Chapter 1

Literature Review

1.1 Introduction

In this chapter, previous work and research that has been performed in the fields relevant to the research carried out during this project, is mentioned and discussed. These fields include the quadcopter platform, computer vision and machine learning techniques.

1.2 Quadcopters

A quadcopter is an autonomous aerial vehicle (UAV) in a four rotor frame configuration. This configuration comes in an X or plus (+) shape, with the control equipment and sensors typically located at the centre. See Figure 1.1 for a picture of the SunKopter in a typical quadcopter X configuration. They can also come equipped with either four or eight motors and props, though the four-rotor variant is very common.

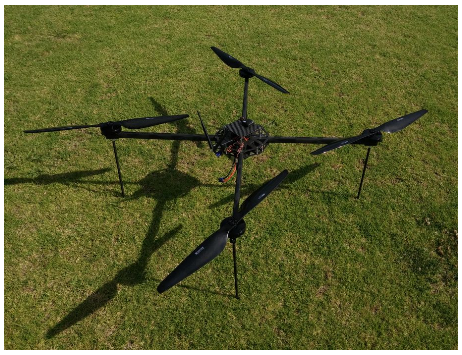


Figure 1.1: A picture of the SunKopter, as used by the Solar and Thermal Energy Research Group.

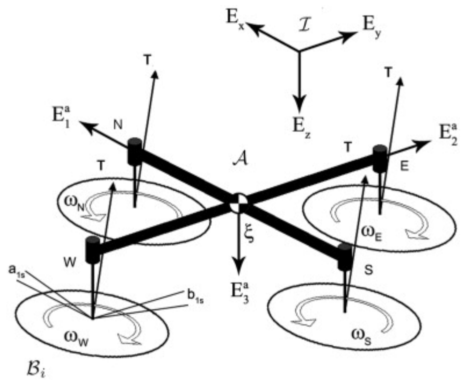


Figure 1.2: A diagram of the quadcopter model, including the blade flapping dynamics. Adapted from Pounds *et al.* (2010).

As described in Chapter REF CHAP1, the goal of this project is to determine the pose estimation error, i.e. the difference between a quadcopter's true pose and its estimated pose. To be able to do this, it is important that the dynamics and control strategy of a typical quadcopter is well defined and understood. This section sets out to discuss the dynamic modelling of a quadcopter, as well as the different control strategies that have been developed in the past.

Plant Modelling

When designing a control system, arguably the most crucial part of the process is to derive an accurate mathematical model of the plant that is to be controlled. The plant in this case is a UAV in a 'quadcopter' configuration.

As part of the X-4 Flyer project, Hamel *et al.* (2002) set out to derive a simple model for their plant using only rigid body dynamics and abstract force and torque actuators. As stated by Pounds *et al.* (2010), this model, like many others that were derived at the time, represents the quadcopter as a rigid body mass with inertia and autogyroscopics, affected only by gravity and actuator torques. Pounds *et al.* further argues that these simple quadcopter models do not accurately represent the complex helicopter-like behaviour exhibited by real quadcopters at high rotor speeds. These high-speed rotor effects include the blade flapping effect, which affect the quadcopter frame's oscillatory modes, rotor flapping from varying quadcopter yaw angles, and variable airflow velocities over the rotor blades from varying roll and pitch angles.

In an attempt to create a quadcopter model that will allow a quadcopter to be more accurately controlled at high rotor speeds, Pounds *et al.* set out to derive a model incorporating the rigid body dynamics, as well as the aerodynamic effects mentioned earlier. Their model is based on the diagram given in Figure 1.2.

Their resulting model was used to develop a simple hovering Proportional Integral Differential (PID) attitude and altitude controller for the purpose of model verification. The results show that the drone stabilised itself in indoor flight with $\pm 1^\circ$ of precision and $\pm 5^\circ$ of precision during outdoor flight. The lower precision during outdoor flights is due to the added wind disturbances, etc. Thus, the model, as it stands, is sufficiently accurate to safely control a drone for hovering. However, they did not compare their model, which includes aerodynamic effects, against the model of Hamel *et al.*, which is based on rigid body dynamics.

