Application of Machine Learning Methods to the Open-Loop Control of a Freeform Fabrication System

Evan Malone, Oliver Purwin
Mechanical & Aerospace Engineering, Cornell University, Ithaca NY USA

Abstract

Freeform fabrication of complete functional devices requires the fabrication system to achieve well-controlled deposition of many materials with widely varying material properties. In a research setting, material preparation processes are not highly refined, causing batch property variation, and cost and time may prohibit accurate quantification of the relevant material properties, such as viscosity, elasticity, etc. for each batch. Closed-loop control based on the deposited material road is problematic due to the difficulty in non-contact measurement of the road geometry, so a labor-intensive calibration and open-loop control method is typically used. In the present work, k-Nearest Neighbor and Support Vector Machine (SVM) machine learning algorithms are applied to the problem of generating open-loop control parameters which produce desired deposited material road geometry from a description of a given material and tool configuration comprising a set of qualitative and quantitative attributes. Training data for the algorithms is generated in the course of ordinary use of the SFF system as the results of manual calibration of control parameters. Given the large instance space and the small training data set compiled thus far, the performance is quite promising, although still insufficient to allow complete automation of the calibration process. The SVM-based approach produces tolerable results when tested with materials not in the training data set. When control parameters produced by the learning algorithms are used as a starting point for manual calibration, significant operator time savings and material waste reduction may be achieved.

Introduction

Solid freeform fabrication (SFF) is the name given to a family of manufacturing processes which allow three dimensional printing of arbitrarily shaped structures, directly from computer-aided design (CAD) data.

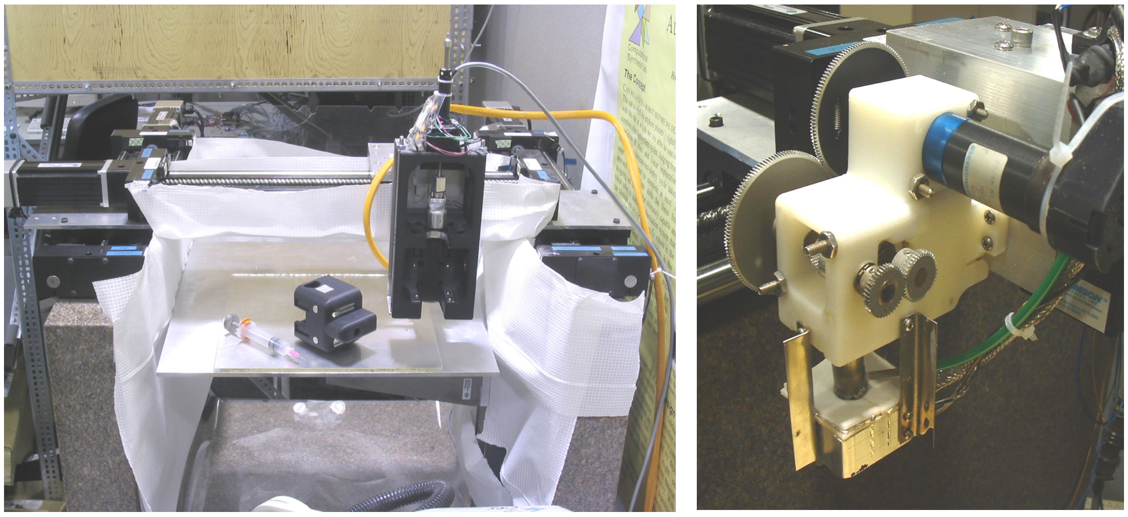
(a) (b)

Figure 1. Fabrication platform: (a) 3-Axis gantry robot for deposition with cartridge/syringe tool, (b) continuous wire-feed tool

Typically, an SFF system consists of a tool which dispenses a “road” or line of material, a robotic positioning system which moves the tool along 3-dimensional trajectory, and a software control system. SFF has traditionally focused on producing passive mechanical parts. Advances in this technology and developments in materials science make it feasible to begin the development of a single, compact, robotic SFF system – including a small set of materials – which can produce complete, active, functional electromechanical devices, e.g. mobile robots. A research SFF system with two material dispensing tools has been constructed (Figure 1**Error! Reference source not found.**) pursuant to this goal, and Figure 2 depicts a Zn-air battery produced with this system. One of the challenges in developing such a system is in achieving precise, accurate, and repeatable dispensing of materials despite the difficulty of automatically monitoring output quality, and the significant variations in properties between materials – even between batches of the same material. Currently, these challenges are handled via an extensive and laborious manual calibration process for each batch of each material, immediately prior to use. During calibration, the SFF system produces a series of rectilinear test patterns (Figure 7), and control parameters are tuned until the produced pattern matches the desired pattern to the operator's satisfaction.

