Detection of Contact Force Transients in Robotic Assembly

Stolt, Andreas; Linderoth, Magnus; Robertsson, Anders; Johansson, Rolf

Published in:
2015 IEEE International Conference on Robotics and Automation

DOI:
10.1109/ICRA.2015.7139293

2015

Document Version:
Peer reviewed version (aka post-print)

Detection of Contact Force Transients in Robotic Assembly

Andreas Stolt, Magnus Linderoth, Anders Robertsson, Rolf Johansson

***Abstract*— A robotic assembly task is usually implemented as a sequence of simple motions, and the transitions between the motions are made when some events occur. These events can usually be detected with thresholds on some signal, but faster response is possible by detecting the transient on that signal. This paper considers the problem of detecting these transients. A force-controlled assembly task is used as an experimental case, and transients in measured force/torque data are considered. A systematic approach to train machine-learning based classifiers is presented. The classifiers are further implemented in the assembly task, resulting in a 15 % reduction of the total assembly time.**

I. INTRODUCTION

Industrial robots are usually position-controlled, i.e., controlled to follow predefined trajectories. The robots do the trajectory following very well, with very high repetitional accuracy and with high speed. But not all types of tasks are suitable for this control strategy. Tasks that require interaction with the environment, such as assembly, would require a very high accuracy of the robot and the calibration of its work cell to be able to accomplish the tasks using position control. Small uncertainties in position may lead to very large forces and potentially damaged equipment. An alternative strategy is to introduce additional sensors, such as a force sensor. Uncertainties can then be compensated for by sensing the contact forces.

A force-controlled assembly task can be implemented as a sequence of simple motions, usually in the form of search motions that are used to resolve uncertainties in the task. An example of this strategy was presented in [13]. The conditions for switching between the simple motions is that some event occurs, e.g., that a contact is established. These events can often be detected by using a threshold on the measured force/torque data. What really is detected is a transient in the measured data. In many cases, the events can be detected from other transients as well, occuring before the force/torque build up detected by the threshold. This effect can be exploited to reduce the total assembly time. This paper considers the problem of detecting transients in force/torque data in the context of force-controlled assembly, but it could also be applied to any other signal in a different context. A systematic approach for training machine-learning based classifiers will be used to accomplish this. A part of the assembly of an emergency stop button will be used as an experimental case, where the setup can be seen in Fig. 1.

Andreas Stolt, andreas.stolt@control.lth.se, Magnus Linderoth, Anders Robertsson, and Rolf Johansson are with the Department of Automatic Control, LTH, Lund University, Sweden.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement No 230902 - ROSETTA. The authors are members of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University.

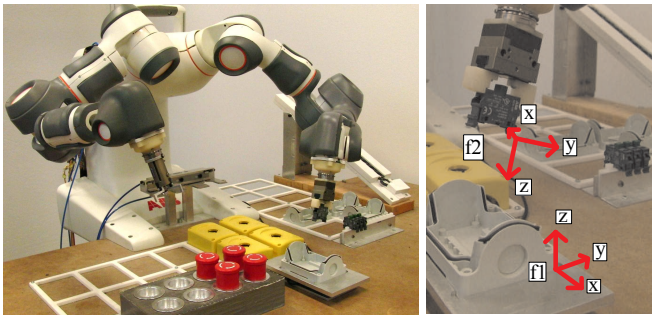

Fig. 1. Left: The experimental setup used for the experiments. Right: The coordinate frames that were used to specify the task.

Machine learning approaches within assembly has previously been used in [11], where a single-axis force sensor was used to detect failures in an assembly scenario. A somewhat different approach was applied in [7], where a Hierarchical Dirichlet Process Hidden Markov model was used to monitor an assembly task, for detection of errors. Another approach to monitoring assembly tasks was presented in [12], where a hierarchical taxonomy was used to monitor a snap assembly task. The force/torque signatures were investigated with respect to relative changes in several different layers that could be used to discriminate nominal behavior from errors. Machine learning has also been used for detecting failures, e.g., [4] and [10], where support vector machines were used to detect tool breakage in milling processes.

II. ASSEMBLY SCENARIO

The assembly scenario considered in this paper is to assemble the internals of an emergency stop button, see the parts in Fig. 1. An electrical switch should be snapped into place in a slot in the bottom box. The scenario and its implementation have previously been presented in [13], [14]. The task is modeled using the iTaSC framework [6]. The feature coordinates used to describe the task are the three translations (x,y,z) in frame $f1$, see right part of Fig. 1, and the Euler ZYX-angles (ϕ,θ,ψ) describing the reorientation from $f1$ to $f2$, the frame attached to the switch in the gripper. The task is accomplished by performing a number of guarded search motions, i.e., search motions that are ended when the corresponding contact force is sensed.

