<td>99.1%</td>
</tr>
<tr>
<td>3</td>
<td>99.1%</td>
</tr>
<tr>
<td>4</td>
<td>99.2%</td>
</tr>
<tr>
<td>5</td>
<td>99.2%</td>
</tr>
<tr>
<td>6</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

Table 4.5: Polynomial Kernel Reduction: Results for the Simple Data Set

Figure 4.14 presents the reduced set vectors and their result in terms of accuracy and generalization (represented by $\|\Psi - \Psi'\|^2$) for the Lip Classification Problem. The accuracy achieved with 37 vectors for the training test is 98.33% and 89.53% for the test set. These values are very close to the original support vector machine which had 98.65% and 90.30% with 2,261 support vectors. The number of vectors in the reduced set is 1.63% of the number of vectors in the original set.

4.3.6 Standard Optimization Method

In [36], an iterative algorithm to find a reduced set of vectors was introduced. An application of this algorithm using gaussian kernels was presented in [34]. The Polynomial Kernel Reduction method, also applies the algorithm defined in [36] to polynomial kernels.
Both implementations are kernel dependent. Even though, the polynomial and gaussian kernels are the most commonly used, there are other types of kernels, and even a kernel can be created for a particular application. The goal of the Standard Optimization Method is to provide a general algorithm to find a reduced set of vectors, regardless of the kernel used.

As well as the previous approach, the standard optimization method (SOM) finds the vectors \( z_i \) which best minimize (4.5). Since the algorithm wants to be kernel independent, a fixed-point cannot be derived like in previous methods. Instead, SOM uses standard unconstrained optimization methods provided by Matlab's Optimization Toolbox: \textit{fminunc} and \textit{fminsearch}. The matlab function \textit{fminunc} is based on the quasi-Newton method and uses the BFGS formula for updating the approximation of the Hessian matrix. For more information about this optimization method see [4, 11, 12] and [37]. The function \textit{fminsearch} is a direct search method that does not use numerical or analytic gradients as in \textit{fminunc}. It
is based on the Nelder-Mead Simplex Method [22].

Figure 4.15: Standard Optimization Method Algorithm

The Standard Optimization Method follows the algorithm described in Section 4.2.2 and shown in Figure 4.15. The main difference is the way the vector $z$ is computed. In the first version of the Standard Optimization Method, the function $fminunc$ is used. Support vectors are given as starting points to perform the optimization, and the objective function is given by:

$$
-\frac{(\Psi \cdot \Phi(z))^2}{(\Phi(z) \cdot \Phi(z))} = -\frac{\sum_{i=1}^{N_z} w_i K(x_i, z))^2}{K(z, z)}
$$

(4.34)

Accuracy in the training and test set, as well as closeness to the original decision boundary ($||\Psi - \Psi'||$) are reported in Figure 4.16. The results are not as good as for the PKR method. It took 250 reduced vectors to achieve similar accuracy and generalization results.
to the ones obtained with 29 vectors using the Polynomial Kernel Reduction method. In order to improve these results, the matlab optimization function \textit{fminsearch} is added to the algorithm. To find \( z \), \textit{fminunc} is used to perform a gradient based search for the minima. A support vector is provided as a starting point. The resulting \( z \) is then used as starting point for \textit{fminsearch} which perform an iterative search. Figure 4.17 shows a comparison of performances among the Polynomial Kernel Reduction (PKR), the Standard Optimization Method using only \textit{fminunc} (SOM1), and the Standard Optimization Method using \textit{fminunc} and \textit{fminsearch} (SOM2). The performance is reported in terms of \( \| \Psi - \Psi' \| \), which represents the closeness of the boundary characterized by the reduced set of vectors to the original SVM decision boundary. Results show that SOM2 outperforms SOM1. They also show that the results of PKR are comparable to the results of SOM2.

![Figure 4.16: SOM: Results for the Lip Classification Problem using the quasi Newton Optimization Method](image-url)
4.4 Summary and Discussion

This chapter addressed the reduced support vector problem. The solution of this problem aims at reducing the computational cost involved in the test phase of a Support Vector Machine. Two main approaches exist to the problem: Subset methods and Reduced Set methods. Previous work on both approaches were reviewed and new methods were proposed. The following remarks summarize the chapter.

