Support Vector Machine Learning in Multi-Robot Teams

Nicol Naidoo, Glen Bright

Mechatronics and Robotics Research Group
Department of Mechanical Engineering
University of KwaZulu-Natal
Durban, South Africa
Email: nic.naidoo@gmail.com, brightg@ukzn.ac.za

***Abstract*—In recent years, there has been a great research interest in cooperative mobile robotics. An advancement in industrial technology has seen the need for distributed applications in robotic systems where teams of robots are required to solve tasks intelligently and efficiently. Heterogeneity in robot teams adds complexity to a cooperative system since each member in the team varies in capability which determines its task abilities. The objective of this research paper is to introduce the use of a machine learning system to facilitate cooperation in multi–robot teams. Tests were performed and simulated for mobile robot cooperation in a material handling application during bottleneck conditions.**

Keywords–Multi–robot systems; cooperation; bottleneck; support vector machine; learning.

I. INTRODUCTION

Cooperation of Multi–Robot Systems (MRS) have drawn increasing attention in the past two decades since these systems have the ability to perform complex tasks more efficiently compared to single–robot systems [1] [2]. An implementation of a cooperative robot team in a manufacturing environment can, for example, solve the issue of bottlenecks in a production line, whereas the limitations of an individual robot can lead to a lot of problems in terms of time wastage, loss of revenue, poor quality products and dissatisfied customers.

Despite the advantages of MRS, there are still many challenges that exist such as task allocation, collision avoidance, communication, coordinating actions and team reasoning [3]. These challenges together with changing environments and robot heterogeneity, make it impossible for the MRS to predict all of the likely scenarios and thereby act on them. An effective solution to this problem is the incorporation of a learning component to the intelligence of a MRS.

Behaviour-Based Systems (BBS) [4] [5] are learning models that are designed using a bottom–up approach where survival behaviours, such as obstacle avoidance, constitute the low–level robot control and exploration and path planning make up the high–level control component; behaviours are introduced to the model until the desired robot–environment interaction is achieved. Behaviour selection is a key challenge in BBS since it determines which behaviours(s) control the robot at any given time; Reinforcement Learning (RL) has successfully contributed in this regard and has become an area of great interest in the research community [3].

Some other areas of learning mechanisms applied to MRS are artificial neural networks [6] and genetic algorithms [7] [8]. The focus of this paper is to discuss the use of the Support Vector Machine (SVM) learning algorithm in MRS. SVM learning is a supervised, classification or inductive learning scheme where the computing system learns from the database of past experiences to predict future outcomes; it has been successfully implemented in many applications, such as bioinformatics, and text and image recognition.

This paper aims to broaden its use in MRS applications where cooperation among robot team members is a key requirement. The remainder of the paper is structured as follows: Section II discusses the background and theory of SVMs, in particular, linear and non–linear classifiers, and some popular SVM libraries that can be used in applications; Section III discusses the design, implementation, and test results of the SVM learning system in a material handling application; Section IV concludes the paper and introduces further work to the research.

II. SVM BACKGROUND

SVM learning is related to statistical theory [9] and was first introduced as a classification method in 1992 [10]. It is widely used in bioinformatics due to its accuracy and ability to work with high–dimensional space data. The standard SVM is a binary linear classifier (commonly referred to as the linear SVM) which predicts whether an input belongs to one of two possible classes; this is accomplished by first building a model from a set of training examples, each consisting of input data that are mapped to the corresponding class label. SVM non–linear classifiers can be created by using non–linear kernel functions, further discussed in Section II-B.

A. SVM linear classifiers

In order to gain an intuition on what support vectors actually are and how they are used to create learning models a few preliminary mathematical terms will now be introduced. Given some training data set, D, with n points:

$$D = \{(\mathbf{x}_i, \mathbf{y}_i), \mathbf{x}_i \in R_m, \mathbf{y}_i \in \{-1, 1\}\}_{i=1}^{n} \quad (1)$$

The boldface $\mathbf{x}$ term is a vector with training example inputs $\mathbf{x}_i$; each $\mathbf{x}_i$ has an m–dimensional size of m features. The classifier term, $\mathbf{y}_i$, is either -1 or 1 and indicates the class to which each point $\mathbf{x}_i$ belongs.