Control Strategies

Introduction

After an accurate model, the next most important part of a good control system is the controller itself. Many controllers have been implemented and tested on quadcopters over the years. Different control strategies have also been investigated. Some of the most prevalent control strategies and controllers are discusses.

Indoor vs. Outdoor Control

There are different types of quadcopters, each of them equipped with different sensors and equipment. The two main types of quadcopters are indoor and outdoor quadcopters. Each of them work in different ways and implement very different control schemes.

Indoor drones may or may not come equipped with an on-board inertial measurement unit (IMU), which includes an accelerometer and gyroscope, for stability control. However, they commonly solely rely on an external motion detection system which tracks the drone, providing position and rotation (also known as pose) feedback to the controller, thereby closing the feedback control loop. These drones are exceptionally accurate and capable of performing remarkable acrobatic feats. However, their use is restricted to carefully regulated and controlled indoor environments.

Outdoor drones, without the luxury of having very accurate external sensors available, have to rely on their on-board sensors to provide the controller with pose feedback. The on-board sensors these drones come equipped with may vary between quadcopter platforms, however they almost certainly come equipped with an IMU to provide pose data. However, since the IMU readings for position data drifts with time due to integration errors, a global positioning system (GPS) sensor is added to provide a base-line reading for a quadcopter's position. Other sensors that may be included are magnetometers, barometers, visual feedback sensors and sonar sensors. To combine the readings of the different sensors, a filtering technique, such as the Extended Kalmann Filter, is used. The pose error of the combination of the different readings are, in theory, less than the most accurate sensor in the suite, but this has not yet been proven and an exact error margin is yet to be determined.

Hovering Control

The stable hovering of a quadcopter has been the focus of many projects and research papers in the past 15 years. As a result, many different control methods and schemes have been investigated, implemented and compared. Hovering control refers to a quadcopters ability to hover and remain stable at a set point in three-dimensional space.

Bouabdallah *et al.* (2004), as part of their 'OS4' project, compared the modern linear quadratic regulator (LQR) and classic PID controller with one another, with respect to the control performance (disturbance rejection, reference tracking, etc.) of a quadcopter.

They found that the PID controller produced better results than the LQR in terms of reference tracking and dynamic performance. This was a surprising result, since LQR controllers normally excel at controlling unstable, underactuated plants such as a quadcopter platform. They suspect that the reason for this result may be because the PID controller is better at handling plant uncertainties, since they neglected the effects of actuator dynamics such as blade and rotor flapping, in their drone model. They do expect that an LQR controller will produce superior results if a more accurate model is used by taking these effects into account.

Some researchers have also investigated controlling a drone using an H-infinity (H_{inf}) control structure and a model predictive controller (MPC). Most notably, Raffo *et al.* (2010) have done extensive research on this topic. MPC's are computationally efficient and modern controllers that drive a plant's state to a reference state within predefined constraints (eg. motor saturation, model dynamics, etc.), while a properly designed non-linear H_{inf} controller is very good at rejecting disturbances (eg. wind gusts, motor vibrations, etc.) and are robust to model uncertainties. They opted to combine the two controllers in an intelligent manner in order to extract the most efficient performance out of their drone.

In their simulations they found that the resulting controller performed admirably, presenting good reference tracking, proving to be robust with uncertain mass and inertia terms and deals well with disturbances on all six degrees of freedom at different points in time. However, they are yet to implement and test their controller configuration on a real drone. Although the algorithms and methods they used are computationally efficient, it may still prove to be too computationally intensive for the limited computing power on-board a drone. Given the fast growth of processing power, however, this controller configuration may become a more viable option in the near future.

Controllers for enabling a drone to hover have already been successfully designed and implemented, and it is therefore possible to accurately control a drone during hovering operations.