1. The contributions of this work to the Subset methods are: the Implicit Mapping method, the Modified Exact Simplification method and the Simulated Annealing method. The Implicit Mapping method is a useful method to illustrate the reduced support vector problem and the issues of working with a high dimensional space. The Modified Exact Simplification provides a way to reduce the number of support
vectors according to a tolerance parameter. However reduction of vectors also compromises accuracy. The Simulated Annealing method provides the ability to fix the number of reduced vectors a priori.

2. The Reduced Set methods are superior. This was expected since this kind of methods finds vectors whose representation in the feature space is a principal component of the decision boundary. Therefore this approach is preferred to solve the reduced support vector problem. Depending on the kernel, the Gaussian Kernel Reduction Method (Gaussian), the Polynomial Kernel Method (Polynomial) or the Standard Optimization Method (other kernels) could be applied.

3. A contribution of this work to the reduced set methods is the deduction of a fixed-point iteration equation to find a reduced vector when a polynomial kernel is used. This equation is used in the Polynomial Kernel Reduction method, which is an application of the approach given in [36, 34] to polynomial kernels instead of a gaussian kernels.

4. The Standard Optimization Method based on optimization functions from the Matlab Optimization toolbox shows that standard methods can be used to reduce the number of vectors in the classification stage of a Support Vector Machine. Its performance was close to the one presented by the Polynomial Kernel Reduction, and it can be improved by tuning the tolerance parameters of the optimization routines for each particular problem.

5. The squared of the Euclidian norm of the difference between the original and the approximated boundaries, $||\Psi - \Psi'||^2$, can be considered as a measure of generalization performance. One of the main advantages of support vector machines is that it creates a decision boundary $\Psi$ which shows good generalization or capacity to classify
correctly new samples. $\|\Psi - \Psi'\|^2$ shows how close is the approximated boundary characterized by the reduced set ($\Psi'$) to the original boundary ($\Psi$) established by the support vector machine. In that sense, the closer the boundaries are, the more the generalization properties of the original SVM are kept.

6. A library of Matlab functions has been created to reduce the number of support vectors. This library contains the implementation of all the approaches mentioned in this chapter.
Chapter 5

Tracking

This chapter presents the implementation of a face-tracking system based on Support Vector Machines. The goal is to analyze the integration of Support Vector Machines into a visual real-time tracking architecture. Face-tracking was selected among other tracking applications because it represents a prototype visual tracking problem and it has a large number of applications, specially in the fields of surveillance and human computer interaction.

In order to define the real-time tracking architecture to be used, a study of different implementations was performed. The real-time requirement imposes several constraints on the variety of techniques available. Most of these techniques were not designed for real-time applications. Since the main intention is to integrate Support Vector Machines, which are considered computationally expensive classifiers, it is necessary that the tracking architecture has a motion estimator to limit the area of search for the features of interest reducing the amount of computation required. Another important consideration was the selection of the features to track. It was recently reported in [14] that detecting facial features such as eyes and lips is superior to detecting the face as a whole. This observation was also taken into account.
Figure 5.1: Feature Based Tracking Architecture

The feature-based architecture (see Figure 5.1) has been designed specifically to meet the requirements presented above. Instead of tracking the whole face, lips and eyes are selected as features to be tracked independently. The architecture consists of several tracking modules, one per each feature, and a Data Fusion Stage. Each of the modules can be configured independently in order to take advantage of the characteristics of a specific feature. The data fusion algorithm exploits the relationships among different features to improve the overall (face) detection, tracking and reacquisition.

A visual tracking module is composed of pre-processing, classification and motion estimation stages, and has two operational modes: Detection and Tracking. In detection mode, the pre-processing and classifier combination is used to obtain an initial position of the feature of interest with high confidence. Then the module is switched to tracking mode, where a motion estimator is used to track the feature detected by the classifier. Both modes of operation give as a result a list of vectors \((f_i, s_i, x_i)\) containing a candidate identifier \((f_i)\), a
score indicating the likelihood of the candidate to be the actual feature \( (s_i) \) and its position \( (x_i) \).