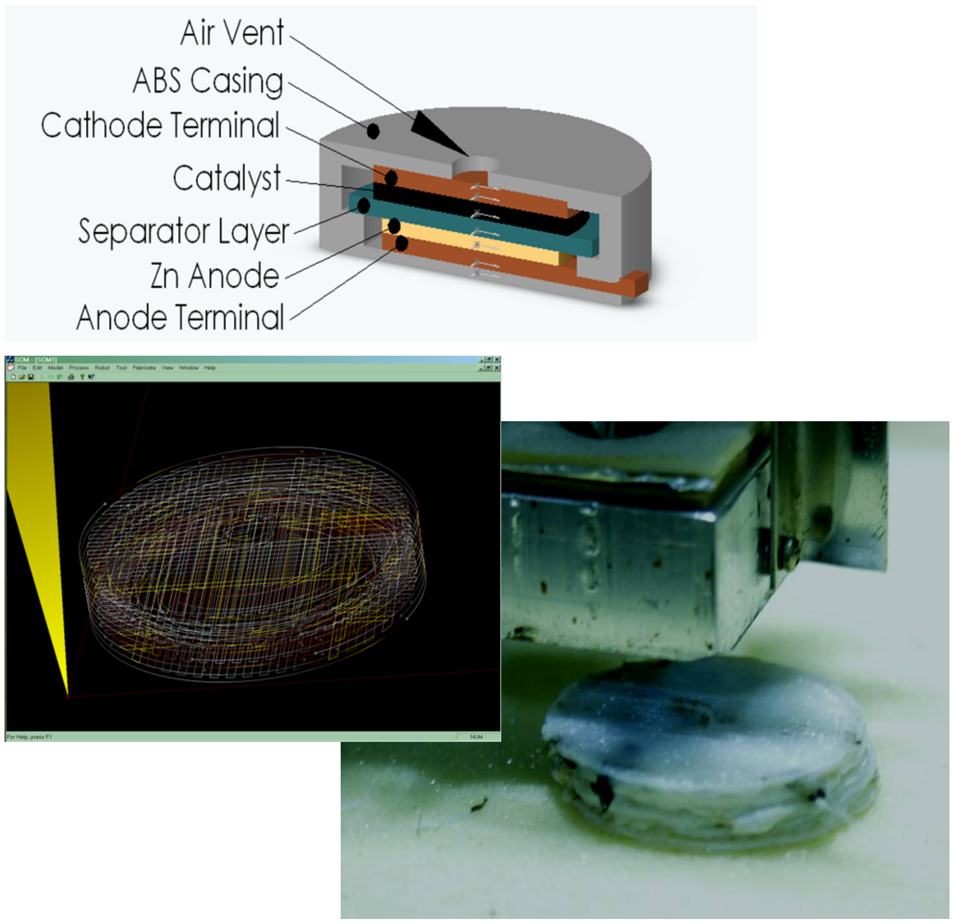

Figure 2. Zn-air battery produced via SFF

The control parameters (Table I) describe piecewise-linear profiles (Figure 3) for the commanded extrusion rate from the tool and for the robot trajectory speed. The k-Nearest Neighbor and SVM algorithms have been applied to the computation of control parameters for the syringe tool (Figure 1**Error! Reference source not found.**a). The inputs (i.e. "attributes") to the learning algorithms are simple quantitative and qualitative descriptions of the material, the tool, and the deposited road (Table II). The attributes consist of parameters which are strictly dependent on the material itself as well as desired properties of the extruded material road (e.g. width and height of the dispensed material). In that these attributes have been intuitively selected, it is unclear whether they are necessary or sufficient to fully represent the problem domain.

