Support Vector Machine Algorithm applied to Industrial Robot Error Recovery

CIDNEY LAU
Support Vector Machine Algorithm applied to Industrial Robot Error Recovery

Titel svenska: Support Vector Machine algoritm tillämpad inom felhantering på industrirobotar

Författare: Cidney Lau
E-post: cidney@kth.se

Uppdragsgivare: ABB Shanghai Ltd.

Examinator: Prof. Anders Lansner
Handledare: Prof. Örjan Ekeberg
Datum: 2015-06-23

Civilingenjörsutbildning Elektroteknik, examensarbete inom Datalogi
Abstract (English)

A Machine Learning approach for error recovery in an industrial robot for the plastic mold industry is proposed in this master thesis project. The goal was to improve the present error recovery method by providing a learning algorithm to the system instead of using the traditional algorithm-based control. The chosen method was the Support Vector Machine (SVM) due to the robustness and the good generalization performance in real-world applications. Furthermore, SVM generates good classifiers even with a minimal number of training examples. In production, there will be no need for a human operator to train the SVM with hundreds or thousands of training examples to achieve good generalization. The advantage with SVM is that good accuracy can be achieved with only a couple of training examples if the training examples are well designed.

Firstly, the algorithm proposed was evaluated experimentally. The experiments consisted of correct handling of classification performance on training examples, which was a hand-coded data set created with defined in- and output signals. Secondly, the results from the experiments were tested in a simulated environment. By using only a few training examples the SVM reached perfect performance. In conclusion, SVM is a good tool for classification and a suitable method for error recovery on the industrial robot for the plastic mold industry.

Abstract (Swedish)


I projektet utvärderades SVM metoden experimentellt varefter den testades i ett simuleringsprogram. Resultatet visade att SVM metoden gav en perfekt precision med hjälp av endast ett fåtal träningsdata. En slutsats från denna studie är att SVM är en bra metod för klassificering och lämplig för felhantering på industrirobotar inom plastindustrin.
1 Introduction

The first industrial robot was constructed by Joseph Engelberger in 1961. Therefore, year 1961 can be considered as the beginning of industrial robotics [1]. The industrial robots have been developing and progressing ever since and robots have replaced the human in many automated manufacturing systems. Primarily in those systems that are dangerous to humans and harmful to the health. Productivity was increased when higher regularity and accuracy was introduced. For example, in the last 15–20 years, car manufacturing has been automated and fully robotized, starting from the initial stage of forging, through engine manufacture, to assembly of parts into the final product. The number of industrial robots is presently estimated to one million and one third of them are made in Japan. In the last decade we have witnessed a rapid development of robots and the development will continue to grow [2, 3, 4].

The industrial robot is based on a hand-coded traditional algorithm. Moreover, industrial robots are rarely associated with intelligence [5]. The reason is that the robot-based manufacturing system is inherently complex. The control and coordination for the industrial robots demand strict attention to detail and reliability [6].

However, robot-based manufacturing systems can today be logically correct but still fail under abnormal conditions. The traditional algorithm-based control of manufacturing systems has no problem dealing with well-ordered, highly structured environments. Moreover, they have also shown a good capability for handling most abnormal conditions with many details involved. But when it comes to dynamic systems or incorporating a 'higher level' understanding of what is desired of the system, the traditional algorithm-based control has limitations. It cannot ensure that the environment is predictable enough to function reliably [7, 8].

That is why many researchers attempt to address the limitations of traditional algorithm-based control through various Machine Learning techniques [7]. Machine Learning is a study of pattern recognition and computational learning theory. Machine Learning is making computers to act without strictly following static program instructions. It is more flexible than the expert system since the learning approach from training data makes links between different ideas and meanings. Machine Learning has the capacity to continuously self-improve and thereby offers increased efficiency and effectiveness. The learning approach can solve complex tasks in logistic applications. It is also concluded in related articles to be feasible for real-world applications [6, 9].

One goal of robotics research is to construct robust and reliable robot systems that are able to handle errors arising from abnormal operating conditions. Error handling is becoming increasingly important in the industrial robotics environments. The traditional algorithm-based controlled systems are inflexible and do not respond well to 'errors' that occur during production. When faced with an unexpected event or error condition, the system would shut down and start setting off alarms and lights. An 'operator' would then attend and diagnose what has failed and take corrective and appropriate action to the situation. Researchers have begun exploring the use of learning systems to incorporate automated diagnosis into today's systems and, in some cases, they recommend the corrective actions to be taken by the operator. However, as systems become more complex and more capable, operators have less experience dealing with errors and trouble-shooting. They become less qualified to take corrective actions. At an ultimate situation, operators become little more than people assigned to read the right manual when an error light turns on and then push the right button. It is at this point when the system itself should be given the responsibility and authority to take corrective action on its own. To obtain this ability, the systems must be provided with on-line automated diagnostics and error recovery capabilities [7, 10].
In computer systems, errors can be defined as either component errors or design errors, but in the robot assembly world, there is a third type: external errors. The external error problem in industrial robots has multiple causes of errors and the error situations in industrial robots are hard to predict in advance. These include external interference with processes, or components, and unexpected events such as breakages and jammed parts [11].

The field of error handling is often divided in three subfields:

- **Detection techniques** for (or the process of) observing the actual state of the controlled system and comparing it with specifications in order to find discrepancies as early as possible.

- **Diagnose techniques** for finding the original fault which caused the error.

- **Recovery applying** the proper corrective actions in order to prevent a possible future error or reach an error free state.

In each of these subfields there are several principles for how to achieve these objectives, as well as different methods for representing the information needed [12].

The most widely used technique for error recovery in the industrial robots is known as the backward error recovery. The backward error recovery finds a previous error free state of the system and returns there, and ‘undoes’ what has been done. However, despite the attractive simplicity of these methods, their fixed response often proves inappropriate in robotic applications. This is because backward recovery is inflexible and it assumes that processes are reversible and objects are recoverable, and that the system has full control over a well-defined environment. These assumptions do not hold in robotics [11, 12].

The international company ABB provides many kinds of industrial robots for today’s industries. When using the ABB robot e.g. machine tending, some error cases are complicated to handle. Currently, the error recovery for the ABB robots in machine tending, especially for plastic injection molding cannot be customized. The error recovery for the ABB plastic molding robots moves the robot back to home position (start position) safely and as soon as possible whenever error occurs. Presently, all errors are handled with the same rule.

According to the customers’ requests, this solution of the error recovery is not always enough. In the real world, customers need more flexibility and options to cover all possible situations. That is why an improvement is valuable for the existing error recovery method, but can this problem be solved with machine learning or other algorithms and approaches?

Many researchers have focused to implement different Machine Learning methods and Artificial Neural Network (ANN) methods into real-world applications and industrial robots. Support vector machines (SVM) which is a Machine Learning algorithm, and artificial neural networks (ANN) have been used in applications such as: bearing fault detection, breast cancer cell detection, drug classification, image retrieval, identifying students with learning disabilities, modeling a microwave transmitter, protein fold recognition, signature recognition and textile color classification. These investigations indicate that the SVM algorithm will generally perform better than the ANN. However, there are exceptions that occur when the user has specific knowledge of the application and the available data. Then SVM is outperformed by ANN [4].

SVM was introduced by Vapnik Chervonenkis in 1995 [3, 4, 13]. SVM seeks to determine a linear separator between binary data classes. The optimal position of the separating plane is specified as where the margin between the plane and the data points are maximized. This concept is generally referred to as a maximal margin classifier, and its strength lies in determining a good general solution to the classification problem without overfitting the data [2, 14].
Another article shows that Extreme Learning Machine (ELM) has poorer generalization ability than the SVM when the size of the training set is small, while it has the potential to yield generalization behavior as good as the SVM when the size of the training set becomes large. The possible reason is the presence of overfitting in the training process of the ELM. The ELM has also superior computational speed compared with the SVM, and this superiority will increase drastically as the size of training set grows. Furthermore, the SVM shows very strong learning ability, avoiding overfitting, as well as strong generalization performance compared to ELM [13].

The rapid development of SVM in statistical learning theory has encouraged researchers to apply SVM to various fields. It has been widely used for text classification, pattern recognition and fault detection, etc. [7, 15].

The main objective of this project was to explore an approach that is entirely based on Machine Learning methods to industrial robots in the hope to improve the production.

The goal of this research was to apply a Machine Learning technique in error handling for the ABB robots in the plastic molding industry and effectively train the robot to classify the most correct solutions according to the customer’s requirements. Furthermore, this learning approach was investigated and evaluated to find out if it’s suitable.