The data fusion stage combines temporal and spatial information from different modules to determine the position of the tracked features and to decide whether the detection mode or the tracking mode of each module should be used. The current and previous data from the list of vectors given by the modules is used to restrict the number of candidates to be classified, and to weight the candidates in order to have the most accurate classification and to choose the mode of operation (detection or tracking). Once the final position of the individual features has been determined, they are used to update the motion estimators from the different modules.

Support Vector Machines are integrated into the architecture as feature classifiers. The motion estimator is divided into two functions: Prediction and Verification. The predictor would narrow down the search area for the feature being tracked. The verifier would find the position of the feature inside the search area. In the current implementation, these functions are performed by a combination of Kalman filters and template matching.

This chapter describes the use of Kalman filters as predictors in the feature-based architecture. The basic theory of Kalman filters is presented in Section 5.1, while Section 5.2 describes the selection of the Kalman filter parameters for the face tracking problem. Section 5.3 presents the results of combining Support Vector Machines and the motion estimator. This section also presents the impact of reducing the number of support vectors in the overall time performance of the feature-based architecture. Section 5.4 finalizes the chapter with summary and discussion.
5.1 Kalman filter Theory

Kalman filters have been used extensively in Computer Vision research in several applications related to object tracking; including face [17], eye [1], and gesture tracking [18]. The Kalman filter is an optimal estimator. The discrete filter can be implemented recursively [26] and can be divided in two stages: A prediction (Time Update) and correction (Measurement Update) stage. In a visual tracking system, the prediction of the Kalman filter is used to reduce the area where the classifier should look for the position of the object being tracked.

In its classical formulation [42], the discrete Kalman filter addresses the problem of estimating the state $x \in \mathbb{R}^n$ of a system governed by the linear stochastic difference equation:

$$X_k = AX_{k-1} + Bu_{k-1} + w$$

(5.1)

and a measurement $z \in \mathbb{R}^m$

$$Z_k = HX_k + v$$

(5.2)

where:

$w$ is a white noise random process that represents the uncertainty in the plant model

$v$ is a white noise random process that represents the measurement noise

$v$ and $w$ are independent processes.

$A$ is an $n \times n$ matrix that relates the state vector at time step $k - 1$ to the current step $k$

$B$ is an $n \times l$ matrix that relates the system inputs $u \in \mathbb{R}^l$ to the state $x$

$H$ is an $m \times n$ matrix that relates the state to the measurements $Z_k$

Defining $\hat{X}_k^-$ to be the a priori state estimate at step $k$ given knowledge of the process prior
to step $k$, and $\hat{X}_k$ to be the a posteriori state estimate at step $k$ given the measurement $Z_k$, then we have:

- **A priori Error**: $e_k^- = X_k - \hat{X}_k$

- **A posteriori Error**: $e_k = X_k - \hat{X}_k$

- **A priori estimate error covariance**: $P_k^- = E[e_k^-e_k^{T-}]$

- **A posteriori estimate error covariance** $P_k = E[e_ke_k^T]$

- **A posteriori state estimate** $\hat{X}_k = \hat{X}_k^- + K(Z_k - H\hat{X}_k^-)$

where $K = P_k^-H^T(HP_k^-H^T + R)^{-1}$ is the Kalman filter gain.

The Kalman filter gain $K$ is obtained by solving a linear quadratic optimal estimation problem, given by the following Riccati equation:

$$
\dot{P} = AF + PA^T - PH^T R^{-1}HP + BQB^T \\

P(0) = P_0
$$

(5.3)

where $P$ is the error covariance $Q = E[ww^T]$ is the process noise covariance matrix and $R = E[ww^T]$ is the measurement noise covariance.

The Kalman filter gain $K$ describes the behavior of the filter and determines if the measurement $Z_k$ or the predicted measurement $\hat{X}_k^-$ is trusted more to find the real state $X_k$.

Figure 5.2 shows the two stages in the recursive implementation of the Kalman filter and their equations.
The iterations start with the initial conditions $P_0^-$ and $\hat{x}_0^-$, which are provided to the correction stage. The correction stage then computes the Kalman gain, updates the estimate and the error covariance. These results are used by the prediction stage to generate the prediction (e.g. the prediction of the feature’s position in the next frame).