1.3 Computer Vision

Introduction

Computer vision is a diverse field which primarily focuses on devising methods for acquiring, processing, analysing and understanding images captured of the real world. There are various sub-fields of research, but a common theme across all the fields is to mimic the human ability to perceive and understand an image, and illicit an appropriate reaction to different visual inputs.

The fields of interest for this thesis, is the object detection, tracking and pose estimation fields. This section aims to discuss the work that has been done in these fields.

Camera Matrix

An image is a collection of two-dimensional vectors representing a collection of three-dimensional space vectors. These collections of vectors are related by a matrix C, known as the camera matrix. The camera matrix contains the intrinsic parameters of the camera that recorded the image, i.e. the focal lengths and principle point of the image, as well as the extrinsic parameters of the camera, i.e. the translation and rotation information. The camera matrix C given is given by Equation 1.1, as derived by Heikkila and Silvén (1997)

$$C = NP \tag{1.1}$$

where N is given by the matrix in Equation 1.2.

$$N = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

Here, f_x and f_y describe the focal lengths of the camera and u_0 and v_0 are the principal points. The pose matrix P is given by the matrix in Equation 1.3.

$$P = \begin{bmatrix} R|T \end{bmatrix} = \begin{bmatrix} r_{11} & r_{21} & r_{31} & t_1 \\ r_{21} & r_{22} & r_{32} & t_2 \\ r_{31} & r_{23} & r_{33} & t_3 \end{bmatrix} \tag{1.3}$$

In the matrix P, R is a 3×3 matrix describing the rotation of the camera, and T is a three-dimensional vector describing the translation of the camera.

The two-dimensional image projection of an object in three-dimensional world space is related through the camera matrix C with the relation given in Equation 1.4. The camera matrix is commonly determined through some camera calibration procedure. One such a procedure is discussed in Section 1.3.

$$\begin{bmatrix} x_c & y_c & 1 \end{bmatrix}^T = C \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix}^T \tag{1.4}$$

Camera Calibration

A properly calibrated camera is a very important part of any computer vision system, since the accuracy of the data extracted from an image may strongly depend on the accuracy of the calibration. The goal of a camera calibration procedure is to produce the complete camera matrix C, as given in Equation 1.1, as well as finding the camera's distortion coefficients introduced by low-quality or fish-eye lenses. There are various camera calibration procedures available, from the two-step calibration described by Melen (1994) to the classical approach given by Slama *et al.* (1980), where a non-linear error function is minimised. However, the minimisation problem presented by Slama *et al.* is computationally inefficient and slow, while Melen's method does not account for image distortion and correction. A popular calibration method is the four-step method, proposed by Heikkila and Silvén (1997) as an extension to the two-step method which was the prevalent calibration procedure at the time.

The 'calibrateCamera()' function of the OpenCV computer vision library [Bradski *et al.* (2000)], makes use of the four-step method.[1] The fine details of the method is beyond the scope of this research, however a broad overview of the steps and equipment required to calibrate a camera, is provided.

To perform the calibration and find the camera matrix, OpenCV requires two sets of data: one two-dimensional image data set, $[x_c\ y_c\ 1]$, as well as a set of corresponding three-dimensional data points, $[x_w\ y_w\ z_w\ 1]$. This implies that image data of an object where the dimensions and coordinates of certain features are known, must be recorded. In practice, any well-characterised object can be used for calibration. For example, some calibration methods rely on a three-dimensional cube covered in precisely laid out markers. However, since manufacturing and distributing such precisely constructed objects to a large audience is infeasible, OpenCV opts to use a more convenient flat, regular pattern, such as chessboard or asymmetrical dot pattern. Figure 1.3 shows an example of a typical chessboard pattern generated by OpenCV. With these flat patterns, the features used to populate the data sets would be the square corners on a chessboard, i.e. where a black block meets another black block, or the dots on an asymmetric dot pattern. The drawback to this approach, however, is that multiple views of the flat calibration pattern is required, whereas a single image of a three-dimensional object would suffice.