Outline

The background knowledge for the ABB robot that was used in this project is introduced in Chapter 2. In Chapter 3, a method is proposed with a justified reason. Furthermore, the basic concept of the background theory is explained and a detailed section of the implementation to the proposed method is given. Experiments and results achieved by using the Machine Learning technique, Support Vector Machine (SVM), are found in Chapter 4. Discussion and proposed tools for future production are presented in Chapter 5. Finally, the conclusions are drawn in Chapter 6.

Definitions and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>ELM</td>
<td>Extreme Learning Machine</td>
</tr>
<tr>
<td>ERM</td>
<td>Empirical Risk Minimization</td>
</tr>
<tr>
<td>KNN</td>
<td>K-nearest neighbors</td>
</tr>
<tr>
<td>RWPM</td>
<td>RobotWare Plastics Mould</td>
</tr>
<tr>
<td>SMO</td>
<td>Sequential Minimal Optimization</td>
</tr>
<tr>
<td>SRM</td>
<td>Structural Risk Minimization</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
</tbody>
</table>
2 Background ABB RobotWare Plastic Mould (RWPM)

ABB is a leading supplier of industrial robots, modular manufacturing systems and service. ABB has installed more than 250,000 robots worldwide.

The ABB RobotWare Plastic Mould (RWPM) is a robot software for machine tending, especially for injection molding. It provides an easy to use user-interface for programming and production, including safe Home Run, user authorization, production statistics, and event log as well as a standardized and structured way of machine tending programming.

The error recovery solution today is to make a safe Home Run when an error occurs. The secure and automatic Home Run system takes the robot safely from any situation where it has stopped without any need for the operator to jog the robot in difficult areas.

However, only the safe Home Run error recovery solution is not always enough, according to the customers. In real-world applications, customers need more options to cover all possible situations. Flexibility is a big issue here and that is why more solutions for the error handling are required.

In this project the ABB six-axis robot for RWPM was used to test the Machine Learning technique for improvement of the current error recovery system in industrial robots.

Figure 1. ABB six-axis robot for RobotWare Plastic Mould.
Description of the robot’s environment

Every station in RWPM, has its own set of paths, and it can have a maximum of six paths. A path consists of a maximum of ten points and each path has a number of move-in points and a number of move-out points. A path has an Entry point and a Target point. The Entry point is placed outside the machine; the main path is then used to move into the Target point inside the machine. From the Target point, there could be two sub paths, path one and path two, see Figure 2. A more detailed description of different definition in RWPM is explained below [16].

![Diagram of the robot’s environment](image)

Figure 2. Illustration of the different stations and the robot path.

Station definition

A station is the special work place where the robot picks up or drops off a part or does some kind of processing of the part. There are three kinds of stations that are used for this project.

- **Home Station**: A mandatory station that is the robot’s home position. In the home position the gripper tool is opened.
- **Scrap Station**: A mandatory station where robots leave all parts before it moves home during home run cycle. In the scrap station the gripper tool is opened.
- **User Station**: User station is any station, except Home and Scrap station, e.g. injection moulding machine or conveyer [16].
**Station Status definition**

Station status is dynamic and it changes during run time for the station. A station status is used during scheduling to determine if a station could be executed and which station that is to be executed. Three status kinds have been used for this project.

**Station OK:** (stnOK) Station is okay i.e. the robot can attend to that station.

**Station NOK:** (stnNOK) Station is not okay, the robot is in a failure state.

**Station Busy:** (stnBusy) Station is busy.

**Path Type definition**

A station has to have at least one path, the main path. Apart from the main path, a station can have zero up to five numbers of sub paths. There are four path types used for this project.

**Main Path:** Move into stations.

**Sub Path:** Path inside a station.

**Cell Path:** Cell path is the path between stations.

**Out Path:** Out path is when the robot is out of the defined path.

**Point Type definition**

A path is an array of ten points with different types. A path has an Entry-point and a Target-point. Points between the Entry-point and the Target-point are move-in points and points from Target-point to last point are move-out points. Three types are used for this project. See Figure 3.

**Target point:** The target point is the point between move in points and move out points.

**In point:** The in points are the points between the entry point and the target point.

**Out point:** The out points are the points after the target point.

![Diagram of paths and points in a station](image-url)

**Figure 3.** Illustration of the paths and points in a station. A station has always a main path and if necessary a path one and/or a path two.
**Tool Status definition**

The gripper is a tool on the robot in plastic mould that is used to pick up, leave and hold part. Tool Status describes the status of the gripper. There are three kinds of gripper status used in this project.

**Tool open:** The gripper is opened.

**Tool closed:** The gripper is closed.

**Tool NOK:** The gripper in a failure state.

**The IRC5 controller FlexPendant**

The FlexPendant is an operator panel in which all operations and programming can be carried out. The FlexPendant is equipped with a large touch screen and a color display, which displays different kinds of system information, pictures, graphs and user interactions activated by the touch of a finger. Limited programming or computer experience is required to learn how to use the FlexPendant, see Figure 4 [17].

![Figure 4. FlexPendant.](image)
3 Methods

The main objective of this project was to explore an approach that was based on Machine Learning for error recovery in industrial robots. The learning method was applied to the ABB robots in the plastic molding industry (RWPM) to effectively train the robot to classify the most correct solutions according to the customer’s requirements. But the question was which learning method that should be tested and whether it was suitable for industrial robots?

Different methods such as Machine Learning methods and ANN methods have been studied for this project. Some article describes that SVM outperform ANN, in terms of both classification accuracy and classification speed [4].

In another article, three kinds of classifier methods were compared. The first was the statistics-based method such as Bayesian method, K-nearest neighbors (KNN) method and SVM. The second was the rule-based method such as decision tree and rough sets. The last one was the ANN method. SVM algorithm is a solution to convex optimization problems, and it is often better than others because its local optimal solution is the global optimal solution [14].

Furthermore, classification needs to be performed in an unstructured environment full of uncertainties and SVM has been used in many real-world applications due to good generalization performance. SVM has also been proven to be a robust method [13].

Therefore, according to the reasons above, SVM was proposed as the learning method for error recovery in this project. The SVM was used to learn approximated classified solutions for the ABB robot error handling system and thereafter conclude if it was a possible and suitable solution.

SVM is a supervised training method that provides learning with a teacher. In other words, the output answer has been associated with the input during the training process.

The basic idea of SVM was to apply a nonlinear mapping \(\Phi\) to map the data of input space into a higher-dimensional feature space, and then implement the linear classification in this higher-dimensional space. SVM seeks to determine a linear separator between binary data classes. The optimal position of the separating plane is specified as where the margin between the plane and the data points is maximized. This concept is generally referred to as a maximal margin classifier, and its strength lies in determining a good general solution to the classification problem without overfitting the data [4].

In order to separate more complex data sets, a kernel function was used to map the data into a higher-dimensioned space where a single hyperplane could separate the binary classes. The kernels tested in this project were:

- Linear kernels
- Polynomial kernels

Multi-class classification was also needed for this project since we wanted more options to cover all possible situations in error recovery. Multi-class classification can be achieved using either a multi-class version of the SVM algorithm.
Limitations

For this project the SVM Machine Learning approach has only been applied to error recovery in the ABB industrial robots for plastic mould. The ABB robot software for machine tending, especially for injection molding is called the RobotWare Plastic Mould (RWPM). Only a selection of errors were handled in the project. The errors that were handled are so called plastic errors which currently use the same rules for recovery i.e. to do a safety Home Run whenever fault occurs.

Furthermore, this project was only evaluated for one set of hand-coded training rules, since the purpose of this project was to improve the error recovery with a Machine Learning approach. This means that one evaluated set of hand-coded training rules was enough. This set of hand-coded training rules were used in all the experiments.

To be able to understand the experiments of the project, Chapter 3.1 and 3.2 give a detailed background theory of the SVM algorithm and the implementation.

3.1 Background theory Support Vector Machine (SVM)

The Artificial Neural Networks algorithm (ANN) changes system parameters and weights during the learning process and these approaches are based on a risk function called Empirical Risk Minimization (ERM). In contrast, the Support vector Machine algorithm (SVM) also uses the ERM and in addition, a risk function known as Structural Risk Minimization (SRM). This has been shown to be superior than using ERM alone [18, 21].

The ERM is only based on minimizing the error of the training data itself. If the training data is sparse and/or not representative of the underlying distribution, then the system will be poorly trained, hence limited classification performance.

SVM is a Machine Learning approach which has drawn much attention because of the high classification performance. The SVM algorithm is a set of universal supervised, feed-forward network based classification algorithm based on the statistical learning theory and a SRM principle. For classification problems, the SVM maps the input vectors into a high dimensional nonlinear feature space and constructs a hyperplane, which linearly separates the data into different classes. This hyperplane is called the maximal margin hyperplane because it maximizes the distance to the closest points of the two classes. By appropriately defining a kernel function relating the data in the input space to the dot product between the data vectors in the feature space, a high dimensional feature space can be achieved.