5.2 Kalman filter Parameter Selection

One Kalman filter was designed for each motion coordinate of the features in the image plane. In the case of eyes, each one has its own pair of filters. Therefore, six filters were designed to predict motion in only one direction. The selection of parameters for the Kalman filters is presented next.

Selecting the parameters of the Kalman filter means to define:

1. The state $X$. 

Figure 5.2: Kalman filter recursive implementation
2. The state-space matrices $A, B, H$.

3. The error covariance matrices: $Q$ and $R$.

4. The initial conditions $P_0^-$ and $\hat{x}_0^-$. 

In the current implementation, the guidelines proposed by Kohler in [18] were followed. He derived the parameter selection for a Kalman filter used in a visual tracking application for a human computer interface. He modelled the motion of a given object as the superposition of the motion of a massless particle with constant velocity and white noise acceleration. Since it is not possible to predict how the face or the person is going to move, it is acceptable to represent the arbitrary motion of the face as white noise. The discrete representation of this model is described as:

$$s_{k+1} = s_k + v_k \Delta t + w_a$$  \hspace{1cm} (5.4)

where $s_k$ is the position of the object (lips or eyes), $v_k$ is its velocity and $w_a$ is a random variable representing white noise acceleration.

Defining the state as:

$$X = \begin{bmatrix} s \\ v \end{bmatrix}$$  \hspace{1cm} (5.5)

where $s$ is the position of the feature and $v$ is its velocity. The model in (5.4) can be represented in state-space form as in equation 5.1 with:

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$  \hspace{1cm} (5.6)

$$B = [0]$$  \hspace{1cm} (5.7)
Since we are interested in measuring the position of the object $z$, the matrix $H$ should extract this information from the state vector $X$. Therefore, $H$ is defined as:

$$ H = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad (5.8) $$

Choosing the process noise covariance matrix $Q$ and the measurement noise covariance $R$ is a difficult task and depends on the particular application. The difficulty is increased by the fact that the Kalman filter depends on information extracted from an image and therefore is influenced by the algorithms and techniques used for this task.

In the current implementation, Kalman filters are used to estimate the position of the feature under tracking in the next frame. This position defines a search window where template matching is used to obtain the measurement of the position of the feature $z$ in the image. The template used has a size of $10 \times 20$ pixels and the search window covers $\pm 20$ pixels from the estimated position.

$R$ represents the error in the measurement of the object's position. The measurement is given by the output of the template matching in the search window. The highest correlation value in the search window above 95% is considered as $z$. According to experimental data the sub-image selected usually has a correlation value over 98%. Given the high correlation values and the area of the search window, the final measurement $z$ should have less than one pixel error. Assuming that the measurement error appears to be an error of $\pm 1$ pixel with the same probability, the square of the standard deviation is: $\frac{1}{3}((-1)^2 + 0^2 + (+1)^2) = \frac{2}{3}$, therefore:

$$ R = \frac{2}{3} \quad (5.9) $$
In the case of the process noise covariance matrix \( (Q) \), Kohler [18] derived the following formula:

\[
Q = \frac{a^2 \Delta t}{6} \begin{bmatrix} 2(\Delta t)^2 & 3\Delta t \\ 3\Delta t & 6 \end{bmatrix}
\]

where \( a \) is the maximum acceleration of the object of interest, and \( \Delta t \) is the difference in frames. In our particular application, \( a \) was determined by performing experiments with different persons at 15 and 30 frames per second. The maximum \( a \) obtained was considered:

\[
\begin{align*}
\hat{a} &= 12 \frac{\text{pixel}}{\text{frame}} @ 15 \text{ fps} \\
\hat{a} &= 6 \frac{\text{pixel}}{\text{frame}} @ 30 \text{ fps}
\end{align*}
\]

In the case of the initial conditions \( \hat{X}_0^- \) and \( P_0^- \), Kohler showed that they do not need to be precise since they will be updated by the Kalman filter. \( \hat{X}_0^- \) is defined as:

\[
\hat{X}_0^- = \begin{bmatrix} s_0^- \\ 0 \end{bmatrix}
\]

where \( s_0^- \) is the initial position detected by the Support Vector Machine classifier. Since the velocity cannot be determined from this model, it is considered to be zero, although it may be different in practice.