In the case of the chessboard pattern, acquiring the two-dimensional pixel coordinates of the corners is accomplished by using OpenCV's 'findChessboardCorners()' and 'findCornerSubPix()' functions, which makes use of a Harris corner detection algorithm, first described by Harris and Stephens (1988). The three-dimensional coordinates is fed to the calibration function according to an axis-system and measurement unit defined by the programmer. These coordinates can be as simple as a vector containing the corner coordinates in square units, e.g. the corner from Figure 1.3 will then be represented as $(3, 2, 0, 1)$ in homogeneous coordinates (note that z_w will be zero since the board is flat).

[1]The version used is OpenCV v2.4.8

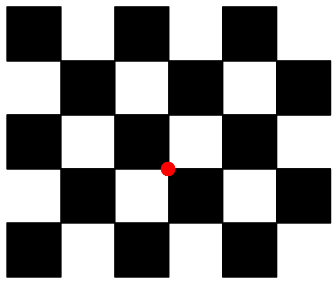


Figure 1.3: An example of a typical chessboard pattern used for calibration.

Next, for best calibration results, OpenCV recommends that camera calibration takes place within a well-lit room with a white background, using a calibration pattern with a wide white border in clear view of the camera and at different locations and orientations relative to the camera. These recommendations are mainly to increase the contrast between the black features and the white background and make it easier for OpenCV to accurately find each feature's pixel coordinates. Furthermore, the more diverse the location and orientation data is, the more accurate the estimate for the intrinsic camera parameters will be.

With the of two-dimensional and three-dimensional data sets recorded, the calibration function can determine the camera matrix C.

Principle n-Points Problem

The Principle n-Points (PnP) problem, as stated by Horaud *et al.* (1989), 'is the problem of finding the position and orientation of a camera with respect to a scene object from n correspondence points', where the scene object would normally be a well-characterised calibration object or pattern. It is a well-researched sub-field of computer vision with various solutions to the problem that have been proposed. These solutions include some non-iterative solutions, such as the P3P solution proposed by Gao *et al.* (2003) and the PnP solvers by Lepetit *et al.* (2009) and Schweighofer and Pinz (2006), and iterative solutions, such as the method proposed by Lu *et al.* (2000).

It was found that iterative methods produce very accurate results, but can become unstable if its not properly initialised and can take a long time to converge. Conversely, the non-iterative ePnP method by Lepetit *et al.* implements Schweighofer and Pinz's robust solver and produces results whose accuracy is comparable to those produced by its iterative counterpart. However, it produces these results in a fraction of the time, having a big $\mathcal{O}$ complexity that grows linearly ($\mathcal{O}(n)$), as opposed to competing non-iterative methods which commonly have a big $\mathcal{O}$ complexity to the order of 4 or more ($\mathcal{O}^4(n)$). On the downside, the accuracy of the ePnP's methods results are fairly dependant on the number of sample points, i.e. the number of point correspondences between the three-dimensional features and their two-dimensional projections.

The OpenCV library has implementations of the P3P and ePnP methods, as well as its own implementation based on Levenberg-Marquardt [Levenberg (1944) and Marquardt (1963)] optimisation, where the pose (i.e. the combination of translation and rotation vectors) that minimises the reprojection error, that is the sum of the squared distances between the actual two-dimensional points and the projected two-dimensional points, is determined and selected.

OpenCV's 'solvePnP()' function can be used to determine the pose of a camera relative to a calibration pattern. This can be accomplished as follows. After the camera calibration procedure, the intrinsic parameter matrix N from Equation 1.2 is determined. Then, using a calibration pattern, a set of three-dimensional object coordinates and its corresponding two-dimensional projection can be obtained. Following from Equation 1.4, with the matrix C and two-dimensional image projections and three-dimensional object coordinates, the pose matrix P can then be found. This matrix then contains the translation and rotation data for the camera relative to the calibration pattern.

Random Sample Consensus

For both the camera calibration and PnP solving functions, two-dimensional image projection data of three-dimensional object data is required. As mentioned, OpenCV's 'findChessboardCorners()' function can automatically detect corner features on a chessboard pattern and provide the two-dimensional projection data. However, the methods employed are prone to erroneously classifying some features as corners, introducing unwanted noise into the system.