Table I. Example control parameters vector

Parameter	Value	Unit
ACCELDELAY	0.2	s
INPUTACCEL	1	step/ms^2
PUSHOUTSPEED	300	step/s
PUSHOUTTIME	0.4	s
INPUTSPEED	50	step/s
OUTPUTACCEL	40	mm/s^2
OUTPUTDECEL	-30	mm/s^2
OUTPUTSPEED	5	mm/s
DECELDELAY	0.25	s
INPUTDECEL	1	step/ms^2
PULLBACKSPEED	-300	step/s
PULLBACKTIME	0.4	s

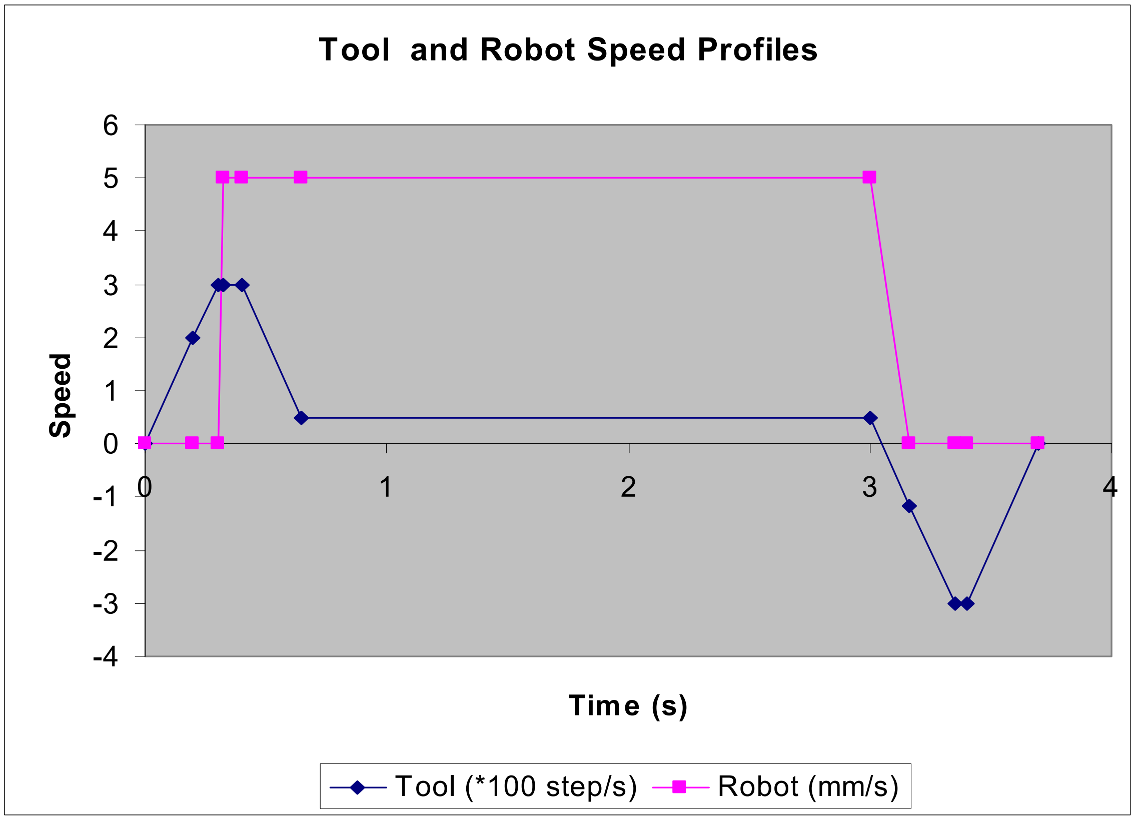

Figure 3. Example control profile

Table II: Material, Tool, and Path Description Attributes

Category	Attribute	Datatype	Example	Note/Enum
Material Intrinsic	MATERIALNAME	Enum	HP_GREASE	name of material
	NUMPHASES	Integer	1	
	ELASTICITY	Enum	LOW	low, medium, high
	FRICTION	Enum	LOW	low, medium, high
	STABILITY	Enum	HIGH	low, medium, high
	CLOGGING	Enum	LOW	low, medium, high
	SHRINKING	Enum	LOW	low, medium, high
	FLOWING	Enum	LOW	low, medium, high
Material Preparation	TRAPPEDGAS	Enum	HIGH	low, medium, high
	AGE	Float	24	hours
	PROCEDURE	Enum	PUMP_IN_LUER	preparation method
Tool Properties	TOOLNAME	Enum	SYRINGE	
	NOZ_ID	Float	0.8382	diameter, mm
	NOZ_TYPE	Enum	NEEDLE	needle or taper
	NOZ_LENGTH	Float	35.5	mm
	PISTON	Enum	NEOPRENE	piston material
Path Properties	PATHWIDTH	Float	1.1	mm
	PATHHEIGHT	Float	1	mm
	STARTERR	Enum	VERY_EARLY	very early, early, ok, late, very late
	STOPERR	Enum	LATE	very early, early, ok, late, very late