In Figure 1 (a), the training examples are classified into positive and negative classes. The hyperplane, H, is the *decision boundary* that divides the regions between positive and negative classes. The decision boundary is said to be *linear* since the examples are linearly separable and a classifier with

 ISBN: 978-1-61208-405-3

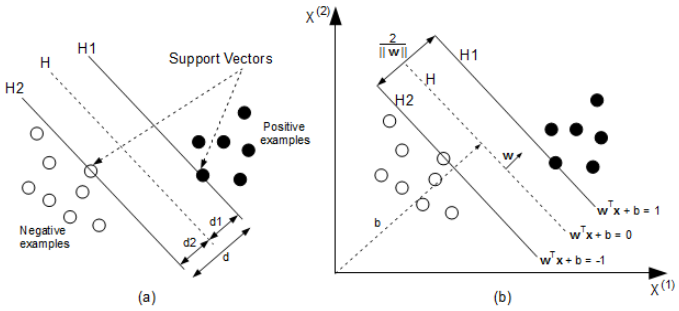

Figure 1. (a) Hyperplanes and margins. (b) Margin classifiers

a linear decision boundary is called a *linear classifier*. *H1* and *H2* are lines that intersect the *support vectors*, these are the training examples that are closest to the decision boundary and they determine the margin (*d1* and *d2*) at which the two classes are separated from the hyperplane (or decision boundary). The SVM algorithm is also termed as the *large margin classifier* since its goal is to maximise the margin *d* for a set of classified training examples.

Figure 1 (b) is an extension to (a) and shows the training examples on a two dimensional feature space with features $x^{(1)}$ and $x^{(2)}$. A linear classifier is based on a linear function of the form:

$$f(x) = \mathbf{w}^T\mathbf{x} + b \tag{2}$$

where **w** is commonly known as the weight vector and *b* is the bias. The product between **w** and **x** is known in linear algebra as the dot product and is defined as $\mathbf{w}^T\mathbf{x} = \sum_i w_i x_i$. The equation for the hyperplane is:

$$H : \mathbf{w}^T\mathbf{x} + b = 0 \tag{3}$$

where the purpose of the bias can be seen as moving the plane away from the origin, i.e., if *b=0* the hyperplane would go through the origin. Equations (4) and (5) are related to planes *H1* and *H2*:

$$H1 : \mathbf{w}^T\mathbf{x} + b = 1 \tag{4}$$

$$H2 : \mathbf{w}^T\mathbf{x} + b = -1 \tag{5}$$

and are equated to 1 and -1 respectively due to the definition of the classifier term, $\mathbf{y}_i$ in (1). Using geometry and referring to Figure 1 (b), the margin between *H* and *H1* is $1/\|\mathbf{w}\|$, where $\|\mathbf{w}\|$ is the length of the vector **w** and is given by $\sqrt{\mathbf{w}^T\mathbf{w}}$; hence the margin between *H1* and *H2* is $2/\|\mathbf{w}\|$. In order to maximise the margin, $\|\mathbf{w}\|$ must be minimised subject to the following constraints which are added to prevent data points falling into the margin:

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1, \quad \{for\ \mathbf{y}_i = 1\} \tag{6}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \{for\ \mathbf{y}_i = -1\} \tag{7}$$

Equations (6) and (7) can be combined to form:

$$\mathbf{y}_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \{for\ 1 \leq i \leq n\} \tag{8}$$

Minimising $\|\mathbf{w}\|$ subject to (8) is a *constrained optimisation problem* and solving it requires using the method of *Lagrange multipliers*. A method that can be used to obtain a dual formulation, expressed in terms of α_i variables [11]:

$$\text{maximise } \alpha\text{: } \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T\mathbf{x}_j \tag{9}$$

$$\text{subject to: } \sum_{i=1}^{n} y_i\alpha_i = 0, \quad \alpha_i \geq 0 \tag{10}$$

The dual formulation also defines the weight vector in terms of the training examples:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i \tag{11}$$

B. SVM non–linear classifiers

In most SVM classification problems, the data set is not linearly separable. Literature [10] solves this challenge by mapping the original finite dimensional space into a higher dimensional space making the separation much easier in that space, as illustrated in Figure 2.