This section 3.1 introduces the theory of SVM, including linearly separable case, linearly nonseparable case and nonlinear case through a two-class classification problem. A multi-class SVM classification problem is also introduced in this section [21].

3.1.1 Optimal hyperplane for the linearly separable patterns

In the linearly separable case, the training set $S$ is given as:

$$S = \{(x_i, y_i)\}_{i=1}^{N}$$

Where $x \in \mathbb{R}^N$ is the input pattern, with given labels $y \in \{-1, +1\}$. The linearly separable case can be separated by a hyperplane decision function. The separating hyperplane decision function is:
\[ f(x) = \mathbf{w} \cdot \mathbf{x} + b \]  

(2)

where \( \mathbf{x} \) is an input vector, \( \mathbf{w} \) is an adjustable weight vector and \( b \) is a bias.

The goal of SVM is to find an optimal hyperplane that satisfies:

\[ \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \quad \text{if} \quad y_i = +1 \]  

(3)

\[ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \quad \text{if} \quad y_i = -1, \; i = 1, \ldots, l \]  

(4)

[18].

In their simplest form, SVMs are hyperplanes that separate training data by a maximal margin. All vectors lying on one side of the hyperplane are labeled as -1, and all vectors lying on the other side are labeled as 1. The training data that lie closest to the hyperplane and determine the hyperplane are called support vectors, SV. All remaining examples of the training set are irrelevant.

Now the margin \( \rho \) has to be maximized of this hyperplane. The margin \( \rho \) is the minimal distance of the hyperplane to the closest data point. The problem is to find \( \mathbf{w} \) that maximizes the margin \( \rho \)

\[ \rho = \frac{y_i f(x_i)}{\|\mathbf{w}\|} \]  

(5)

[19].

An infinite number of solutions exists. In order to limit the number of solutions, the only solution that is considered is normalized as \( \rho \cdot \|\mathbf{w}\| = 1 \). The wider the margin \( \rho \) is, the smaller the VC-dimension of the resulting classifier. By widening the classification margin \( \rho \), the confidence interval for the classification error is reduced. Maximizing the margin \( \rho \) is equivalent to minimizing \( \|\mathbf{w}\| \).

Therefore, for the linearly separable case, to find the optimal hyperplane is to solve the following optimization problem:

\[ \text{Minimize:} \quad \frac{1}{2} \|\mathbf{w}\|^2 \]  

(6)

\[ \text{Subject to constraints:} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \; i = 1, 2, \ldots, N \]  

(7)

[20].

To solve the optimization problem, the method of Lagrange multipliers is used. If \( \lambda_i \geq 0 \) are the Lagrange multipliers, the optimization problem can be written as:

\[ L(\mathbf{w}, b, \lambda_i) = \frac{1}{2} (\mathbf{w} \cdot \mathbf{w}) - \sum_{i=1}^{N} \lambda_i \left[ y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right] \]  

(8)

The solution to the constrained optimization problem is to determine the saddle point of the Lagrangian function. The saddle point of this function \( L(\mathbf{w}, b, \lambda_i) \) is found, when it is minimized with respect to \( \mathbf{w} \) and \( b \) and maximized with respect to \( \lambda_i \geq 0 \). Thus, differentiating \( L(\mathbf{w}, b, \lambda_i) \) with respect to \( \mathbf{w} \) and \( b \) and setting the result equal to zero, the following two conditions of optimality are achieved:
Condition 1: \[ \frac{\partial L(w, b, \lambda_i)}{\partial w} = 0 \] (9)

Condition 2: \[ \frac{\partial L(w, b, \lambda_i)}{\partial b} = 0 \] (10)

Solving these two partial derivatives conditions gives:

\[ w = \sum_{i=1}^{N} \lambda_i y_i x_i \] (11)

This condition must be satisfied by the Lagrange multipliers. The following condition shows that the vector \( w \) is a linear combination of the training vectors:

\[ \sum_{i=1}^{N} \lambda_i y_i = 0 \] (12)

It is important to note that at the saddle point, for each Lagrange multiplier \( \lambda_i \), the product of that multiplier with its corresponding constraint vanishes, as shown by:

\[ \lambda_i [y_i (w \cdot x_i + b) - 1] = 0, \quad i = 1, 2, \ldots, N \] (13)

Therefore, only those multipliers exactly meeting Eq. (13) can assume nonzero values. These \( x_i \) for which \( \lambda_i > 0 \) are the support vectors (SV).

The dual formulation of the optimized problem, where the Lagrangian is expressed in function of the \( \lambda_i \) only, is translated as:

Maximize: \[ L_{dual}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \] (14)

Subject to constraints: \[ \sum_{i=1}^{N} \lambda_i y_i = 0, \text{ with } \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \] (15)

Finally, the separation function is:

\[ f(x) = \sum_{i=1}^{N} \lambda_i^* y_i (x \cdot x_i) + b \] (16)

Note that the dual problem is cast entirely in terms of the training data. Furthermore, the dual function to be maximized depends only on the input patterns in the form of a set of dot products, \( \{x_i \cdot x_j\}_{i,j=1}^{N} \). When the optimum Lagrange multipliers \( \lambda_{0,i} \) is determined, the optimum weight vector \( w_0 \) may be computed [18, 19].

\[ w_0 = \sum_{i=1}^{N} \lambda_{0,i} y_i x_i \] (17)
3.1.2 Optimal hyperplane for the linearly nonseparable patterns

Linear separability cannot always be assumed. If the inequalities in Eq. (3) and (4), do not hold for some data points in \( S \), the SVM becomes linearly nonseparable. When the training data isn’t linearly separable, the slack variables \( \xi_i \) can be added to the optimization problem into the decision surface.

\[
y_i \left( w^T x_i + b \right) \geq 1 - \xi_i, \quad i = 1, 2, ..., N
\]  

(18)

For \( 0 \leq \xi_i < 1 \), the data point fall inside the region of separation but on the right side of the decision surface. For \( \xi_i > 1 \), they fall on the wrong side of the decision surface. Now the goal of the SVM is to find a separation hyperplane for which the misclassification error can be minimized while maximizing the soft margin of separation. The SVs lie exactly on the edge of the margin when the soft margin is not used and on the edge or in the margin area in soft margin classification. To find an optimal hyperplane for a linearly nonseparable case, the following constrained problem needs to be solved:

\[
\begin{align*}
\text{Minimize:} & \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i \\
\text{Subject to constraints:} & \quad y_i \left( w^T x_i + b \right) \geq 1 - \xi_i, \quad i = 1, 2, ..., N \\
& \quad \xi_i \geq 0, \quad i = 1, 2, ..., N
\end{align*}
\]

(19) (20)

As before, minimizing the first term in Eq. (19) is related to minimizing the VC dimension of the SVM. The second term is an upper bound on the number of test errors, where \( C \) is a user-defined positive parameter. It controls the trade-off between complexity of the machine and the number of nonseparable points. In particular, this is the only free parameter in SVM [18, 19, 20].
One can also translate this optimization problem into a dual form. The dual Lagrangian that has to be maximized is: [19]

Maximize: 

\[ L_{\text{dual}}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j) \] (21)

Subject to constraints: 

\[ \sum_{i=1}^{N} \lambda_i y_i = 0, \text{ with } 0 \leq \lambda_i \leq C, \quad i = 1, 2, ..., N \] (22)

Note that neither the slack variables nor the Lagrange multipliers appear in the dual form. The dual problem for the nonseparable case is similar to the linear separable case except for a minor difference. The dual function is to be maximized in both cases thus the nonseparable case differs in that the constraint is replaced with the more stringent constraint \( 0 \leq \lambda_i \leq C \). Except for this modification, the constrained optimization for the nonseparable case and computations of the optimum values of the weight vector \( \mathbf{w} \) and bias \( b \) proceed in the same way as in the linearly separable case [18].

### 3.1.3 Optimal hyperplane for the nonlinear patterns

In most real life problems, the input data isn’t linearly separable. To solve the nonlinear case, the input vector is mapped/transformed into a higher dimensional feature space, where the training data will be linearly separable.

![Figure 6. Nonlinear SVM.](image)

Picture an input space made up of nonlinearly separable patterns. Cover’s theorem states that such a multidimensional space may be transformed into a new feature space where the patterns are linearly separable with high probability, provided that two conditions are satisfied. Firstly, the transformation is nonlinear. Secondly, the dimensionality of the feature space is high enough. Note that Cover’s theorem does not discuss the optimality of the separating hyperplane. It is only by using an optimal separating hyperplane that the VC dimension is minimized and generalization is achieved [18].