To remedy this, Fischler and Bolles (1981) proposed a new algorithm to iteratively sift through a data set and reject any outlier data. This algorithm is dubbed Random Sample Consensus (RANSAC) and is a commonly used method in the computer vision field where it is used to determine if an image feature, e.g. a corner on a chessboard that has been classified as such, has been classified correctly, thereby reducing the amount of noise in a data set.

1.4 Machine Learning

Introduction

Machine learning is a field of computer science with strong ties with the fields of mathematical statistics and optimisation. The field is well-established and has its roots starting with the paper by Turing (1950), where he poses the question, 'can machines think?'. Michalski *et al.* (2013) offers a somewhat more formal definition: 'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E'. This means that researchers in the field of machine learning are attempting to find efficient

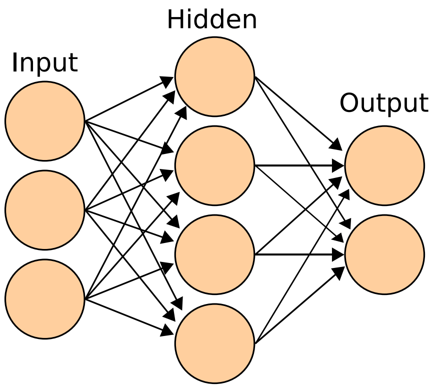


Figure 1.4: Diagram of a typical ANN [Commons (2006)].

methods and algorithms that will allow a computer to be trained to make predictions with an arbitrary input data set.

An example of a system that uses machine learning is the face detection software that commonly come packaged with digital cameras. Here, the camera has been trained to extract facial information from image data, and consequently can detect faces when presented with another image containing faces.

Various machine learning algorithms and types have been developed, each of them having their unique advantages and limitations. Here, brief discussions on the most prevalent machine learning methods is provided.

Artificial Neural Network

Introduction

Artificial neural network (ANN) algorithms is a family of machine learning algorithms that have seen a rise in popularity in recent years. The aim of ANN's is to model the vast network of interconnected neurons in a biological brain in such a way that it can be trained to recognise patterns and make decisions based on what it perceives, much like any animal or human would.

Normally, an ANN consists of a multitude of artificial 'neurons', or nodes, which can number anything between a few dozen to many millions, arranged in a series of layers. Each of the nodes in every layer is connected to each other node in the layers on either side, forming a vast network of interconnected nodes, forming something analogous to a living brain. A diagram of the layout of a simple ANN is given in Figure 1.4.

Each network has two special layers, called the input and output layers. The input layer accepts information from which the ANN's designer wants data extracted. The output layer is responsible for producing the output, which contains the information on how the network responded to the input excitation data. In-between these extreme layers lie the so-called 'hidden' layers. These layers form the majority of the network and is responsible for interpreting the input data and calculating and producing the networks output.

The connections between the hidden nodes are represented by a weighting factor which are determined during a training process. These weights define how much influence the nodes have over another, i.e. if a weight is positive, it excites another node, whereas if its negative, it suppresses it. The input information traverses the hidden layers, activating the node with the next highest connection weighting. The output data is then determined by which of the nodes were traversed in the hidden layers.

Consider a simple example. You wish to create an ANN that can recognise a whether a picture contains a man or a dog. You train it with 25 images of different men and dogs, telling the ANN which is which. Then, after it has been trained, you show it a picture containing a young boy, which is totally unfamiliar to the network. Based on the way you trained it, however, the network should recognise enough human features in the child to come to the conclusion that it is more likely that the child is a man than it is a dog.

This ability to classify information that technically falls out of the network's training environment is one of the strengths of ANN's. Other advantages, as stated by Tu (1996), are ANN's ability to implicitly detect any non-linear relationships between multi-dimensional input and output dimensions and they can be trained using different training schemes. Modern software packages and libraries have also made it fairly easy to develop a model without any formal statistical training. Some drawbacks of ANN's are that they may require an immense amount of computing power if many nodes are initialised, they are prone to overfitting data and the trained models are extremely 'black-box' solutions, making it very difficult to identify and characterise the relationships between the nodes.