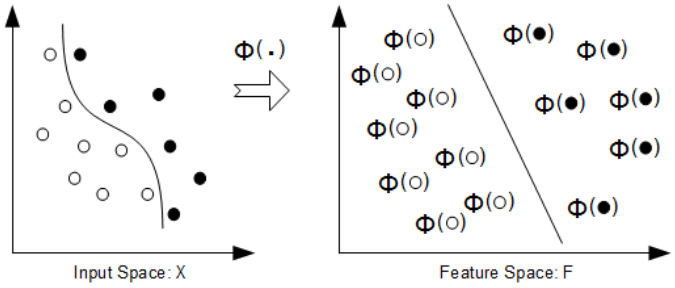

Figure 2. Non-linear classification mapping

The mapping is achieved by the use of *Kernel functions* and the dot product property in the linear SVM algorithm. The $\mathbf{x}_i^T\mathbf{x}_j$ terms in (9) are replaced by the kernel function, *K*:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j) \tag{12}$$

which can represent (among others) a *polynomial*, *gaussian*, or *hyperbolic function* [12]. The linear classifier is also known as the *linear kernel*.

C. Multi–class SVM

SVMs are inherently binary classifiers however, there are many applications where multiple classifications are required. The common method of solving the *M-class* problem is to divide it into multiple binary classification problems [13]:

- One-vs-All: This method constructs *N* binary SVM classifiers, where *N* represents the number of classes. Every *i*-th SVM is trained to differentiate the training examples of the *i*-th class from the examples of the other classes. At the classification phase, samples are classified in accordance to the highest output function among all the SVMs.
- One-vs-One: This strategy constructs one SVM for every pair of classes, hence for an *M-class* problem of *N* classes, *N*(*N*-1)/2 SVMs are trained. A maximum-wins voting concept is used where each SVM classifier assigns the sample to one of the two classes and

 ISBN: 978-1-61208-405-3

the number of votes for the assigned class increases by one; in the end, the class with the most votes determines the classification of the sample.

Another approach to the *M-class* problem, which avoids the use of multiple binary classification problems, involves the application of a single optimisation model [14].

D. SVM software libraries

Over the past two decades there has been a wide interest in SVM algorithms which has led to the development of many solvers for SVM optimisation problems. Two popular open source solvers are LIBSVM [15] and SVM^{light} [16]. These solvers form excellent tools for researchers since they eliminate the vast quantity of time that could be spent on the complex software development of SVM optimisation algorithms and thus allow the scientist to focus on the primary components of the research. The LIBSVM library was used in this research.

III. MULTI-ROBOT COOPERATION APPLICATION

The multi–robot cooperation research was tasked for advanced manufacturing environment applications where dissimilar (or heterogeneous) mobile robots are used in discrete processes. The idea of cooperation between robots when there is a need can prevent bottlenecks, improve material flow and thus contribute to the upkeep of a good supply chain management system. The objective of the research is to aid any member in a team of heterogeneous robots in task decision making. Each robot in the system must be capable of moving autonomously in the known environment while avoiding obstacles and maintaining a teamwork approach in the resolution of common goals. An essential component of the design is the machine learning algorithm which is used to predict suitable goal destinations for each mobile robot, given a set of input parameters.

Figure 3. Mobile robot hardware used for the research

The three mobile robots (Figure 3) used in the research were the Performance PeopleBot, the Segway RMP200, and the Segway RMP400. The platforms were chosen on the basis of their availability; they are mainly used for research purposes and not suited for manufacturing environment applications, which is acceptable for the research since the objective is to establish the concept of a cooperating team of heterogeneous mobile robots, irrespective of their abilities and functionality.