Let \( \mathbf{x} \) denote a vector drawn from the input space with dimension \( m_0 \). Let \( \{ \varphi_j(\mathbf{x}) \}_{j=1}^{m_1} \) denote a set of nonlinear transformations for the input space to the feature space with the dimension \( m_1 \) of the feature space. \( K(\mathbf{x}, \mathbf{x}_i) \) is the inner-product kernel that implicitly maps data from input space into a higher dimensional feature space via a nonlinear kernel function:

\[ K(\mathbf{x}, \mathbf{x}_i) = K(x, x_i) = \varphi^T(x) \varphi(x_i) = \sum_{j=0}^{m_1} \varphi_j(x) \varphi_j(x_i) \] (23)

for \( i = 1, 2, ..., N \)
The dual Lagrangians needs to be maximized for the separable case in the high-dimensional feature space. Therefore, the dual optimization problem for optimal hyperplane is now defined by:

Maximize: 
\[ L_{\text{dual}}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \] (24)

Subject to constraints: 
\[ \sum_{i=1}^{N} \lambda_i y_i , \text{ with } \lambda_i \geq 0 , \ i = 1, 2, ..., N \] (25)

The dual optimization problem with the slack variables has the following function that needs to be optimized:

Maximize: 
\[ L_{\text{dual}}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \] (26)

Subject to constraints: 
\[ \sum_{i=1}^{N} \lambda_i y_i , \text{ with } 0 \leq \lambda_i \leq C , \ i = 1, 2, ..., N \] (27)

The requirement on the kernel \( K(x, x_i) \) is to satisfy the Mercer’s theorem:

- \( K \) must be symmetric, and
- \( K \) must be positive definite.

Note that there is no need to use or know the form of \( \varphi \), since the mapping is never performed explicitly. Only the inner products of the data have to be calculated, while the explicit form of the data can stay implicit. Therefore, SVM can computationally afford to work in implicitly very large feature space, e.g. \( 10^{10} \)-dimensional. SVM can also control and avoid the overfitting issue by controlling the capacity and maximizing the margin \( \rho \).

### 3.1.4 Inner-Product Kernel

The SVM algorithm can construct a variety of learning machines by using different kernel functions. In Table 2, a summary of the inner-product kernel for four common types of SVMs are presented [18].

<table>
<thead>
<tr>
<th>Type of Support Vector Machine</th>
<th>Inner-product kernel ( K(x, x_i) ), ( i = 1, 2, ..., N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Kernel</td>
<td>( x^T x_i )</td>
</tr>
<tr>
<td>Polynomial learning machine</td>
<td>( (\gamma x^T x_i + \kappa)^p )</td>
</tr>
<tr>
<td>Radial-basis function network (RBF)</td>
<td>( \exp \left( -\frac{1}{2\sigma^2} |x - x_i|^2 \right) )</td>
</tr>
<tr>
<td>Sigmoid kernel (Two-layer perceptron)</td>
<td>( \tanh(\beta_0 x^T x_i + \beta_i) )</td>
</tr>
</tbody>
</table>

Table 1. Inner-Product Kernel.
For the polynomial kernel the obtained decision function has the following form:

\[
 f(x) = \text{sign}
 \left( \sum_{s\text{upportvectors}} y_i \lambda_i (x_i \cdot x + 1)^p - b \right)
\]  

(28)

The Radial Basis Function networks (RBF) has the obtained decision functions:

\[
 f(x) = \text{sign}
 \left( \sum_{s\text{upportvectors}} \lambda_i K_i (|x_i - x|) - b \right)
\]

(29)

where \( K_i (|x_i - x|) = \exp \left( - \gamma \|x - x_i\|^2 \right) = \exp \left( - \frac{1}{2\sigma^2}\|x - x_i\|^2 \right) \). The number of support vectors that are obtained with SVM, corresponds to the number of RBF kernels that is used to build up this network.

The sigmoid kernel is a similar approach to the two-layer neural networks with the separating function as follows:

\[
 f(x) = \text{sign}
 \left( \sum_{s\text{upportvectors}} \lambda_i \tanh(\beta_i x^T x_i + \beta_i) - b \right)
\]

(30)

Similar as the RBF kernels, the number of support vectors corresponds to the number of first-layer neurons that the network requires for optimal generalizations. The weights of the first-layer neurons are the support vectors \( x_i \), while the weights of the second-layer of neurons are given by \( \lambda_i \).

The linear kernel is able to work directly on the input space without mapping the data from input space into a higher dimensional feature space.

### 3.1.5 Multi-class SVM classification problem

Multi-class classification, where the number of classes \( N \) is larger than two, can be obtained by combining two-class SVM classification. Three different approaches are presented below.

**One-against-all** or **one-against-\( k \)** is an approach where \( k \) is the number of classes. In this method, \( k \) different classifiers are constructed, one for each class. By training \( k - 1 \) two-class classifier, each machine is trained as a classifier for one class against all other classes.

**One-against-one** is an approach that constructs a multi-class classifier. A \( \binom{N}{2} \) two-class machines separates each pair of classes, together with a majority voting scheme to estimate the final classification. Hence, the class with the maximal number of votes among all classifiers is the estimation.

Finally, one can solve a multi-class SVM classification directly. All the \( N \) classes are considered at once by modifying the optimization problem and optimizing the margins of \( N \) hyperplanes at the same time [19].
## 3.2 Implementations

In all problems, a quadratic problem has to be solved that guarantees a unique maximum. To solve this problem numerically you need to perform a gradient ascent, also known as steepest ascent [19]. We start from an initial estimate of the solution \( \lambda^0 \), which we then iteratively update by following the steepest ascending path. The length of the update is the learning rate \( \eta \) of the algorithm. The learning rate should be chosen carefully. If it is too large, it will not converge, but if it is very small, convergence of the algorithm will be very slow. For every training example the following individual learning rate \( \eta_i \) is proposed:

\[
\eta_i = \frac{1}{K(x_i, x_i)}
\]

(31)

This proposed learning rate \( \eta_i \) is proven to satisfy the sufficient condition for convergence.

There are a number of practical problems that arise when the learning problem becomes larger i.e. when the number of training examples increases substantially. Essentially the complexity of the optimization problem grows with the size of the matrix of Kernel values \( K(x, x_i) \), which grows quadratically with the number of training examples \( N \).

A number of solutions have been proposed to address this issue. Two solutions are especially mentioned that only require a part of the data that is applied to the optimizer at the same time.

- **Chunking**: Starting with a random subset (“chunk”) of training examples, the optimizer will run on this subset to find the initial Support Vectors (SV). This initial solution is consecutively used in order to find these training examples that most violate this solution. The latter examples and the current Support Vectors form together a new subset. This subset is used to run the optimizer again. This procedure is repeated until some stopping criterion is met.

- **Decomposition**: Decomposition methods are currently one of the major methods for training support vector machines. In the previous method, the size of the subset must be larger than the number of Support Vectors, which for large problems can still be a problem. This decomposition method fixes the subset size (which can be very small) and will only run the optimizer on different small subsets of the problem at a time [21].

### 3.2.1 Solving the Quadratic Problems

The difficulty of solving

\[
\min_\lambda \frac{1}{2} \lambda^T Q \lambda - e^T \lambda \\
0 \leq \lambda_i \leq C, \quad i = 1, \ldots, N
\]

(32)

\[
Subject to constraints: \quad y^T \lambda = 0
\]

(33)

is the density of \( Q \) because \( Q_{ij} \) is in general not zero. \( e \) is the vector of all ones, \( C > 0 \) is the upper bound, \( Q \) is a positive semidefinite matrix, \( Q_{ij} \equiv y_i y_j K(x_i, x_j) \), and \( K(x_i, x_j) = \phi^T(x_i) \phi(x_j) \) is the kernel. The decomposition method is used to conquer this difficulty. This method modifies a subset of \( \lambda \) per iteration. This subset, denoted as the working set \( B \), leads to a small sub-problem to be minimized in each iteration. An extreme case
is the Sequential Minimal Optimization (SMO), which restricts B to have only two elements. Then in each iteration, one solves a simple two-variable problem without needing optimization software.

**Sequential Minimal Optimization (SMO)**

This method is basically the decomposition method pushed to the limit. Using a subset size of two, only two points are considered for optimization at the same time. In this case the optimization problem can be solved analytically and hence no iterative quadratic optimization program is required.