Data Collection

Training data has been compiled using the current manual calibration process, a set of materials with a variety of properties, and a standard test pattern. Each successful test pattern (e.g. the material has been dispensed with uniform width over the entire road length, and deposition begins and ends at the start and end, respectively, of the geometric path) is considered a training sample. The data STARTERR and STOPERR, which describe the observed delay between the start of path motion, and the commencement of material deposition, are used to determine acceptable training data. Ideally, only training data for which both STARTERR and STOPERR have the value OK would be used, but given that achieving a single such data point can take more than an hour, STARTERR and STOPERR values of OK, EARLY and LATE are considered acceptable. Although this adds some noise to the data set, it increases the number of data points by an order of magnitude, and it is believed that the noise will be roughly symmetrically distributed about the ideal parameter values. A training sample contains the material, tool and path attributes and the required control parameters. Using these criteria, the training data set comprises **63** calibrations and **9** separate materials.

During the training process the main objective is to generate as many training samples as possible, the actual width and height of the dispensed material is of minor importance, as long as the road is uniform. On the other hand, when testing the trained machine learning algorithms, the road properties become very important since the objective of the SFF process is to dispense the material with a well-defined geometry. The precision and accuracy of material dispensing on test patterns, as well as the time required for calibration are used as the basis for comparison between each of the machine learning methods and the manual calibration method.

Implementation

Two different algorithms for the machine learning task were implemented: k-Nearest Neighbor (k-NN) and Support Vector Machines (SVM). The following sections cover the implementation and test of these algorithms. Vectors will be indicated using bold letters. A bold letter with an index denotes a particular vector out of a set of vectors.

Description of k-NN Approach

The k-NN algorithm is a memory-based or instance-based algorithm, meaning that the entire training dataset remains in memory (see Mitchell for in-depth explanation). There is no explicit training step for the algorithm or extraction of a generalized classification rule. During classification (generation of a set of control parameters for a new material), the algorithm compares the new material description vector, $\mathbf{x}_{\text{new},}$ with all description vectors in the training set and computes a "distance" function based on the attributes (material, tool, and road description data). The label or parameter vector $\boldsymbol{y}_{\textbf{new}}$ for $\mathbf{x}_{\text{new}}$ is a combination of the parameter vectors of the closest neighbors. The idea is that attribute vectors which are close in the instance space should also have similar labels. This method has the advantage that an entire vector of labels can be mapped to a new example at once as opposed to SVMs, where each SVM can only produce a single output value. Since this algorithm is relatively easy to program, an implementation was written in C as part of the project. The training and test data are stored in text files to maintain readability by a human user.

Procedure to classify a new attribute vector $\mathbf{x}_{\text{new}}$:

1) Normalization of the attributes
2) Computation of distances to all samples in the training set
3) Sorting of training set according to distance to $\mathbf{x}_{\text{new}}$
4) Computation of weighted average of the labels of the k nearest neighbors

Normalization

In the problem at hand, different attributes have different ranges. For example, the average "Pushout Speed" is three orders of magnitude larger than the average "Pushout Time". In order to ensure an equal weighting of the distances for all attributes x_i, all attribute vectors are normalized to a range of 0 to 1. The normalization is based on the minimum and maximum values $x_{i,\min}$ and $x_{i,\max}$ of each attribute in the training data. The normalized attribute $x_{i,\text{norm}}$ is therefore:

$$x_{i,norm} = \frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}}$$

The entire training set and the new examples are normalized.

Compute Distances

The distance d between two attribute vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ is defined as the Euclidean distance

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{N} (\mathbf{x}_{1,i} - \mathbf{x}_{2,i})^2}$$

where N is the number of attributes in the attribute vectors.

Sort the training set

After the distances of the new example to all instances in the training set have been computed, the training set is sorted according to distance. Since the training set in this particular case is not large (less than 100 examples), a very simple "Insert Sort" algorithm has been implemented.