The assembly sequence is controlled by a state machine, with force/torque measurements triggering state transitions. Simple threshold detectors for some force or torque channels can be used for all transitions. However, the overall assembly time can be decreased by instead detecting transients in

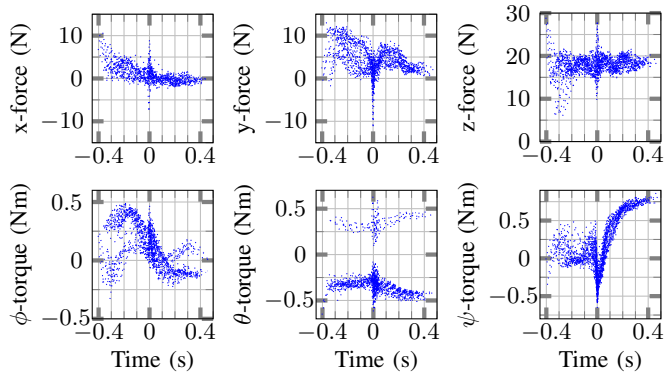

Fig. 2. The recorded data from the snap motion. To increase the readability, all sequences have been shifted such that the transient occurs at $t = 0$ s. The force/torque directions refer to the feature coordinates defined in Sec. II.

the force/torque data. Another positive effect is that the magnitude of the forces in the task can be decreased, as the largest forces usually relate to the thresholds used, which usually need to be large to get robustness. In one of the states, the switch slides along a surface to find the slot and it is finished when contact is established with an edge. A possible improvement is to instead detect the transient arising from the switch sliding into the slot, which happens before contact with the edge is established. This motion will be called the slide motion in the rest of the paper. Another improvement is for the state where the switch is snapped into place, which can be detected from the snap transient, instead from when contact is established with the bottom of the box. This motion will be called the snap motion.

III. PROCEDURE

To get a classifier for a transient from an assembly sequence, several steps have to be taken. First, data have to be gathered and preprocessed, then the classifier has to be trained. The procedure is described in this section, and the classifiers considered are introduced. For method description, the snap motion transition is used as an example.

A. Data gathering

In a production setting, each emergency stop button that the robot assembles will have new components. To mimic this scenario, it would therefore be a benefit to have a large number of different switches available, recording one assembly operation from each one of them. In the current setup, however, only 12 switches were available, and only using them once would give a rather small training data set. Therefore, five recordings were made for each switch, and the total number of recordings made was thus 60.

The training data were divided into two halves, one to be used for training and one to be used for validation. The division was made such that the training and the validation data set did not contain any recordings from the same switch.

The transients considered, during the slide motion and the snap motion, were manually marked in each recording. The force/torque data from the recorded snaps are displayed in Fig. 2. It might seem to be an easy task to detect a snap in these data, but the classifier should be able to do it with only a subset of the data available. The smaller subset, the better, as that would decrease the computational burden, and also decrease the delay from when the snap occurs until the detection is done.

B. Data pretreatment

The data contained six different channels of force and torque data, i.e., the three force directions and the three torque directions as displayed in Fig. 2. A subset of these channels were used by the classifier. A number n_{pre} of samples before the transient and a number n_{post} of samples after the transient were taken from the channels that were used and were put after one another in a vector, which will be called a sequence. If the number of channels used was n_{ch}, then a sequence will be a vector in $\mathbb{R}^{n_{ch}\cdot(n_{pre}+1+n_{post})}$. The job of the classifier is thus to determine if a sequence contains a transient or not.

Before forming the sequence vector, the mean value was removed from each channel. This should make the classifier independent to any offset force/torque, e.g., the offset seen in the θ-torque in Fig. 2. The data were also prescaled such that all force/torque components got approximately the same magnitude. This should make the problem of training the classifiers better numerically conditioned.

C. Data for training

Every recording contained one interesting transient and a lot of background sequences, i.e., data not containing a transient. The number of background sequences in each recording is approximately equal to the length of the recording, which was about 100 for the snap transient example. Using all background sequences would give an unreasonably large training data set. Instead, a number of N (chosen to be 20) randomly selected background sequences and the sequence containing the transient were used from each recording. These selected sequences were used for training the classifier, and then it was validated against all background sequences in the recordings chosen for training. If any misclassified sequences were found, these were included into the training sequences and the training of the classifier was performed once again. This procedure was iterated until no more misclassified sequences in the recordings chosen for training were found, not already included in the training sequences, or until a maximum number of iterations was reached (10 was used).