Concretely, the optimization requires that:

\[
\sum_{i=0}^{N} \lambda_i y_i = 0
\]  

Let’s denote two multipliers \(\lambda_1\) and \(\lambda_2\), hence the minimum number of multipliers that can be modified at the same time is 2. Given the Eq. (34) this constraint must satisfy:

\[
\lambda_1 y_1 + \lambda_2 y_2 = \text{const} = \lambda_1^{\text{old}} y_1 + \lambda_2^{\text{old}} y_2
\]  

while \(0 \leq \lambda_1, \lambda_2 \leq C\). If we calculate \(\lambda_2^{\text{new}}\) first, the later implies that \(\lambda_2^{\text{new}}\) has to satisfy:

\[
U \leq \lambda_2^{\text{new}} \leq V
\]

with, for the case \(y_1 \neq y_2\):

\[
U = \max(0, \lambda_2^{\text{old}} - \lambda_1^{\text{old}})
\]

\[
V = \min(C, C - \lambda_2^{\text{old}} + \lambda_1^{\text{old}})
\]

While, for the case \(y_1 = y_2\):

\[
U = \max(0, \lambda_2^{\text{old}} + \lambda_1^{\text{old}} - C)
\]

\[
V = \min(C, \lambda_2^{\text{old}} + \lambda_1^{\text{old}})
\]

Now one can define the basic quantities of the algorithm. The classification error (as a SV) that a training example generates with the current set of multipliers is:

\[
E_i = f(x_i) - y_i = \left(f(x) = \sum_{j=0}^{N} \lambda_j y_j K(x_j, x_i) + b\right) - y_i,
\]

for \(i = 1, 2, \ldots, N\)

And the second derivative of the objective function along the diagonal line is:

\[
\kappa = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2) = \|\phi(x_1) - \phi(x_2)\|^2
\]

Using these quantities, the updates are given by the pairs of Lagrange multipliers, which will maximize the objective function when only these pair of multipliers are allowed to change:

\[
\lambda_2^{\text{new, unclipped}} = \lambda_2^{\text{old}} + \frac{y_2(E1 - E2)}{\kappa}
\]
This is clipped to enforce the constraint of Eq. (36):

\[
\lambda_2^\text{new} = \begin{cases} 
V & \lambda_2^\text{new,uncropped} > V \\
\lambda_2^\text{new,uncropped} & U \leq \lambda_2^\text{new,uncropped} \leq V \\
U & \lambda_2^\text{new,uncropped} < U 
\end{cases} \tag{42}
\]

And the value of \(\lambda_1^\text{new}\) is given by:

\[
\lambda_1^\text{new} = \lambda_1^\text{old} + y_i \left( \lambda_2^\text{old} - \lambda_2^\text{new} \right) \tag{43}
\]

**The Calculation of \(b\)**

After the solution \(\lambda\) of the dual optimization problem is obtained, the variable \(b\) must be calculated when used in the decision function.

Consider the case of \(y_i = 1\). If there is \(\lambda_i\) which satisfies \(0 < \lambda_i < C\), then \(r_i = \nabla f(\lambda)_i\). To avoid numerical errors, we average them:

\[
r_i = \frac{\sum_{0 < \lambda_i < C, y_i = 1} \nabla f(\lambda)_i}{\sum_{0 < \lambda_i < C, y_i = 1} 1} \tag{44}
\]

On the other hand, if there is no such \(\lambda_i\), as \(r_i\) must satisfy

\[
\max_{\lambda_i = C, y_i = 1} \nabla f(\lambda)_i \leq r_i \leq \max_{\lambda_i = 0, y_i = 1} \nabla f(\lambda)_i \tag{45}
\]

We take \(r_i\), as the midpoint of the range. For \(y_i = -1\), we can calculate \(r_2\) in a similar way. After \(r_1\) and \(r_2\) are obtained, the calculation of \(b\) will be:

\[
b = \frac{r_1 - r_2}{2} \tag{46}
\]

**Multi-class classification**

The “one-against-one” approach has been used. The approach in which \(k(k-1)/2\) classifiers are constructed and each approach trains data from two different classes. For training data from the \(i\):th and the \(j\):th classes, the following binary classification problem is solved:

\[
\min_{w^i, b^i, \xi^i} \frac{1}{2} (w^i)^T w^i + C \left( \sum_i (\xi^i) \right) \tag{47}
\]

Subject to:

\[
\left( [ (w^i)^T \phi(x_i) ] + b^i \right) \geq 1 - \xi^i, \text{ if } x_i \text{ in the } i\text{'th class}, \tag{48}
\]

\[
\left( [ (w^j)^T \phi(x_i) ] + b^j \right) \geq 1 - \xi^j, \text{ if } x_i \text{ in the } j\text{'th class}, \tag{49}
\]

\[
\xi^i \geq 0. \tag{50}
\]

In classification we use a voting strategy. Each binary classification is considered to be a vote where votes can be cast for all data points \(x\) – in the end a point is designated to be in a class with maximum number of votes. In case those two classes have identical votes, although it may not be a good strategy, we simply select the one with the smallest index [22].
3.3 Training sets

The SVM optimal hyperplane for nonlinear patterns algorithm is selected and used in the experiments. But before we use the algorithm, we needed to define the hand-coded data set with both inputs and outputs.

There is a specific vector format used for the SVM algorithm called the LIBSVM [22]. In Figure 7, the vector format for the LIBSVM is shown.

\[
\text{[label]} \text{[index1]:[value1]} \text{[index2]:[value2]} \ldots \text{[indexN]} \\
\text{[label]} \text{[index1]:[value1]} \text{[index2]:[value2]} \ldots
\]

Figure 7. Illustration of the SVM vector format.

The ordered index will be the different features or attributes with a respectively value. Features or attributes show the characteristic hold by the input vectors. The number of features also stands for the number of dimensions.

To give a deeper understanding of the vector format, an example is needed. Let us assume that two different points lie in a 2-dimensional plane with the coordinates and outputs shown in Table 2.

<table>
<thead>
<tr>
<th>Output</th>
<th>x-axis</th>
<th>y-axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2. Illustration of the output and input feature x- and y-axis with respectively values.

With the assuming data, the SVM vector format will have the following structure:

1 1:0 2:3
2 1:5 2:8

According to this structure, the two features or attributes (X and Y) are described. The index1 in this case has the feature or attribute X and index2 has the feature or attribute Y. The output in the table stands for the format label.

3.3.1 Output data

The error recovery in RWPM today moves the robot to home position safely and as soon as possible when an error occurs. The plastic errors are all defined errors for RWPM, for example when the machine is stopped because of ejectors are stocked. Currently, RWPM handles all errors with the same rule, which is to do a safety Home Run whenever fault occurs. There are two ways to handle the Home Run recovery:

1. If robot does not hold a part, move robot to home position and stop.
2. If robot holds a part when an error occurs, at fault occurrence, scrap part at Scrap station and then move the robot to home position [16].
According to the customers’ requests, only two solutions of the error recovery are not always enough. In the real world, customers need more options to cover all possible situations. Flexibility is a big issue here and that is why more solutions for the error handling are required. For this project, additional two solutions are presented according to the customers’ demands. The first solution is that the robot drops the part at the instant place before it returns to home position. The second solution is to stop the robot and let the operators jog the robot to the appropriate position.

The output for this project describes the different output cases for the classification. The defined output cases should be according to the customer’s needs. The SVM algorithm is constructed in such way that it allows the customer to add and change the output definition anytime.

The four customized new defined outputs that were the labels for this project are:

1. **Home Run**
   Move robot to home position and stop

2. **Scrap – Home Run**
   Scrap part at Scrap station and then move the robot to home position

3. **Drop – Home Run**
   Drop part at instant place and then move the robot to home position

4. **Stop – Jog**
   Stop and let the operators jog the robot

### 3.3.2 Input data

Different inputs are required to be able to use and classify the SVM algorithm. For the error recovery using the SVM algorithm, only five inputs were defined, i.e. Active Station, Station Status, Path Type, Point Type and Tool Status. In real production more inputs might be needed. However, the aim of this research was to try to adapt a learning theory to the industrial robots, therefore five inputs were enough to prove the principle. The inputs have been carefully chosen to solve this classification problem. The most relevant inputs are shown in Table 3, since the most important information was to find out the current position and status of the robot when an error occurred, and act to the defined solutions accordingly.