Given \( s \), the maximum distance that the object can move and \( v \), its maximum velocity, the maximum position error \( x_k - \hat{x}_k^- \) will be \( s/2 \). Let’s assume that the distance the feature travels varies according to a gaussian distribution with standard deviation \( 2\sigma_s = s/2 \). Let’s assume the same holds for the velocity, \( 2\sigma_v = v/2 \). Then, the initial condition \( P_0^- \) can be
computed as:

\[
P_0^- = \frac{1}{16} \begin{bmatrix} s^2 & 0 \Delta t \\ 0 & v^2 \end{bmatrix}
\]

(5.14)

In our experiments, the following values were found for \( s \) and \( v \):

\[
s = 6 \text{ pixel @15 fps}
\]  
(5.15)

\[
s = 3 \text{ pixel @30 fps}
\]  
(5.16)

\[
v = \frac{6 \text{ pixel}}{\text{frame}} \text{ @15 fps}
\]  
(5.17)

\[
v = \frac{3 \text{ pixel}}{\text{frame}} \text{ @30 fps}
\]  
(5.18)

### 5.3 Results

This section presents the timing results of using Support Vector Machines in the tracking architecture described in Chapter 2. Three people were asked to participate in the experiments with the face tracking application. Five videos per person were timed and analyzed. Each of the videos lasted approximately 2 minutes yielding information for 50000 frames.

<table>
<thead>
<tr>
<th></th>
<th>#SV</th>
<th>Min(ms)</th>
<th>Max(ms)</th>
<th>Avg(ms)</th>
<th>Avg Time/point(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>2415</td>
<td>45.58</td>
<td>167.00</td>
<td>102.28</td>
<td>1.21</td>
</tr>
<tr>
<td>Lips</td>
<td>2261</td>
<td>34.09</td>
<td>197.74</td>
<td>72.73</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Table 5.1: SVM Classification Time

Table 5.1 shows the maximum, minimum and average time in milliseconds to perform a classification with the support vectors resulting from the SMO training as presented in Chapter 2. The average time to classify a single feature candidate is also presented. Both
classifiers, for eyes and for lips, need more than one millisecond per candidate. Their average classification time in both cases exceeds 33.3 milliseconds, which means these classifiers would not be able to achieve real-time performance by themselves.

<table>
<thead>
<tr>
<th></th>
<th>#SV</th>
<th>Avg(ms)</th>
<th>Avg fr on track</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>2415</td>
<td>7.54</td>
<td>26.76</td>
</tr>
<tr>
<td>Lips</td>
<td>2261</td>
<td>6.46</td>
<td>26.98</td>
</tr>
</tbody>
</table>

Table 5.2: SVM and Kalman Filters Tracking Time

Table 5.2 summarizes the time performance of the support vector machines classifiers in combination with Kalman filters and template matching. The classifier is used to detect the initial position of the feature. This initial step may introduce a noticeable delay in the video stream, which is then recovered in the subsequent frames where template matching is used. The average time per frame for this latter step is 4.1 milliseconds. Therefore, the average process time per frame is reduced to less than 8 milliseconds in both features.

The feature-based architecture includes a latter step of data fusion, which basically combines the information of all the features to improve reacquisition when the track of one of them is lost. This final stage helps to reduce the number of candidate features to be evaluated when reacquiring one feature, improving time performance. Figure 5.3 presents the average distribution of the computation in the feature-based architecture based on the two-minute videos.

Even though, support vector machines are used on average once every 20 frames, the complexity of evaluating them takes 42% of the computation time. The computational time spent in the motion estimation step is similar (46%), however the motion estimator subroutine is called 20 times more often than the support vector machines and its computation
Tracking application: Distribution of Computation using Support Vectors

Figure 5.3: Distribution of Computation

is divided among 20 frames in average while support vector machines concentrate in only one frame. The pre-processing stage and the data fusion stages combined take 12% of the computation. Reducing the computational time spent in SVM classification would clearly improve the overall time performance of the tracking architecture.

Following the ideas presented in chapter 4, the support vectors for both classifiers were reduced until the metric $\|\Psi_m\|$ fell below 1. The results show that with far less number of vectors, the accuracy in test and training sets are similar to the accuracy of the original SVM (see Table 5.3 below).