Different ANN topologies, i.e. layouts, have been developed and proposed. Some of them are briefly discussed here.

Feed-forward Network

The oldest, and arguably the simplest, ANN topography is the feed-forward network (FFN), also referred to as multi-layer perceptron if there are multiple node layers in the hidden layer. Its layout is typically very similar to the layout given in Figure 1.4, with a single input and output layer, with multiple hidden node layers.

The FFN used some form of supervised training algorithm, with the back-propagation algorithm commonly used, where the hidden nodes are adjusted until the output the network produces is as close as possible to the target output specified by the designer. This configurations biggest attraction is its simplicity, relatively fast training speed, depending on the number of hidden layers, and its ability to derive non-linear relationships between dimensions. However, FFN's are prone to converging very slowly, and sometimes getting stuck in local minima, as stated by Svozil *et al.* (1997). However, improvements to the backpropagation training procedure have reduced this effect.

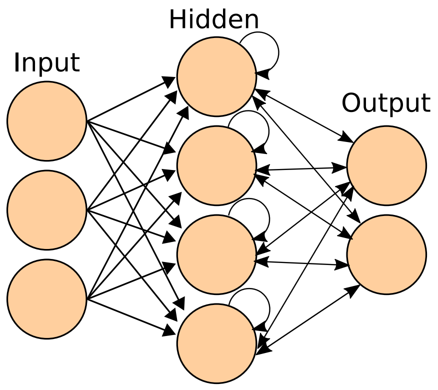


Figure 1.5: A diagram of a simple RNN. Adapted from Commons (2006).

Recurrent Neural Network

The recurrent neural network (RNN) is a family of neural networks. The nodes of an RNN makes provision for feedback between different hidden layers and the output layer, which creates an internal state for the network. See Figure 1.5 for a diagram of a RNN topography.

In contrast to the FFN, this internal saved state allows the RNN to process arbitrary sequences of input data, making it adept at processing unsegmented speech or handwriting patterns. However, the added complexity of adding feedback loops between the different layers become very expensive computationally, especially when large networks with many layers and inputs are used. Increases in computing power and improvements in the training process, as well as a better general understanding of RNN's and ANN's in general, have alleviated the computational expense somewhat.

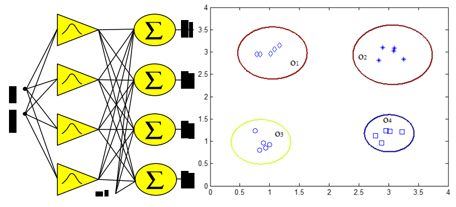
There are different RNN configurations available. These include the Hopfield network [Hopfield (1982)], the echo network [Jaeger (2001)] and the recurrent multilayer perceptron network [Tutschku (1995)].

Radial Basis Function Network

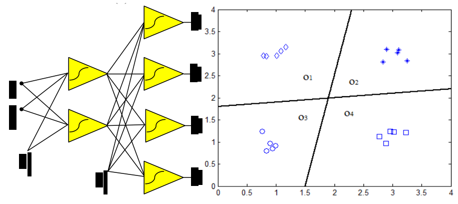
The radial basis function network (RBFN) is a type of ANN, which is a subfamily of the RNN family, as stated by Wilamowski and Jaeger (1996).

The RBFN topology is fixed to a three-layer architecture, with one input layer, one hidden layer and one output layer. The input layer provides the input. The hidden layer then remaps these inputs to make them linearly separable, where the output layer does the separation [Xie *et al.* (2011)].