<table>
<thead>
<tr>
<th>1. Active Station</th>
<th>2. Station Status</th>
<th>3. Path Type</th>
<th>4. Point Type</th>
<th>5. Tool Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1 0 0</td>
<td>stnNOK</td>
<td>Target</td>
<td>tlOpen</td>
</tr>
<tr>
<td>Home</td>
<td>0 1 0</td>
<td>stnOK</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>Scrap</td>
<td>0 0 1</td>
<td>stnBusy</td>
<td>0 1 0</td>
<td>tlClosed</td>
</tr>
<tr>
<td>User</td>
<td>0 0 0 1</td>
<td></td>
<td>0 0 1</td>
<td>tlNOK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Active Station</th>
<th>Station Status</th>
<th>Path Type</th>
<th>Point Type</th>
<th>Tool Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1 0 0</td>
<td>Cellpath</td>
<td>Target</td>
<td>tlOpen</td>
</tr>
<tr>
<td>Home</td>
<td>0 1 0</td>
<td>Mainpath</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>Scrap</td>
<td>0 0 1</td>
<td>Subpath</td>
<td>0 1 0</td>
<td>tlClosed</td>
</tr>
<tr>
<td>User</td>
<td>0 0 0 1</td>
<td>Outpath</td>
<td>0 0 1</td>
<td>tlNOK</td>
</tr>
</tbody>
</table>

4 features x 3 features x 4 features x 3 features x 3 features = 432 combinations

Table 3. Illustration of the defined inputs for this project and the scaling method.
Currently, the network only supports numerical data. Non-numerical data has to be changed into numerical data. One way to do this is to use several binary attributes to represent a categorical attribute. That is why every input in this project was divided to three or four features/attributes. The Active Station has 4 features, Station Status has 3 features, Path Type has 4 features, Point Type has 3 features and Tool Status has 3 features. The multiplied features gave:

$$4 \text{ features} \times 3 \text{ features} \times 4 \text{ features} \times 3 \text{ features} \times 3 \text{ features} = 432$$

The possibility of the entire input vector was calculated to 432 combinations, see Table 3. In this project, 17 features were pre-processed, which means the input space holds the number of 17 dimensions. To speed up the solving SVM algorithm a scaling method was proposed. Another advantage of the method was to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, linearly scaling each attribute to the range $[0, 1]$ is recommended. The binary scaling method is also shown in the Table 3 and an example of the input vector format is:

$$[1 1:0 2:0 3:0 4:1 5:0 6:1 7:0 8:0 9:1 10:0 11:0 12:0 13:0 14:1 15:1 16:0 17:0]$$

Active Station = User
Station Status = stnOK
Path Type = Mainpath
Point Type = Outpoint
Tool Status = tlOpen

Feature 1-4 belongs to the input Active Station and feature 5-7 belongs to Station Status etc.

In order to check the output classified answer, the training example vector format has to be exposed with the right output answer. Furthermore, in order to construct good classifier by training, the training data must be from the same source as the unseen test data.

### 3.3.3 Applied Rules to the error recovery

In present error recovery, the same rule was applied to all faults:

1. If robot does not hold part, move robot to home position and stop
2. If robot holds a part at fault occurrence, scrap part and then move to home position

Since a new idea was proposed with four defined solutions, new rules for the proposed improvements had to be created.

The four new customized outputs:

1. Home Run $\Rightarrow$ HR
2. Scrap – Home Run $\Rightarrow$ Scrap
3. Drop – Home Run $\Rightarrow$ Drop
4. Stop – Jog $\Rightarrow$ Jog

The new rules were created manually based on input from interviewing experienced robot engineers and knowledge from RobotWare Plastic Mould, RWPM. For the SVM algorithm, 13 rules are presented in the Table 4. The rules were created thoughtfully and cover at least one example from every customized output.

The definition of the rules affects the SVM algorithm directly.
<table>
<thead>
<tr>
<th># Rules</th>
<th>Output</th>
<th>Active Station</th>
<th>Station Status</th>
<th>Path Type</th>
<th>Point Type</th>
<th>Tool Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jog</td>
<td>Any</td>
<td>Any</td>
<td>Outpath</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>2</td>
<td>Drop</td>
<td>None</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>tNOK</td>
</tr>
<tr>
<td>3</td>
<td>Drop</td>
<td>Home</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>tNOK</td>
</tr>
<tr>
<td>4</td>
<td>Drop</td>
<td>User</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>tNOK</td>
</tr>
<tr>
<td>5</td>
<td>Scrap</td>
<td>Scrap</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>tNOK</td>
</tr>
<tr>
<td>6</td>
<td>Scrap</td>
<td>Any</td>
<td>stNOK</td>
<td>Any</td>
<td>Any</td>
<td>tClosed</td>
</tr>
<tr>
<td>7</td>
<td>HR</td>
<td>Any</td>
<td>stNOK</td>
<td>Any</td>
<td>Any</td>
<td>tOpen</td>
</tr>
<tr>
<td>8</td>
<td>Scrap</td>
<td>Any</td>
<td>stBusy</td>
<td>Any</td>
<td>Any</td>
<td>tClosed</td>
</tr>
<tr>
<td>9</td>
<td>HR</td>
<td>Any</td>
<td>stBusy</td>
<td>Any</td>
<td>Any</td>
<td>tOpen</td>
</tr>
<tr>
<td>10</td>
<td>HR</td>
<td>Any</td>
<td>stOK</td>
<td>Any</td>
<td>Target</td>
<td>Any</td>
</tr>
<tr>
<td>11</td>
<td>HR</td>
<td>Any</td>
<td>stOK</td>
<td>Any</td>
<td>In point</td>
<td>Any</td>
</tr>
<tr>
<td>12</td>
<td>Scrap</td>
<td>Any</td>
<td>stOK</td>
<td>Any</td>
<td>Out point</td>
<td>tClosed</td>
</tr>
<tr>
<td>13</td>
<td>HR</td>
<td>Any</td>
<td>stOK</td>
<td>Any</td>
<td>Out point</td>
<td>tClosed</td>
</tr>
</tbody>
</table>

Table 4. Illustration of the 13 rules for the proposed improvement.

The number position for all the rules was ordered by priority. The first rule had the highest priority. The higher the priority the higher impact of the SVM algorithm. ‘Any’ in the table means that it can be any value. The 13 designed rules above were first of all used for research to see if the SVM algorithm was a good tool for error recovery on the RWPM. The rules were only used to generate one set of test data. It should be done differently for different cases. To test in real environment more realistic rules are required. Note that these rules will be different depending on the customers’ environment.

**Training set (TR)**

The training set is training examples which the system uses to create an optimal model for the classification by choosing from the whole combination of 432. Once the system is trained it will not be necessary to retrain the algorithm again. The system allows one to add new training examples whenever it is needed to aim for better results.

**Testing set (TE)**

The testing set consists of examples that are used to measure the accuracy of the classification. All the input vectors in combination, without the training set, were used as testing set to test results.
4 Experiments and Results

Applying SVM to real-world applications, the selection of a suitable kernel functions is important to get good classifiers. The aim of the experiments was to find the optimal algorithm to select the optimal model to fit the RWPM error recovery. This section puts particular emphasis on comparing different models of SVM obtained by choosing different kernels. In the following two kernels;

- the linear kernel and
- the polynomial kernel

were chosen to examine the capacity of SVM.

4.1 Linear kernel

The advantage with the linear kernel is that only one parameter is needed since the network solves linear and nonlinear SVM in the same way, together with a slack variable. This facilitates the operators to use the system. The parameter to be concerned of is the parameter $C$. Parameter $C$ is a trade-off between training errors and the flatness of the solution. The larger the $C$ is the less the final training error will be. But if $C$ increases too much the risk of losing generalization properties of the classifier will be bigger, because it will try to fit as best as possible to all the training examples including the possible errors of the dataset. In addition a large $C$ also increases the time needed for training. If parameter $C$ is small, then the classifier is flat meaning that the derivatives are small. The goal is to find a $C$ that keeps the training errors small but also generalizes well.

Different values of the parameter $C$ have been tested and a good value for this experiment has been chosen to $C = 10$ for the linear kernel. 13 random selected training examples, one from each of the predefined rules, were trained and tested with the whole combination set excluding the random selected training examples. After training with only 13 random selected training examples, one example from each rule, the predicted accuracy was estimated to 70.41 %. Hence the training examples were randomly selected; different rules are likely to appear differently. To see how the algorithm was behaving, 13 new random selected examples were added for each time, see Table 5.

<table>
<thead>
<tr>
<th># Cases</th>
<th># TR</th>
<th># SV</th>
<th># TE</th>
<th>Accuracy</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>13</td>
<td>419</td>
<td>295 / 419</td>
<td>70.41 %</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>22</td>
<td>406</td>
<td>316 / 406</td>
<td>77.83 %</td>
</tr>
<tr>
<td>3</td>
<td>39</td>
<td>33</td>
<td>393</td>
<td>320 / 393</td>
<td>81.43 %</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>41</td>
<td>380</td>
<td>361 / 380</td>
<td>95 %</td>
</tr>
<tr>
<td>5</td>
<td>65</td>
<td>53</td>
<td>367</td>
<td>353 / 367</td>
<td>96.19 %</td>
</tr>
<tr>
<td>6</td>
<td>78</td>
<td>66</td>
<td>354</td>
<td>351 / 354</td>
<td>99.15 %</td>
</tr>
<tr>
<td>7</td>
<td>91</td>
<td>73</td>
<td>341</td>
<td>341 / 341</td>
<td>100 %</td>
</tr>
<tr>
<td>8</td>
<td>104</td>
<td>81</td>
<td>328</td>
<td>328 / 328</td>
<td>100 %</td>
</tr>
<tr>
<td>9</td>
<td>117</td>
<td>94</td>
<td>315</td>
<td>315 / 315</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Table 5. Illustration of the result by using linear kernel.
According to Table 5, one can see that after giving 91 (13x7) training examples the SVM algorithm will have an accuracy of 100 %. Not all training examples are important for the algorithm. Only 73 training examples were selected for the training and the selected training examples are called support vectors, SV.