Table 5.4 shows the improvement in time performance when the reduced support vectors are used. These results even suggest that SVM classifiers could be used in every frame and
Table 5.3: Results of reducing the number of Support Vectors

<table>
<thead>
<tr>
<th></th>
<th>#SV</th>
<th>Accuracy Training Set</th>
<th>Accuracy Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>2415</td>
<td>98.8</td>
<td>91.1</td>
</tr>
<tr>
<td>Lips</td>
<td>2261</td>
<td>98.7</td>
<td>91.1</td>
</tr>
<tr>
<td>Eyes</td>
<td>40</td>
<td>98.7</td>
<td>90.4</td>
</tr>
<tr>
<td>Lips</td>
<td>37</td>
<td>98.3</td>
<td>89.5</td>
</tr>
</tbody>
</table>

real time performance could still be achieved. Table 5.5 shows the results of integrating the classifiers with Kalman filters and template matching. The average time per feature is now less than 4 milliseconds. Comparing with the results showed in Table 5.2, the average time per frame has been lowered by half in each feature.

Table 5.4: Reduced Support Vectors Classification Time

<table>
<thead>
<tr>
<th></th>
<th>#SV</th>
<th>Min(ms)</th>
<th>Max(ms)</th>
<th>Avg(ms)</th>
<th>Avg Time/point(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyes</td>
<td>40</td>
<td>0.9139</td>
<td>3.3485</td>
<td>2.0509</td>
<td>0.0242</td>
</tr>
<tr>
<td>Lips</td>
<td>37</td>
<td>0.6850</td>
<td>3.9729</td>
<td>1.4613</td>
<td>0.0228</td>
</tr>
</tbody>
</table>

Table 5.5: RSV and Kalman Filters

With the introduction of the reduced support vectors into the feature-based architecture, the distribution of the computation changes significantly (see Figure 5.4). The computational time spent in the classifiers is reduced from 42% to 1.4%.

5.4 Summary and Discussion

In this chapter, Support Vector Machines have been successfully integrated to a tracking architecture. This architecture combines computational expensive classifiers such as SVMs
Figure 5.4: Distribution of Computation using Reduced Support Vectors

with motion estimators to reduce the overall computational cost. Even though experimental results indicate that real-time performance could be achieved in average, there would almost be no time remaining for other processes. The frames where Support Vector Machines are used show a significant delay. Once the feature is locked, the subsequent frames are processed much faster.

Reducing the number of support vectors is problem dependent. The trade-off between reduction (e.g. time performance) and accuracy and generalization performance will vary according to the particular characteristics of the data set. However, the methods presented in this work can be applied to a large variety of applications.

In our particular application for face tracking, the classification performed by the original
SVM without reducing the number of support vectors took 42% of the total computation in the tracking architecture. This percentage was lowered to less than 2% by reducing the number of support vectors involved in the classification process. The Polynomial Kernel Reduction algorithm (see Section 4.3.5) was used. The number of support vectors was reduced by 98% of the original one. Due to the reduction in the computational cost of the classifier, the detection and tracking of the three features (2 eyes and the lips) can be performed at every frame.

Support Vector Machines are used as feature detectors in the tracking architecture implemented. Reducing the computational cost of SVM classification gives additional time for other tasks. This is critical if an application runs in real-time. This additional time can be allocated to the pre-processing, motion estimation or data fusion stages. In the case of the face tracking application presented, new issues can be addressed: Pre-processing could have an additional step for shape recognition, different input spaces for the SVM could be evaluated, a particle filter could be integrated into the data fusion stage, just to mention a few suggestions.

The face tracking application implemented in this work represents a prototype of the use of Support Vector Machines as classifiers in tracking applications. It also illustrates the impact of reducing the number of support vectors in the overall time performance of the tracking system. This application clearly shows that the use of Support Vector Machines in tracking applications is feasible and that it can be done in real-time.
Chapter 6

Conclusions and Future Work

In this work we have investigated the feasibility of using Support Vector Machines as classifiers in real-time visual tracking systems. The focus has been on reducing the training time and reducing the computational time involved in classification.