Compute new label

The label $\mathbf{y}_{\text{new}}$ for the new example $\mathbf{x}_{\text{new}}$ is calculated according to

$$y_{new} = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

where k is the number of neighbors to be considered, $\mathbf{y}_{\text{i}}$ are the labels of the closest neighbors, and w_{i} are the weights assigned to each neighbor. There are many different ways of defining the weights w_{i}, but in general closer neighbors should carry more weight than neighbors who are farther away. Two different weight functions were used. The first attempt was based on the inverted distance to the new example:

$$w_i = \frac{1}{d(\mathbf{x}_{new}, \mathbf{x}_i)}$$

The distance d had to be checked before computing the weight in order to avoid a division by zero in case $\mathbf{x}_{\text{new}}$ matches up perfectly with one of the instances from the training set. This seemed to work fairly well, although large numbers of neighbors increased the leave-one-out error.

The second version used an exponential drop-off:

$$w_i = \frac{1}{\exp[d(\mathbf{x}_{new}, \mathbf{x}_i)/f]}$$

where f is the kernel width, a parameter to control the amount of roll-off. The second version leaves more room for tuning, since the parameter f can be chosen freely.

Experience

The k-Nearest Neighbor algorithm is very easy to use, and it is straightforward to understand exactly how a particular classification was made. The algorithm is very fast given the small data set. Furthermore, there are no convergence problems (see also the SVM section).

In general the quality of the output data is good but not great. There are no cases of blatant misclassification, since it looks for labels that have been used before and combines them. On the other hand, that is one of its biggest drawbacks: k-NN in the current version cannot extrapolate beyond what it saw before. If a new material or tool configuration is used that's outside the "known region" of the instance space of the training data, the results will be poor.

One particular problem with the data is that it comes in clusters. During each manual calibration run, five or six training samples were collected. These attribute vectors/labels don't differ much from each other, since they have the same material and the same nozzle. The k-NN tends to lock onto the closest cluster in the instance space and produce labels that are very similar to the ones in that particular cluster, instead of averaging over several clusters. This was especially apparent when using the inverse distance as a weighting function. Small numbers of neighbors produced lower leave-one-out errors, but didn't do much averaging, i.e. "learning". In that case, the algorithm basically works as a database which looks up the closest instance in the training set. On the other hand, large numbers of neighbors generalized too much and the differences between labels started to blur. All classifications looked more or less the same, which leads to very mediocre results.

The exponential weighting method is superior, especially when using a large number of neighbors. Better results were achieved by setting k to the total number of examples in the training set and just tuning the kernel width f.

Description of SVM Approach

Support Vector Machines operate on the principle of calculating an optimal separating surface between data points with different labels, such that similarly labeled data all (or nearly all) lie on one side of the separating surface. SVMs can be used not only for identifying discrete classes, but also can be applied to regression problems, i.e. finding the function f

$$\begin{aligned} &f : X \rightarrow Y \\ &X \in \Re^N, Y \in \Re \end{aligned}$$

given the training data set

$$\begin{aligned} &D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\} \\ &\mathbf{x}_i \in X, y_i \in Y \end{aligned}$$

where N is the dimension of the instance space and l is the number of training examples. A good introduction to this topic can be found in Smola and Schölkopf. The regression takes the form

$$f(\mathbf{x}_{new}) = y = \mathbf{w} \bullet \mathbf{x}_{new} + b$$

The vector **w** is the weight vector, b is the offset. In order to find **w** and b the following optimization problem has to be solved:

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

subject to

$$\begin{aligned} &y_i - \mathbf{w} \bullet \mathbf{x}_i - b \le \varepsilon + \xi_i \\ &\mathbf{w} \bullet \mathbf{x}_i + b - y_i \le \varepsilon + \xi_i^* \\ &\xi_i, \xi_i^* \ge 0 \end{aligned}$$

with C being the weight on the slack variables and ξ_i, ξ_i^* being the slack variables. The parameter ε represents the distance from the linear classifier (as defined by **w** and b) up to which values of y_i are considered "good". If y_i differs by more than ε from $\mathbf{w} \bullet \mathbf{x} + b$, then additional cost is added to the minimized function. This also explains why there are two slack variables: one is for the positive, the other one for the negative deviation from the linear classifier.
In practice, the dual of this problem is solved and the regression takes the form

$$y = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)k(\mathbf{x}_i, \mathbf{x}_{new}) + b$$

where l is the number of support vectors, α_i and α_i* are the Lagrange multipliers of the trained SVM, $\mathbf{x}_i$ are the support vectors, and b is the offset. The function $k(.)$ is the kernel.