As described before, the optimizer will run on this subset to find the initial SV. This initial solution was used to find these training examples that were most violate for this solution. The latter examples and the current support vectors formed together a new subset. This subset was used to run the optimizer again repeatedly until some stopping criterion was met.

The result of the SVM training is also shown in the Graph 1.

![Graph 1](image_url)

**Graph 1. Illustration of the result by using linear kernel.**

The result of Graph 1 shows that the classification accuracy converges to 100 % after 91 training examples. As soon as the classification accuracy reaches the top, the result will not change irrespective of the newly added training examples.

### 4.2 Polynomial kernel

To estimate when the SVM algorithm converges to 100 % accuracy when compared to the result from the linear kernel, the polynomial kernel (described in Chapter 3) with different parameters was used for the next experiment. Table 6 shows similar results as the linear kernel. The parameters were set from the beginning, in case one, and they were not changed during the other cases when new training examples were added. If new parameters were added for every case, the result will be even better. New parameters could be set for every new added training example, but the flexibility will reduce, since the parameter set has to be done manually for each case. For instance, polynomial kernels $K(x_i,x_j) = (\gamma x_i^T x_j + \kappa)^p$, often uses the degree parameter $1,\ldots,5$ and the parameter gamma $\gamma \in [0,1]$. The polynomial kernels are a good model for low degree. If high degree is used, numerical difficulties will tend to appear. The gamma ($\gamma$) parameter serves as an inner product coefficient in the polynomial kernel. The scaling parameter $\kappa$ is usually fixed i.e. $K(x_i,x_j) \approx 1$ for all $i = 1,\ldots,l$. After defining a reasonable set of parameter combinations, the best parameter has to be selected for every set
i.e. the test error is minimized. But in this experiment, parameters were only set in the first case since it will not facilitate the operators to use the system if parameters have to be reset after every case. Different parameters were tested and the following parameters gave the best result in the first case with polynomial kernel:

- Degree ($P$): 2
- $\gamma$: 0.05
- $\kappa$: 1
- C: 10

<table>
<thead>
<tr>
<th># Cases</th>
<th># TR</th>
<th># SV</th>
<th># TE</th>
<th>Accuracy</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>13</td>
<td>419</td>
<td>300 / 419</td>
<td>71.60 %</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>24</td>
<td>406</td>
<td>321 / 406</td>
<td>79.06 %</td>
</tr>
<tr>
<td>3</td>
<td>39</td>
<td>36</td>
<td>393</td>
<td>323 / 393</td>
<td>82.19 %</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>47</td>
<td>380</td>
<td>353 / 380</td>
<td>92.89 %</td>
</tr>
<tr>
<td>5</td>
<td>65</td>
<td>53</td>
<td>367</td>
<td>347 / 367</td>
<td>94.55 %</td>
</tr>
<tr>
<td>6</td>
<td>78</td>
<td>68</td>
<td>354</td>
<td>349 / 354</td>
<td>98.59 %</td>
</tr>
<tr>
<td>7</td>
<td>91</td>
<td>76</td>
<td>341</td>
<td>341 / 341</td>
<td>100 %</td>
</tr>
<tr>
<td>8</td>
<td>104</td>
<td>86</td>
<td>328</td>
<td>328 / 328</td>
<td>100 %</td>
</tr>
<tr>
<td>9</td>
<td>117</td>
<td>101</td>
<td>315</td>
<td>315 / 315</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Table 6. Illustration of the results by using polynomial kernel.

The difference is not that big when comparing the results from Table 5 and 6. One can also see that the Graph 2 of the result from the polynomial kernel is quite similar to the result from the linear kernel.

Polynomial kernel type

![Graph 2](image_url)

Graph 2. Illustration of the results by using polynomial kernel.
Graph 3 illustrates similar curves from both experiments. Based on the results from both kernels, the linear kernel was preferred, since it was easier to use.

![Graph 3](image)

Graph 3. Illustration of the results with both linear and polynomial kernels.

Based on the results from the two experiments above, the linear kernel is proposed. Firstly, the linear kernel does not demand any parameter settings. Secondly, since the linear kernel classifier is the best classifier to use from a theoretical-learning point of view (given limited prior knowledge and a limited amount of training data). The linear kernel is not necessarily the best choice, but it has been chosen for the ease of use.

Hereafter, the linear kernel will be used in the following experiment since result shows that the data was linearly separable while accuracy converges to 100%. Therefore, there will be no need to experiment different kernels. Another advantage was that the polynomial kernel has three main parameters to be adjusted while the linear kernel has none.

### 4.3 Re-train misjudged experiments

The next experiment will consider the behavior of the SVM algorithm in more realistic cases. This experiment shows that by a few training examples, good generalization could be achieved. For example, in real productions, few training examples are exposed for the SVM algorithm in the beginning and according to error occurrences, new training examples will be added to the training set. Table 7 shows the experiment result when one training example was added at the time. This new examples that were added to the training set were chosen from the previous incorrectly classified case. After the first training with 13 examples, 300 of 419 cases were correctly classified. That means 119 (419-300=119) cases were misjudged, incorrectly classified. The new added example will therefore be chosen from the 119 incorrectly cases. The disadvantage of this experiment was that the new added training examples that were picked from each case were carefully analyzed to improve the classification result. The reason that only 13 examples were trained in the beginning was to provide a more realistic investigation by choosing at least one example from each of the 13 predefined rules. But in real production, the operators will not have time to create that many training examples manually.
<table>
<thead>
<tr>
<th># Cases</th>
<th># TR</th>
<th># SV</th>
<th># TE</th>
<th>Accuracy</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>13</td>
<td>419</td>
<td>300 / 419</td>
<td>71,60 %</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>14</td>
<td>418</td>
<td>324 / 418</td>
<td>77,51 %</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>15</td>
<td>417</td>
<td>363 / 417</td>
<td>87,05 %</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>416</td>
<td>366 / 416</td>
<td>87,98 %</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>17</td>
<td>415</td>
<td>357 / 415</td>
<td>86,02 %</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>18</td>
<td>414</td>
<td>361 / 414</td>
<td>87,20 %</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>19</td>
<td>413</td>
<td>374 / 413</td>
<td>90,57 %</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>20</td>
<td>412</td>
<td>377 / 412</td>
<td>91,50 %</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>21</td>
<td>411</td>
<td>383 / 411</td>
<td>93,19 %</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>22</td>
<td>410</td>
<td>395 / 410</td>
<td>96,34 %</td>
</tr>
<tr>
<td>11</td>
<td>23</td>
<td>23</td>
<td>409</td>
<td>394 / 409</td>
<td>96,33 %</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>24</td>
<td>408</td>
<td>401 / 408</td>
<td>98,28 %</td>
</tr>
<tr>
<td>13</td>
<td>25</td>
<td>25</td>
<td>407</td>
<td>404 / 407</td>
<td>99,26 %</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
<td>25</td>
<td>406</td>
<td>406 / 406</td>
<td>100 %</td>
</tr>
<tr>
<td>15</td>
<td>27</td>
<td>25</td>
<td>405</td>
<td>405 / 405</td>
<td>100 %</td>
</tr>
<tr>
<td>16</td>
<td>28</td>
<td>26</td>
<td>404</td>
<td>404 / 404</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Table 7. Illustration of the results from a realistic experiment.

By finding out which cases that were classified incorrectly and re-train the system by adding one of them, the system will converge to 100 % already after 13 added training examples. Of course the result will change according to which case that is to be chosen. But the consensus from this experiment was that the SVM algorithm could classify to 100 % accuracy with only a few training examples. Furthermore, the system was re-trained by two more correct classified cases just to see how the system behaves after reaching 100% accuracy. As predicted, the system continues to give perfect accuracy.