To reduce the training time, parallel computing techniques were applied to develop several parallel algorithms based on Platt’s Sequential Minimal Optimization (SMO) algorithm. As a result, three new parallel algorithms were developed:

1. The Unbiased Sub-Optimal Blocked Parallel Algorithm introduces the use of the unbiased version of the SMO algorithm in parallel trainers. It presents a trade-off between accuracy and time performance. Experimental results show that this algorithm provides a better time performance than other sub-optimal algorithms when the number of training samples per processor is large.

2. The Modified Biased Sub-Optimal Blocked Parallel Algorithm modifies the calculation of the global bias parameter in a previous algorithm presented in [10]. Results show that this modification improves the performance in terms of accuracy.

3. The SMO-Chunking Parallel Algorithm is a new optimal parallel trainer which combines the SMO and the chunking algorithms. It achieves an optimal solution to the
training problem, although its time performance mainly depends on the final number of support vectors found.

Overall, sub-optimal training algorithms showed significant speed-up properties. However, they increase the number of support vectors and lose accuracy and generalization properties. If training time is the main constraint, then these algorithms are better. However, they are not suitable when the SVM classifier is intended for a real-time application due to the large number of support vectors.

In the case of the SMO-Chunking algorithm, even though it presents an optimal solution, its speed-up is a function of the of the number of support vectors rather than the number of processors.

To reduce the computational time during the testing phase different approaches to reduce the number of support vectors after training were investigated. Two main approaches 1) Subset methods and 2) Reduced set methods were investigated and several algorithms were developed and experimentally tested. The results are summarized below:

1. The Modified Exact Simplification algorithm allows a trade-off between accuracy and the number of support vectors by introducing a tolerance parameter into the algorithm presented in [9].

2. The Simulated Annealing algorithm presents the main advantage of specifying a priori the number of support vectors $N_z$ in the reduced set.

3. The Explicit Mapping algorithm uses standard linear algebra techniques to reduce the number of support vectors in a high dimensional (feature) space. This method is useful to illustrate the problem of reducing the number of support vectors and the issues of working with a high dimensional space.
4. The Polynomial Kernel Reduction algorithm is an application of the iterative algorithm of [36, 34] to polynomial kernels.

5. The Standard Optimization Method solves the reduced support vector problem by nonlinear optimization. It was implemented using the Matlab Optimization toolbox and can be applied to any kernel.

Comparing the reduction performance of both approaches, the Reduced Set methods performed better. Therefore, in order to reduce the number of support vectors of a given SVM, it is suggested to use these methods depending on the kernel: the Gaussian Kernel Reduction method (Gaussians), the Polynomial Kernel method (Polynomials) or the Standard Optimization Method (other kernels).

Using these techniques a real-time face tracking system based on SVM was implemented and tested. The tracking system architecture also uses a combination of Kalman filters and template matching for motion estimation.

Results show that the use of Support Vector Machines in real-time tracking applications is feasible. It also demonstrated that a significant reduction on the number of support vectors can be achieved without compromising classification accuracy significantly. In our particular application for face tracking, the classification performed by the original SVM without reducing the number of support vectors took 42% of the total computation in the tracking architecture and less than 2% after the reduction was performed.

Reducing the computational cost of SVM classification gives additional time for other tasks. This is critical in real-time. Furthermore, this additional time can be allocated to the pre-processing, motion estimation or other stages of the visual tracking system.
6.1 Future Work

The following problems can be identified as suitable for further research.

1. Parallel reduced support vector training. Instead of performing parallel training followed by reduction of support vectors a simultaneous parallel training with support vector reduction should be investigated. In order to do this a more in depth study of the properties of the reduced support vectors and their parallelization is necessary.

2. Kernel design. In this work, the impact of selecting a kernel for classification was not addressed. Most of the applications found in the literature, select the kernel by trial and error with the most common ones being gaussian and polynomial. A study on the issues involved in selecting and appropriate kernel and designing a kernel for a particular application is worth considering.

3. Generalization Measure. One of the main advantages of support vector machines is that it creates a decision boundary $\Psi$ which shows good generalization or capacity to classify correctly new samples. The squared of the Euclidian norm of the difference between the original and the approximated boundaries presented in Chapter 4, $\|\Psi - \Psi'\|^2$, should be investigated further to see if it can be used as a generalization measure.
Bibliography