Implementation

An existing C-library SVM implementation (SVMlight, see Joachims) is used for the training and regression. A wrapper function was written that handles the input and output to SVMlight, i.e. it prepared the input files and read the data from the output files, as well as making function calls to the SVM library with the appropriate parameters, e.g. the kernel type.

As opposed to kNN, SVMs in regression mode have only one output. In order to compute the full control vector **y** with 12 components, 12 SVMs were run in parallel. This approach is justified, since the entire control law will be modeled by the SVM as a black box. Therefore any coupling between the control parameters will implicitly be taken care of. The full regression is thus:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{12} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 \bullet \mathbf{x}_{new} + b_1 \\ \vdots \\ \mathbf{w}_{12} \bullet \mathbf{x}_{new} + b_{12} \end{bmatrix}$$

where $\mathbf{w}_i$ and b_i are the weight vector and offset of the i^th SVM. All SVMs were trained on the same attribute vectors $\mathbf{x}_i$, but different labels y_i (depending on the control parameter). Note that the labels y_i are scalars.

For some reason, the built-in "leave-one-out error" (see below) computation in SVMlight did not work for regression and so the wrapper function includes that functionality. The wrapper function reads the entire data set, extracts the line with the example that is to be left out and writes both the single example and the rest of the training data to two different files. Then the SVMs are trained, the example classified and the output compared with the original label.

Experience

The experience with SVMs was very good. The trained SVMs are able to produce control parameters with small leave-one-out errors and tolerable results when testing them on the physical machine. SVMs are remarkably superior to k-NN when dealing with novel materials and/or nozzles.

One observed problem was the convergence of the SVMs when training them for a leave-one-out error computation. It turned out that SVM light is very sensitive to different values of C or higher order Kernels. Usually the training of a single SVM takes only a fraction of a second, but from time to time the algorithm just failed to converge and kept running until the user stopped it. Adjusting the parameter "-e" (error for termination criterion) helped in most cases.

Results

A standard method to compare the performance of machine learning algorithms is to compute the leave-one-out errors. In this test, one data sample gets removed from the training data and the machine learning algorithm is trained on the remaining data. The trained algorithm is then used to classify the left out example and the results are compared to the original label. Ideally, the difference should be zero. This test is repeated for each sample in the data set. The average error is the leave-one-out error.
Figure 4 to Figure 6 show the normalized (0 to 1) leave-one-out errors for the following algorithms:

- 1-NN, weight equal to inverted distance (1-NN, inv)
- 5-NN, weight equal to inverted distance (5-NN, inv)
- 10-NN, weight equal to inverted distance (10-NN, inv)
- 1-NN, exponential weight, f=1 (1-NN, exp 1)
- 5-NN, exponential weight, f=1 (5-NN, exp 1)
- 49-NN, exponential weight, f=1 (49-NN, exp 1)
- 63-NN, exponential weight, f=0.1 (63-NN, exp 0.1)
- 63-NN, exponential weight, f=0.01 (63-NN, exp 0.01)
- 63-NN, exponential weight, f=10 (63-NN, exp 10)
- 63-NN, exponential weight, f=100 (63-NN, exp 100)

- SVM, C = default, 1st order polynomial kernel (SVM, C=def, exp=1)
- SVM, C = 1, 1st order polynomial kernel (SVM, C=1, exp=1)
- SVM, C = 0.1, 1st order polynomial kernel (SVM, C=0.1, exp=1)
- SVM, C = default, 2nd order polynomial kernel (SVM, C=def, exp=2)