Graph 4. Illustration of the results from a realistic experiment.
5 Discussion

Linear kernel and polynomial kernel were investigated to find out the optimal kernel selection. By creating simple and well designed training examples, one does not have to use complicated and advanced kernels and still be able to achieve good results. The first experiment with linear kernel showed that after 91 training examples the SVM algorithm classified every test examples to 100% accuracy. This means that the goal of this experiment was achieved. But how reliable is this kernel method and what weaknesses does it enclose? The weaknesses of this experiment are that the training examples were randomly chosen for each time, which means that the result only shows that the network will approach to 100% eventually, but do not show any patterns of how fast it will converge or how many training examples that are needed. The SVM algorithm will converge faster if good training examples, training examples that become support vectors, are exposed. But this has not been tested in this first experiment, since the aim of this experiment was to investigate the performance of the linear kernel.

The second experiment was performed with the polynomial kernel with degree 2 to compare the result with linear kernel. Since the training examples were well designed, SVM algorithm should classify with good result by using polynomial kernel degree 2. As illustrated above in Graph 3, the result showed that the kernel converged to 100% similar to the linear kernel. But this experiment has a very pronounced weakness. The parameters were not set after every case. This means that the network will not give the best result in classifying the test examples. Despite the weakness, one can see that 100% accuracy was achieved after 91 training examples. Hypothetically, the polynomial kernel will converge faster than linear kernel. But it requires manual setting of new parameters every time to fit the new case. For this project, the aim was to find methods that do not complicate the use for production operators. That is why linear kernel will be the most suitable kernel for the error recovery.

The third experiment was performed using the linear kernel to do a realistic example in order to re-train misjudged examples. This experiment began with 13 training examples like the first two experiments, but the difference was that only one new training example was added for each new case. In real production, new training examples are achieved only when an error occurs. That is why the new training example is not randomly chosen. The new training example is chosen carefully on the basis of the cases that have not been correctly classified from the previous result cases. Taken together, SVM algorithm is a good tool for error recovery since convergence could be achieved after only 13 new added training examples. In conclusion, by using only a few training data the SVM is a good tool for classification.

It is satisfying that all three experiments gave 100% accuracy, since the real-world industrial robots need to be robust and reliable. Anything less than 100% accuracy will not be acceptable. However, the results are better than expected and it is questionable how all results can be so accurate. Are the experiments too simple or is the SVM method suitable for error recovery? One thing to think about is that the training examples have a great impact on the result. Different training examples give different results and the better designed training examples the better result. In this project we have 432 combinations and 13 rules that were created manually based on input from experienced robot engineers and knowledge from the system. It could of course be more or less than 13 rules that we provided the system. The rules were designed thoughtfully and covered different situations in different areas. This means that these 13 rules could be too well designed which makes it much easier for the experiments to reach 100% accuracy. In the real-world, the errors will occur and the operator is the one that designs the new rule for the system. This is also important and needs to be taken into account, since the operator’s decision will directly impact on the result. Furthermore, we only defined 4 output solutions, classes. System with more classes and ‘higher level’ complexity may not behave the same way and with 100% accuracy.
Though SVM seems to be suitable for this project, it still has a couple of disadvantages for real-world applications. SVM is established on the basis of the quadratic planning and it cannot distinguish the attribute importance through a training set. Also, it is time-consuming for large volume data classification and time series prediction. The training time and the occupied space for SVM must be shortened and reduced in order to be applicable in many real-world applications.

We talk a lot about the risks, safety and environment in today's society, but how are the risks, safety and environment in the robotics industry? Risk analysis involves the systematic use of available information to describe and calculate the risks of a given system. In risk analysis, we talk a lot about the probabilities and consequences. What are the probabilities of various adverse events and their consequences? With that as a basis, one can take various decisions to reduce risks. Therefore, it is important that we define all kinds of risks in the industrial robots error recovery environment, especially if machine learning methods are applied in the system as proposed in this project. Not having hand-coded traditional algorithm and letting the robot think, learn and act for itself is a high risk. It is not predictable and definitely not a guarantee on the behavior of the robots. For the SVM method in this experiment, the idea was to have the operator to define a new rule whenever an error occurs. That itself is a risk, since it puts that much freedom, trust and responsibility on the operator. So before any new learning algorithm is introduced to the industrial robots for error recovery in production, we need to define the goal. By defining the goal, all the uncertainties that prevent us from reaching the goal can be mapped. We need to map all situations that lead to problems and accidents.

Examples of risks for error recovery in industrial robots are:

- **Operator:** Badly defined error recovery from operator that could lead to
  - Stopped production
  - The robot breaks
  - Accidents, injuries

- **Robot:** Unpredictable behavior
  - Bug in the program
  - Breakages
  - Jammed parts
  - Incorrect decision
  - Stopped production
  - The robot breaks
  - Environment damages
  - Accidents, injuries

All these risks will also have economic consequences.
5.1 Future research

A new step towards the future would be to build systems based on learning from industrial robots.

In this project, a step by step implementation is proposed using the SVM algorithm for error recovery in production on the ABB robot in RWPM as a future improvement.

5.1.1 Implementation of proposed Error recovery tool based on SVM

At launch the system is untrained, which means that the SVM algorithm contains zero numbers of training data. This gives the operator the possibility to create the wanted error recovery coordinated to the environment. The second advantage is to make it easy to use for the operators, since there will be no rules to follow at early stages. The idea is to let the operators create their own rules among the production whenever errors occur. In the beginning, the robot will always stop when an error occurs to let the operator choose what solution that is required. The different cases of solutions will be predefined and presented on the FlexPendant touch screen together with the inputs. The operators only need to choose the wanted solution (in support of the giving inputs information) and add it to the SVM algorithm training set. Inputs and output solutions are predefined for the proposed example below, but they can be changed after the customers’ requirements. The rules are set by the example below.

1. When an error occurs, the robot will STOP immediately. The FlexPendant screen in production will automatically change to the new screen below.
The new screen tells the operator the input status after STOP. On the basis of this input information, the operator can choose the appropriate solution.

2. After choosing the wanted error recovery solution, press the button to the right in order to add the case as a training example to the SVM algorithm.

3. The saving process is needed to confirm that the procedure has been trained and saved. A pop up window will confirm that the procedure has been succeeded.

4. If the operator wish to ignore adding the case to the training set, press the button Cancel.

5. Step 1-3 is repeated until the network is completely trained.

When is the training satisfied? When is the network completely trained to give correctly classified solutions for all cases?

After a few numbers of added training examples the training model of the SVM algorithm will be satisfied and converge to 100 %, which means that the robot will classify correct solutions in all cases. The estimated numbers of needed training examples depend on the input vector matrix.

In the running production, the status panel on the FlexPendant production screen will have the above appearance (from step 1 to step 5) whenever an error occurs. In the status panel, Cell Status is telling the operator that something is wrong in the production. Since the SVM algorithm is completely trained, the system will activate and run the SVM algorithm according to the giving input shown on the screen.
1. The SVM network will automatically be used for error recovery in running productions.
2. The output solution from the SVM network will be presented on the FlexPendant Status screen, see picture below.
3. To complete the whole error recovery, the robot should follow the criteria of the classified solution.
6 Conclusions

In this project a Machine Learning algorithm for error recovery in RWPM has been implemented and examined. The aim of this project was to improve the existing error handling for ABB RobotWare Plastic Mould (RWPM) by implementing and investigate a pure learning approach to the robot. The reason why the SVM algorithm has been chosen to solve the problem is that SVM is a robust method that provides good generalization performance in real-world applications. In production, there will be no need for the operator to train the SVM with hundreds or thousands of training examples to achieve good generalization. The advantage with SVM is that good generalization could be generated with only a few training examples if the training examples are well designed. In conclusion, SVM is a good tool for classification and a suitable method for error recovery on the industrial robot for the plastic mold industry.

The standard technique for SVM is first to choose a suitable kernel method. There are no specific rules on how to choose these suitable kernels, but trial and error to achieve the best results. Some guidelines will be to begin with the simple kernel and move on to more complicated kernels if needed. Two kernels have been investigated to find a suitable kernel method. The first one was linear kernel and the second one was polynomial kernel with degree 2. The experiments showed that there was not a big difference between the two kernels. The result for the linear kernels and the polynomial kernels achieved almost the same accuracy rates. However, concerning the ease of use, the most suitable algorithm for the error recovery would be the linear kernel, since the linear kernel does not have to adjust to any parameters.
7 Acknowledgments

This master thesis was performed at ABB Shanghai Ltd. I would like to thank my supervisor Dr. Zhongxue Gan in ABB R&D for giving me this great opportunity to work for him. It was truly inspiring and enlightening. I would also like to thank my supervisor Prof. Örjan Ekeberg at the department of Computer Science and Communication (CSC) at the Royal Institute of Technology (KTH), and my mentor Anna Liberg at R&D in ABB Shanghai Ltd, for the valuable advice and support. Big thanks to my family especially my sister Dr. Joey Lau Börjesson and all the members in the ABB R&D team for helping me through the difficulties during the project.
8 References