Despite belonging to the same family of ANN, there are a number of significant differences between the RBFN and RNN. Firstly, the three-layer RBFN's are simpler than multi-layered RNN's, making the training process for RBFN's generally faster than that of an RNN. Secondly, and most importantly, as stated



(a)



(b)

Figure 1.6: A comparison between the RBF classifier in Figure 1.6a and the RNN classifier in Figure 1.6b as shown by Xie *et al.* (2011).

by Xie *et al.*, is the difference in how the RBFN and RNN classifies the data: the RBFN data clusters are separated by a hyper sphere, whereas the RNN's use arbitrarily shaped hyper surfaces. See Figure 1.6 for a demonstration of these class separation strategies.

This makes RBFN's an attractive option to interpolate multidimensional data. Xie *et al.* went further to determine that RBFN's are ideally suited to interpolate noisy data where the data surfaces contains regular valleys and peaks. In contrast, normal ANN's and RNN's are more effecient for classification problems and for well-conditioned, regularly spaced data.

As described by Skala (2012), the function on each node is given by Equation 1.5.

$$f(\boldsymbol{x_j}) = \sum_{j=1}^{N} \lambda_j \phi(||\boldsymbol{x}_i - \boldsymbol{x}_j||) \tag{1.5}$$

In Equation 1.5, λ_j is the node weighting factor which is determined during the model training phase. The function ϕ is the radial function which takes the euclidian norm of the distance between the node centre and the input vector as an input. This radial function is variable and the designer can select the function which best describes the data, though a Gaussian radial function, i.e. $\phi(r) = e^{-(\frac{r}{\epsilon})^2}$, is commonly used.

Support Vector Machines

Support vector machines (SVM) is a widely used classification technique first proposed by Cortes and Vapnik (1995). It works by finding a hyperplane

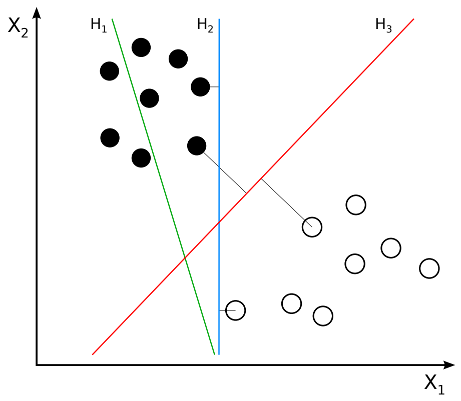


Figure 1.7: SVM plot illustrating different class separators [Commons (2012)].H_1 does not separate the classes, while H_2 has only minimal class separation. H_3 exhibits the widest separation margin.

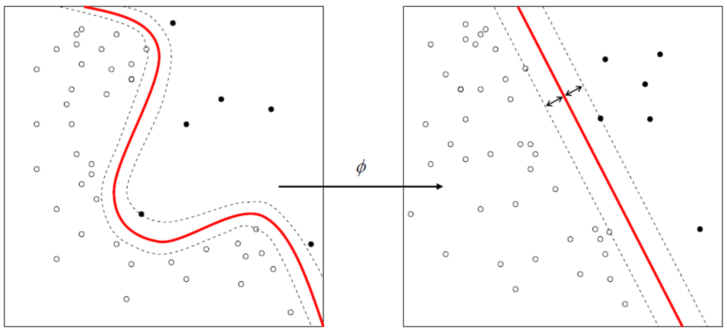


Figure 1.8: An example of an SVM with a non-linear kernel separator [Commons (2011)].

between two data classes that separates the classes by the widest possible average margin. See Figure 1.7 for an illustration of this.

Cortes and Vapnik's original method was limited to linear hyperplanes. Since then, the algorithm has extended to be non-linear hyperplanes by applying what is known as the 'kernel trick', as described by Amari and Wu (1999). An example of such a non-linear separator can be seen in Figure 1.8.

SVM's are also limited to binary, i.e. two class, problems, somewhat limiting their use for high-dimensional problems. However, they are extremely popular in scientific circles thanks to their accuracy and relative simplicity.

Bibliography

Amari, S.-i. and Wu, S. (1999). Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, vol. 12, no. 6, pp. 783–789.