A. Material handling application description

The objectives of this research were tested in a material handling application, as illustrated by the Supervisory Control and Data Acquisition (SCADA) screenshot shown in Figure 4. The application shows a resource buffer (“R”), a storage buffer (“S”), 6 process buffers (“B1”–“B6”), 3 machines (“M1”–“M3”), and a conveyor; it was designed in this manner to demonstrate the cooperative ability of the system during bottleneck and fault conditions. The application was set up for the PeopleBot to transport material from “R” to “B1”, the RMP200 move material from “B4” to “B5”, and the RMP400 to finally move the end product from “B6” to “S”.

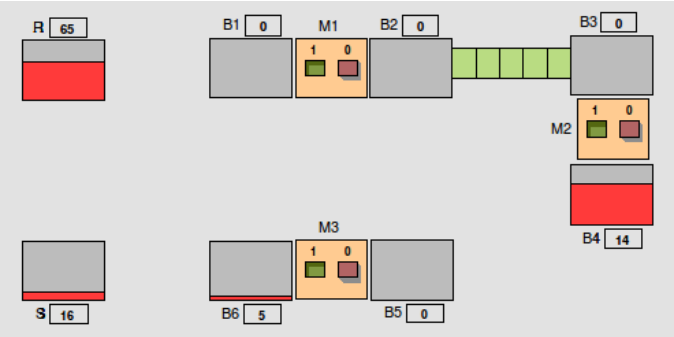

Figure 4. Material handling application for the research

The numbers within the blocks shown in Figure 4 represent the quantity of material in the buffer and the buffer levels are illustrated as a percentage of their total capacity, thus the bottlenecks in the process can be seen at a glance during production. The calculations for the quantity of material, buffer capacities, and machine process rates are all done in the simulation program which is located in the SCADA component of the system, the details of which are beyond the scope of this paper.

During the implementation and debug phase of this research, bottleneck conditions were intentionally created by altering: 1) the material handling capacities of the robots, 2) the machine efficiencies, and 3) the buffer capacities.

B. Design overview

Figure 5 shows the design overview of the Mechatronic system. The scope of the design consists of an integration of the following components:

- Robot hardware
- Middleware
- Agent program
- SCADA

The *robot hardware* comprises of the mechanical robot (PeopleBot, RMP200, RMP400), the sensors (LRF, sonars), and the actuators (drives, motors). The *middleware* layer is necessary since it is responsible for interpreting the high level (agent program) commands and presents them to the sensors and actuators through the use of low level software driver modules.

 ISBN: 978-1-61208-405-3

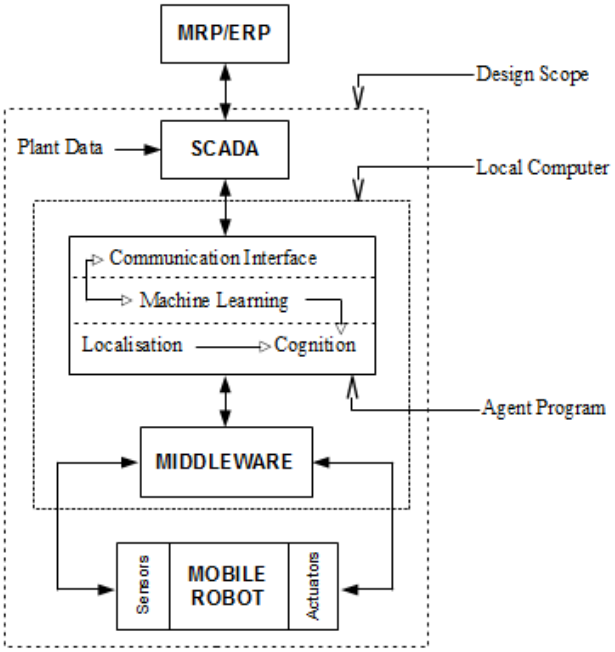

Figure 5. Design overview of the Mechatronic system

The *agent program* is the robot's decision making component in the system design as it determines which task (primary or secondary) is required by the robot at a specific point in time. In addition to the *localisation* and *cognition* modules, the agent program contains the *machine learning* module which was incorporated in the system design due to the following benefits:

- Robot heterogeneity and task taxonomy: due to the different capabilities of each robot together with the variations in tasks, the system is required to identify whether or not a particular robot can perform a secondary task when required. An integrated learning system will ensure that each robot goes through an engineering teaching process so that the robot "agent" can identify itself as a helping agent when the need (bottleneck) arises.

- Manufacturing environment reconfiguration: changes in the environment, caused by the manufacturing of different products or the implementation of new machinery, will have a minimal impact on the cooperative function of each robot since the learning module ensures that robot agents are re-taught accordingly. A further advantage is the saving of money and resources that would have been required to reconfigure the robots to adapt to the new environment.

The *agent program* also comprises a *communication interface* which sends, receives and processes data packets to/from the plant SCADA system. The SCADA is a vital component in the manufacturing plant automation system since it makes process information available to operators and engineers for the purpose of monitoring and control.

C. SVM implementation

The LIBSVM library was used in the agent program for the train and prediction algorithms, and the polynomial kernel was chosen as the non–linear SVM kernel function. There are two phases to the SVM algorithm:

- the *learning* phase, where agents are taught by the system on the best goal location to follow. The teaching process can take place in an offline (simulation) environment, or online through the Graphical User Interface (GUI) interface of the SCADA system.
  The objective of the learning phase is to build a knowledge database of SVM features with training examples. Figure 6 is an extract of the "train.txt" file that contains the training examples. The SVM features in the file (labeled 1 to 8) are the buffers in the manufacturing application and the training examples (the values positioned to the right of the colons) are the number of materials in each buffer. The (output) goal location for the robot is the first number in each line of the file.

```
1 1:100 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:0 3:0 4:0 5:0 6:0 7:0 8:0
1 1:80 2:20 3:0 4:0 5:0 6:0 7:0 8:0
1 1:60 2:0 3:0 4:0 5:15 6:1 7:4 8:0
1 1:60 2:0 3:0 4:0 5:15 6:0 7:5 8:0
```

Figure 6. Train.txt file extract with SVM features and training examples

- the *train–prediction* phase uses the data collated in the learning phase (i.e., the data contained in the train.txt file) to generate training models for each agent; the goal output for each agent is then accomplished by using the current data values (obtained from the data packet) as inputs to the prediction algorithm. The current data values represent the immediate status of the manufacturing process; they are stored as a string of data in the "test.txt" file which is used as an input to the SVM prediction algorithm. Figure 7 illustrates the entire process of training, building the model, and predicting the goal output for each robot in the system.

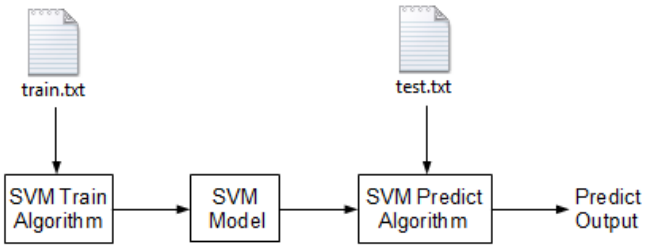

Figure 7. Process of the SVM train–predict phase

D. Simulation results and discussion

This section produces the results of the tests performed during the simulation of the system. Bottlenecks were created by varying the load carrying capacities of the robots, however, there were other options by which this could have been done,

namely: 1) vary the machine or conveyor efficiencies, and 2) change the buffer capacities.