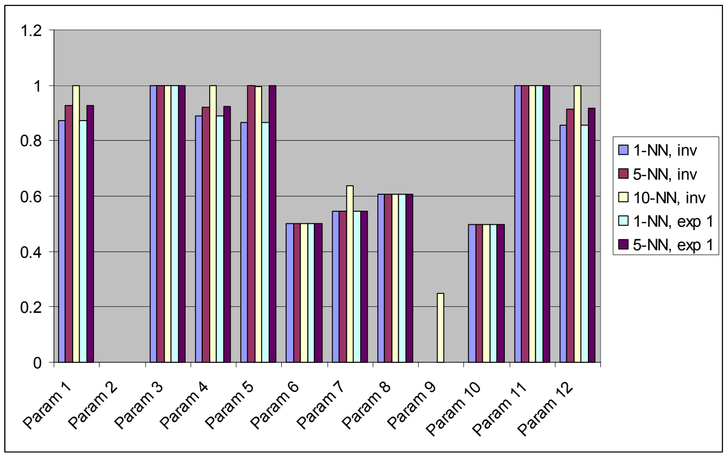

Figure 4. Leave-one-out errors

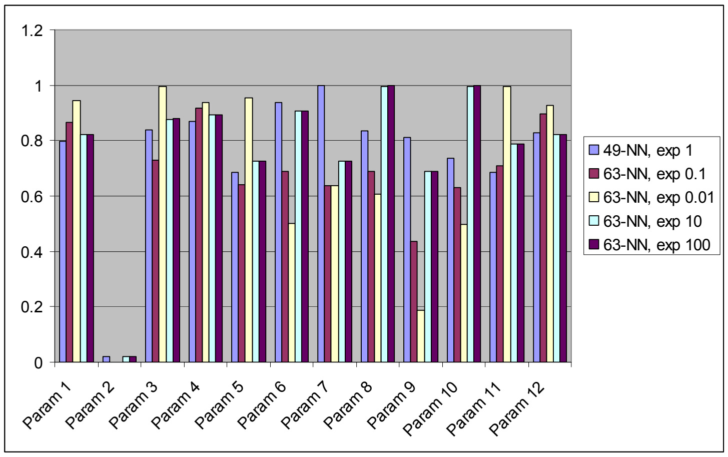

Figure 5. Leave-one-out errors 2

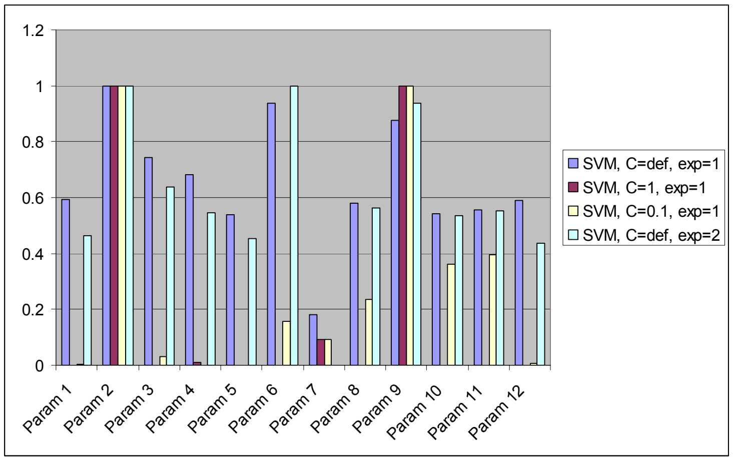

Figure 6. Leave-one-out errors 3

Table III. Calibration Time Savings

Manual Calibration (63 calibrations, 9 materials)	Mean	Std. Dev.
Machine time (h:m)	0:28	0:23
Operator time (h:m)	1:02	0:43
Iterations	11.89	4.94
64-NN-Assisted Calibration (1 material)		
Machine time (h:m)	0:16	
Operator time (h:m)	0:30	
Iterations	4	