Bouabdallah, S., Noth, A. and Siegwart, R. (2004). PID vs LQ control techniques applied to an indoor micro quadrotor. In: *International Conference on Intelligent Robots and Systems. Proceedings.*, vol. 3, pp. 2451–2456. IEEE.

Bradski, G. *et al.* (2000). The opencv library. *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120–126.

Commons, W. (2006). Artificial neural networks. `http://bit.ly/1IYexjC`.

Commons, W. (2011). Kernel machine. `http://bit.ly/1Pv8esq`.

Commons, W. (2012). Svm separating hyperplanes. `http://bit.ly/1UO98Di`.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, vol. 20, no. 3, pp. 273–297.

Fischler, M.A. and Bolles, R.C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, vol. 24, no. 6, pp. 381–395.

Gao, X.-S., Hou, X.-R., Tang, J. and Cheng, H.-F. (2003). Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 8, pp. 930–943.

Hamel, T., Mahony, R., Lozano, R. and Ostrowski, J. (2002). Dynamic modelling and configuration stabilization for an x4-flyer. *a a*, vol. 1, no. 2, p. 3.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In: *Alvey vision conference*, vol. 15, p. 50. Citeseer.

Heikkila, J. and Silvén, O. (1997). A four-step camera calibration procedure with implicit image correction. In: *Computer Society Conference on Computer Vision and Pattern Recognition. Proceedings.*, pp. 1106–1112. IEEE.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558.

Horaud, R., Conio, B., Leboulleux, O. and Lacolle, L.B. (1989). An analytic solution for the perspective 4-point problem. In: *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR'89., IEEE Computer Society Conference on*, pp. 500–507. IEEE.

Jaeger, H. (2001). The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34.

Lepetit, V., Moreno-Noguer, F. and Fua, P. (2009). Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, vol. 81, no. 2, pp. 155–166.

Levenberg, K. (1944). A method for the solution of certain non–linear problems in least squares. *Quarterly of Applied Mathemetics*, vol. 2, p. 164.

Lu, C.-P., Hager, G.D. and Mjolsness, E. (2000). Fast and globally convergent pose estimation from video images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 6, pp. 610–622.

Marquardt, D.W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441.

Melen, T. (1994). *Geometrical modelling and calibration of video cameras for underwater navigation.* Institutt for Teknisk Kybernetikk, Universitetet i Trondheim, Norges Tekniske Høgskole.

Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (2013). *Machine learning: An artificial intelligence approach.* Springer Science & Business Media.

Pounds, P., Mahony, R. and Corke, P. (2010). Modelling and control of a large quadrotor robot. *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699.

Raffo, G.V., Ortega, M.G. and Rubio, F.R. (2010). An integral predictive/nonlinear Hinf control structure for a quadrotor helicopter. *Automatica*, vol. 46, no. 1, pp. 29–39.

Schweighofer, G. and Pinz, A. (2006). Robust pose estimation from a planar target. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 12, pp. 2024–2030.

Skala, V. (2012). Radial basis functions for high dimensional visualization. *VisGra-ICONS 2012*, pp. 218–222.

Slama, C.C., Theurer, C., Henriksen, S.W. *et al.* (1980). *Manual of photogrammetry.* Ed. 4. American Society of photogrammetry.

Svozil, D., Kvasnicka, V. and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62.

Tu, J.V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, vol. 49, no. 11, pp. 1225–1231.

Turing, A.M. (1950). Computing machinery and intelligence. *Mind*, pp. 433–460.

Tutschku, K. (1995). Recurrent multilayer perceptrons for identification and control: The road to applications. *Univ. Würzburg, Germany, ser. Research Report Series.*

Wilamowski, B.M. and Jaeger, R.C. (1996). Implementation of rbf type networks by mlp networks. In: *Neural Networks, 1996., IEEE International Conference on*, vol. 3, pp. 1670–1675. IEEE.

Xie, T., Yu, H. and Wilamowski, B. (2011). Comparison between traditional neural networks and radial basis function networks. In: *IEEE International Symposium on Industrial Electronics.*