There is a risk, however, that this strategy will lead to different classifiers depending on the initial choice of background sequences. To average this effect out, for every choice of parameters, the classifier was trained a number of N_{seeds}, usually around 10-20, each time with a different configuration of the random number generator used for choosing the initial background sequences. The score for the given parameters was then taken as the average of the N_{seeds} performed trainings.

D. Cost function

It might be more important not to make any false positive classifications than any false negative classifications, or the other way around. In the experimental scenario considered, classifying a background sequence as a transient would be worse than missing a transient, as backup classifiers based on thresholds existed. This asymmetry in the problem can be included by using the following cost function for training

$$J = cost_{FP} \cdot n_{FP} + cost_{FN} \cdot n_{FN} \tag{1}$$

where $cost_{FP}$ and $cost_{FN}$ are the costs associated with false positives and negatives, respectively, and n_{FP} and n_{FN} are the number of false positives and false negatives.

E. Classifiers considered

Four different classifiers were applied to the data, a simple least-squares classifier, two different versions of support vector machines, and a boosting classifier. A detailed description of the methods can, e.g., be found in [1], and a summary of them can be found below.

1) Least-squares: The first classification method used was based on the least-squares (LS) method. The model used was

$$y(x) = w^T x + w_0 = \begin{bmatrix} w^T & w_0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \tilde{w}^T \tilde{x} \tag{2}$$

where x denotes the data sequence, $\tilde{w}$ the parameters, and y should be 1 for a transient and -1 for the background. The classifier is based on finding a hyperplane that separates the transients from the background.

All training data can be described by

$$\underbrace{\begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}^T}_{Y} = \underbrace{\begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_n \end{bmatrix}^T}_{\tilde{X}} \tilde{w} \tag{3}$$

and by using a sum-of-squares error function

$$J(\tilde{w}) = \sum_{i=1}^{n} c_i (y_i - \tilde{x}_i^T \tilde{w})^2 = (Y - \tilde{X}\tilde{w})^T C (Y - \tilde{X}\tilde{w}) \tag{4}$$

where C is a diagonal matrix describing the cost for the element i, the optimal $\tilde{w}$ is given by

$$\tilde{w} = \left(\tilde{X}^T C \tilde{X}\right)^{-1} \tilde{X}^T C Y \tag{5}$$

The cost matrix C was chosen such that $c_i = cost_{FP}$ if $y_i = -1$ and $c_i = cost_{FN}$ if $y_i = 1$. This gives an approximation of the cost function (1). Classification is performed by calculating $y(x)$, and the classification boundary is 0, i.e., $y(x) > 0$ means that x is classified as a transient.

2) Support vector machines: The simplest form of support vector machines (SVM) uses the model (2), but instead of choosing the parameters based on the error function (4), the parameters are chosen such that the resulting hyperplane is the one with the largest margin to the different classes of data. The problem can be stated as the optimization problem

$$\begin{array}{lll} \underset{\text{over } w,\zeta}{\text{minimize}} & P \sum_{i=1}^{n} c_i \zeta_i + \frac{1}{2}\|\tilde{w}\|_2^2 & \\ \text{subject to} & y_i y(x_i) \geq 1 - \zeta_i & , \quad i = 1, \dots, n \\ & \zeta_i \geq 0 & , \quad i = 1, \dots, n \end{array} \tag{6}$$

where ζ_i are slack variables that allow for misclassifications, c_i is the cost for the misclassification, and $y(x)$ is defined as in (2). The cost is chosen as $c_i = cost_{FP}$ if $y_i = -1$ and $c_i = cost_{FN}$ if $y_i = 1$, i.e., an approximation of (1). The sum of the misclassifications is penalized by the parameter P. This parameter will control the trade-off between classifying everything correct versus having a large margin with possibly a few incorrect classifications. The optimization problem (6) is convex, and it can therefore be solved efficiently with guarantuee of finding global minimum. This classifier will be called the primal SVM.

By considering the dual formulation of (6), it can be shown [1] that the following problem is equivalent to solve

$$\begin{array}{lll} \underset{\text{over } a}{\text{minimize}} & \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j k(x_i, x_j) & \\ \text{subject to} & 0 \leq a_i \leq P c_i \quad , & i = 1, \dots, n \\ & \sum_{i=1}^{n} a_i y_i = 0 & \end{array} \tag{7}$$

The function $k(x_i, x_j) = x_i^T x_j$ is a kernel function, which can be replaced by any other positive definite kernel function.