During the SVM teach phase of the tests, the PeopleBot was taught to help the RMP200 at the bottleneck. Figure 4 showed a screenshot of the material handling application, where the PeopleBot's primary task is to move materials from the resource buffer ("R" or "B0") to "B1", the RMP200 has the single task of transporting material from "B4" to "B5", and the RMP400 also has a single task of moving the final product from "B6" to the storage buffer ("S" or "B7").

A bottleneck was created at "B4" by reducing the load carrying capacity of the RMP200 from 20 materials to 5 materials. The capacity of the PeopleBot remained the same (at 20 materials), this ensured that the material build up rate at B4 was greater than the buffer process rate, resulting in a bottleneck.

Four types of simulation tests were performed:

- normal operation: the load carrying capacities of the robots were configured to prevent bottleneck conditions.
- bottleneck condition: the load carrying capacities of the robots were configured to promote bottleneck conditions.
- cooperation at the bottleneck: a robot agent was allowed to help another agent at the bottleneck.
- cooperation during a robot fault: a robot agent was allowed to take over the tasks of the faulty robot so that the possibility of the occurrence of a bottleneck is reduced.

A discussion of all four types of tests is beyond the scope of this paper, hence only the *bottleneck condition* and *cooperation at the bottleneck* cases will be discussed.

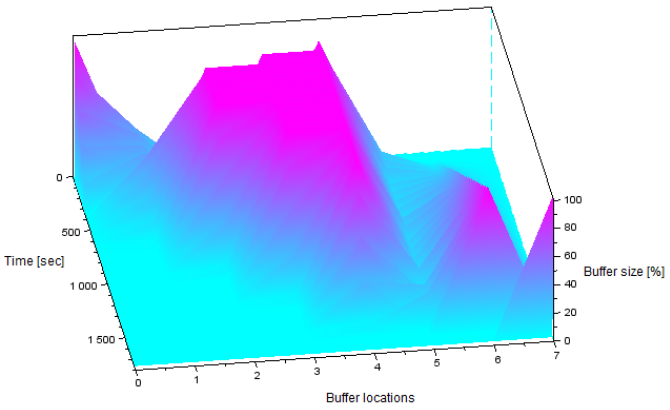

Figure 8. Material distribution graph: bottleneck condition

The material distribution graph for the *bottleneck condition* simulation test is given in Figure 8. The graph has three axes: the x–axis represents the buffer locations, ranging from 0 (buffer B0) to 7 (buffer B7); the y–axis represents the time (in seconds) of the simulation; the z–axis gives the number of materials, in a percentage, at each buffer location. The percentage is calculated by the following equation:

$$B_{size} = \frac{B_{num}}{B_{cap}} * 100 \qquad (13)$$

where B_{num} is the number of materials in the buffer and B_{cap} is a constant which represents the number of materials that the buffer can contain, i.e., the buffer capacity.

The visual trend in the graph shows a decrease in material count at the resource buffer (which was initialised with 100 materials) and an increase in material count at the storage buffer, towards the end of the simulation. Table I gives more detail to the *bottleneck condition* simulation and lists the values of some test parameters such as the total simulation (or production) time and the total operation time of each robot agent.

TABLE I. SIMULATION RESULTS FOR THE BOTTLENECK CONDITION

Test parameter	*Value*
Total simulation time	1763 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	349 sec (19.8%)
Agent 2 operation time	1520 sec (86.2%)
Agent 3 operation time	93 sec (5.3%)
Buffer 2 @100%	282 sec (16.0%)
Buffer 3 @100%	594 sec (33.7%)
Buffer 4 @100%	936 sec (53.1%)

Agents 1, 2 and 3 are the PeopleBot, RMP200 and RMP400 respectively. The values within brackets in the table are the percentages of the total simulation time. The large simulation time for the *bottleneck condition* is due to the bottleneck at buffer 4, where the RMP200 cannot transport the required amount of material to keep up with the incoming rate at the buffer. The bottleneck problem caused a cascaded effect (depicted in Figure 8) to fill up buffer 3 and buffer 2. The purpose of the *bottleneck condition* simulation was two–fold: 1) to emphasise the impact of the bottleneck on the production system, and 2) to set the stage for an implementation of the cooperative learning system in mitigating the bottleneck.