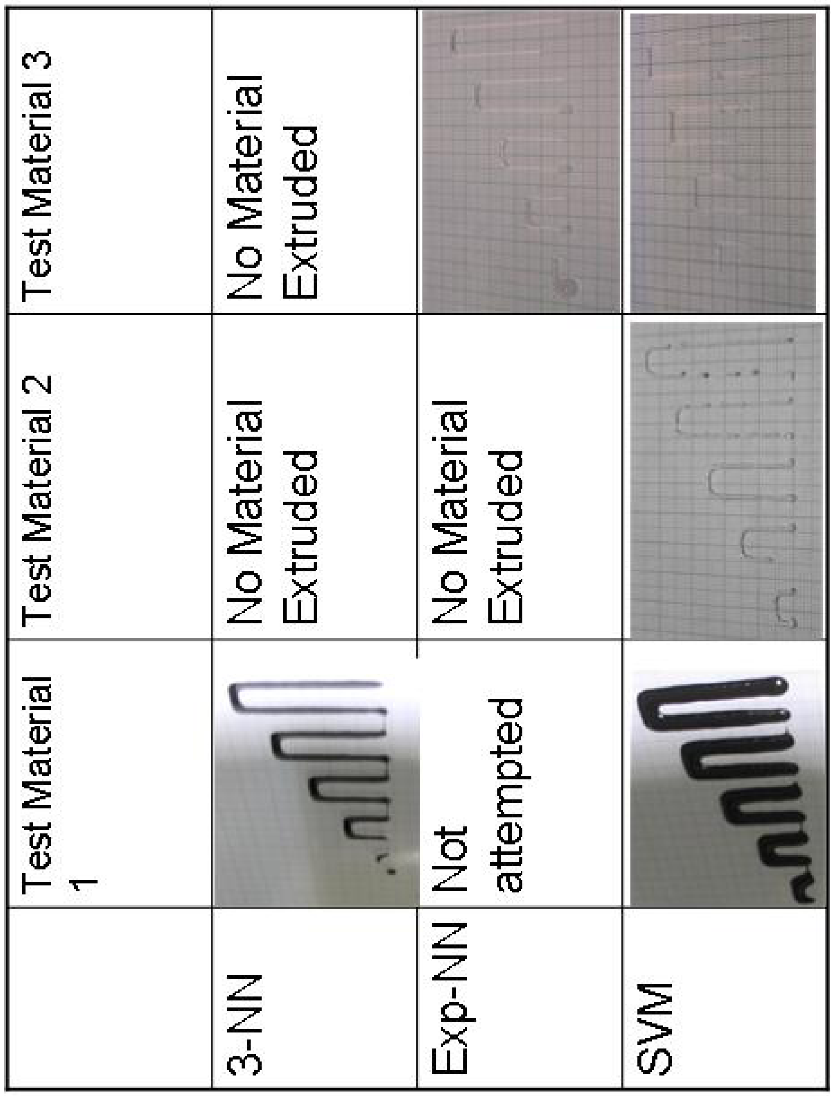

Figure 7. Test patterns produced by automatically generated control parameters

Conclusion

The time and effort required to gather data for these experiments, hence the small training data set, has limited the statistical significance of any conclusions. Qualitatively, the SVM algorithm is superior to k-NN with materials not in the training data set. This qualitative observation is supported by the leave-one-out errors, as seen in Figure 4, Figure 5, and Figure 6. Experience of the operator with manual calibrations showed that parameters 2, 6, 7, and 10 are the least important. These are the ones where k-NN has the lowest errors, whereas the SVM has smaller errors for those parameters which have larger impact on the manual calibrations. As Figure 7 indicates, the k-Nearest neighbor algorithm often generated control parameters that failed to generate any discernable material deposition at all, while the SVM always produced results at least as good as the first guess of a human operator. For the time being, using learning algorithms to produce control parameters has brought no benefit to the deposition performance of the system, and does not yet allow for complete automation of the calibration process. On the other hand, employing the learning algorithms to produce an initial guess at control parameters seems to decrease the time required to obtain satisfactory manual calibration by as much as a factor of 2 (Table III). If this bears out in more extensive testing, the savings in operator time, not to mention chemical waste are very significant. This benefit alone warrants permanent inclusion of the learning algorithms into the software of the system. It remains to be seen whether or not, given a sufficiently large training data set, the learning algorithms will be able to produce completely satisfactory or superior deposition performance without manual intervention.

Future work will include the exploration of intelligent weighting schemes for attributes, such as described by Domingos. These will be explored in the context of ongoing use of the system. Such automated weighting schemes have the added benefit of indicating which attributes are most significant in generating classifications, guiding the selection of additional attributes, and removal of redundant attributes. A neural network will also be explored as an alternative to the learning algorithms explored here.

References

Anguita, D., A. Boni, L. Tagliafico. *SVM performance assessment for the control of injection moulding processes and plasticating extrusion*. International Journal of Systems Science, 2002, Volume 33, number 9, pages 723-735

Domingos, F., *Control-Sensitive Feature Selection for Lazy Learners.* Artificial Intelligence Review, 1997. **11**(1-5): p. 227-253.

Joachims, T., *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999

Mitchell, T.M., *Machine Learning*, McGraw Hill, 1997

Smola, A.J. and B. Schoelkopf. A *tutorial on support vector regression*. NeuroCOLT2 Technical Report NC2-TR-1998-030, 1998

Tobin, K.W., S.S. Gleason, and T.P. Karnowski. *Adaptation of the fuzzy k-nearest neighbor classifier for manufacturing automation*. in *Machine Vision Applications in Industrial Inspection VI San Jose, CA, USA 27 Jan. 1998*. 1998: SPIE Press.

Wettschereck, D., D.W. Aha, and T. Mohri, *A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms.* Artificial Intelligence Review, 1997. **11**(1-5): p. 273-314.