The *cooperation at the bottleneck* simulation was performed by allowing the SVM–trained PeopleBot agent to assist the RMP200 agent at the bottleneck (buffer 4), hence the PeopleBot executes its primary task of transporting material from B0 to B1 as well as "cooperates" by effecting its secondary task of moving material from B4 to B5. The material distribution graph in Figure 9 reflect the results of the cooperative learning system.

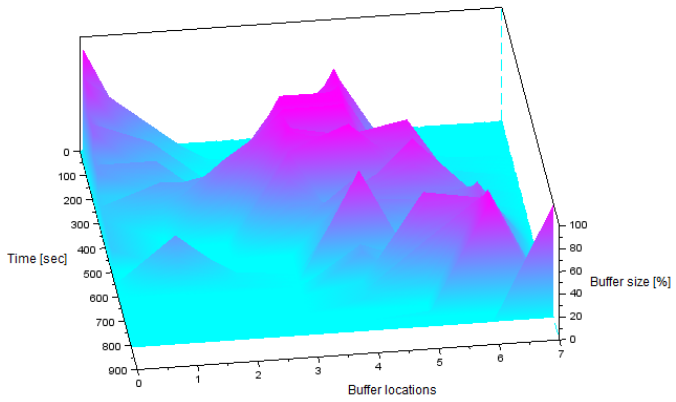

Figure 9. Material distribution graph: robot cooperation at bottleneck

An analysis of the SVM output results in the subplot of Figure 10 gives an interesting perspective on the periods at which the algorithm determines the assistance of the PeopleBot at the bottleneck. The SVM outputs for the PeopleBot agent are either "1" or "2", representing the primary or secondary task respectively. During the teach phase, the PeopleBot agent was taught to assist at B4 when the size of B0 is low and when the sizes of B4 and/or B3 are high. The effect of the teaching exercise is clearly shown in Figure 10 since the SVM predictions are "2" during conditions where the test parameters of the SVM features (i.e., the buffer sizes) are approximately the same as the SVM training examples.

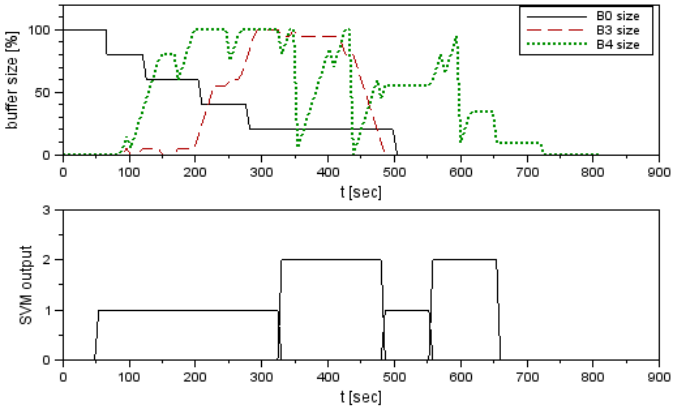

Figure 10. PeopleBot SVM outputs: cooperation at bottleneck

TABLE II. SIMULATION RESULTS FOR THE COOPERATION AT THE BOTTLENECK CONDITION

Test parameter	*Value*
Total simulation time	809 sec
Agent 1 load capacity	20 materials
Agent 2 load capacity	5 materials
Agent 3 load capacity	100 materials
Agent 1 operation time	600 sec (74.2%)
Agent 1 primary task	62.5%
Agent 1 secondary task	37.5%
Agent 2 operation time	678 sec (83.8%)
Agent 3 operation time	91 sec (11.3%)
Buffer 3 @100%	42 sec (5.2%)
Buffer 4 @100%	138 sec (17.1%)

Table II lists the total simulation time of 809 seconds—a 54% reduction in comparison to the previous simulation case. The table also reflects the task distribution percentage for agent 1: the SVM algorithm determined the secondary goal for the PeopleBot 3 times out of a total of 8 iterations in the simulation, i.e., the PeopleBot spent 37.5% of its operation time on the secondary task and 67.5% on its primary task. The simulation also resulted in an elimination of buffer 2 from the bottleneck cascade and showed reduced buffer–full times of buffer 3 and buffer 4 to 5.2% and 17.1%, respectively.

IV. CONCLUSION

The main objective of the research was the demonstration of a cooperative robot system using a machine learning approach. This objective was achieved by the successful performance of the SVM algorithm, where the bottlenecks were alleviated by the cooperating agent, significantly improving the manufacturing production times. The SVM learning algorithm essentially predicts and determines the goal tasks of each robot agent in the network by using a database of training examples.

The research discussed in this paper broadens the use of SVM algorithms (and potentially other supervised learning algorithms) in the area of multi–robot systems and manufacturing applications. The attraction of a learning based system is the semi–elimination of hard coded programmed solutions for specific scenarios; the learning system can adapt to dynamic environments and plant reconfiguration conditions.

Further work to this research will see the implementation of a reinforced learning system where the agents dynamically learn the "positive" and "negative" examples from the environment without going through a training exercise facilitated by the robot operator. Another desired modification to the system is the use of an automated selection of a training database in a suite of databases, this is useful when an agent has to solve a variety of problems, requiring the employment of multiple sets of training data.

REFERENCES

[1] Y. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," Autonomous Robots, vol. 4, no. 1, Kluwer Academic Publishers Hingham, MA, USA, 1997, pp. 7–27.

[2] L. E. Parker, T. Arai, and E. Pagello, "Advances in Multi-Robot Systems," IEEE Transactions on Robotics and Automation, vol. 18, 2002, pp. 655–661.

[3] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," In Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05), Essex, UK, 2005.

[4] L. E. Parker, "Alliance: An architecture for fault tolerant multi robot cooperation," IEEE Transactions on Robotics and Automation, vol. 14, no. 2, IEEE, 1998, pp. 220–240.

[5] M. J. Mataric, "Learning in behaviour-based multi-robot systems: policies, models and other agents," Journal of Cognitive Systems Research, vol. 2, 2001, pp. 81–93.

[6] S. Bhattacharya and S. Talapatra, "Robot motion planning using neural networks: A modified approach," International Journal of Lateral Computing, vol. 2, no. 1, World Federation on Lateral Computing, 2005, pp. 9–13.

[7] E. Sahin and W. Spears, "Swarm robotics, a state of the art survey," Lecture notes in Computer Science, vol. 3342, 2005.

[8] N. Naidoo, G. Bright, and R. Stopforth, "Material flow optimisation in flexible manufacturing systems," In Proceedings of the 6th IEEE Robotics and Mechatronics Conference (RobMech), Durban, South Africa, 2013, pp. 1–5.

[9] V. Vapnik, The Nature of Statistical Learning Theory. Springer–Verlag, New York, 2000.

[10] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," In Proceedings of the 5th annual workshop on Computational learning theory, ACM, New York, USA, 1992, pp. 144–152.

[11] C. Cortes and V. Vapnik, "Support vector networks," Machine Learning, vol. 20, no. 3, Kluwer Academic Publishers, Boston, 1995, pp. 273–297.

[12] B. Schlkopf and A. Smola, Learning with Kernels. MIT Press, 2002.

[13] C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," IEEE Transactions on Neural Networks, IEEE, 2002.

[14] K. Crammer and Y. Singer, "Algorithmic implementation of multiclass kernel-based vector machines," Journal of Machine Learning Research, vol. 2, no. 1, ACM, 2002, pp. 265–292.

[15] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," retrieved: March 2015. [Online]. Available: http://www.csie.ntu.edu.tw/ cjlin/papers/libsvm.pdf

[16] T. Joachims, Making large-scale support vector machine learning practical: Advances in Kernel Methods. MIT Press, 1998.

 ISBN: 978-1-61208-405-3