The formulation (7) was used together with a Gaussian kernel, $k(x_i, x_j) = e^{-(x_i - x_j)^T (x_i - x_j)/l}$, where $l > 0$ is a parameter that corresponds to the width of the kernel. The classification function is given by

$$y(x) = \sum_{i=1}^{n} a_i y_i k(x, x_i) + b \tag{8}$$

Most of the a_i are zero, and the non-zero ones correspond to the support vectors x_i. The parameter b is calculated through

$$b = \frac{1}{N_M} \sum_{i \in M} \left(y_i - \sum_{j \in S} a_j y_j k(x_i, x_j) \right) \tag{9}$$

where S is the set of all support vectors, M is the set of indices of data points having $0 < a_i < P c_i$, and N_M is the number of elements in M.

3) Boosting: This is a classifier that uses the result from many simple classifiers, called weak classifiers, to make the final classification. The performance of the final classifier is usually significantly better than any of the weak classifiers. During training, weights are used to give data points that are hard for the weak classifiers to correctly classify more importance. The final classifier is a weighted average of the weak classifiers, which in turn has been trained with differently weighted training data. The algorithm used is called AdaBoost and was first proposed in [8].

The weak classifiers considered were threshold detectors for one of the coordinate axes. For each weak classifier, all axes were considered, and the one giving the best separation, concerning the cost function (1) and the importance weights, was chosen.

F. Training procedure

The first part of the training procedure consisted in deciding which channels to use, the number n_{post} samples after the transient, and the number of n_{pre} samples before the

transient. This is a difficult task, as the available parameter space is quite large. A proposal is that it can be performed in three steps, by first selecting the channels to use, then the number of samples after the transient, and finally the number of samples before the transient.

The value of the parameters to be used can be found by choosing one classifier and evaluate the performance on the validation data for the different parameters. It is also possible to use a combination of classifiers.

1) Varying the channels to use: The first step was to vary the channels used. The values of n_{pre} and n_{post} were chosen to be large (30 was used), such that the limiting factor for the total information available was in the choice of channels. All possible combinations of channels were evaluated.

One choice for continuing from this step would be to choose the combination of channels giving the best result, but this would most likely mean that all channels should be chosen. That is undesirable, as the complexity of the classifier increases with the number of channels used. Instead, the best choices for one to four channels were chosen and used for the following steps.

2) Varying n_{post}: The number of samples used after the transient should be kept at a minimum, not delaying the detection of the transient more than absolutely necessary. This was done by varying n_{post} from zero to a large value (30 was used), while fixing n_{pre} to a large value (30 was used). In other words, all information before the transient was kept while the amount of information after the transient was varied. The evaluation was based the performance measure J_1, the sum of J defined in (1) plus a quadratic penalty term.

$$J_1 = J + p_1 n_{post}^2 \tag{10}$$

3) Varying n_{pre}: The trade-off for the number of samples used before the transient is about performance versus complexity of the classifier. The parameter n_{pre} was varied from zero to a large value (30 was used), with n_{post} given by the choice from the previous step. The evaluation was similar as in the previous step, but now with the linear penalty $p_2 n_{pre}$. This should reflect that it is not as costly to have some more samples before the transient than after.

4) Training of other classifiers: With all parameters concerning how much data the classifier should use decided, choosing the classifier specific parameters remained. Neither the LS nor the boosting classifier ($N = 50$ weak classifiers were used) had any specific parameters, but the SVM classifiers considered had some further parameters to tune. The primal formulation has the so called misclassification penalty, P in (6). The search for the best choice of this parameter was performed in two steps. First the parameter space was coarsely gridded with logarithmic spacing. A classifier was trained in every grid point and evaluated against the validation data. The second step consisted in refining the grid around the best grid points from the first step and redoing the training in the new grid points.

The dual SVM classifier had the width of the Gaussian kernel as an extra parameter, besides the misclassification penalty. The parameter space was therefore two-dimensional. The training procedure was performed in the same way as for the primal SVM classifier, but with an even coarser grid to begin with to reduce the computational load.

G. Implementation

The training procedure described in this section was implemented in Matlab. The training of the SVM classifiers was performed by using CVX, a package for specifying and solving convex programs [5], [9].

IV. EXPERIMENTAL RESULTS

A. Robot system

The robot used in the assembly scenario was the ABB YuMi (previously known as FRIDA). It is a dual-arm manipulator with 7 joints in each arm. The robot was controlled with the IRC5 control system, extended with an open control system [2], [3], which made it possible to modify the references for the low-level joint control loops. An ATI Mini40 6-DOF force/torque sensor was mounted on the table beneath the fixture for the bottom box. The sensor was used without any configured low-pass filter, and it was sampled at $250\,\mathrm{Hz}$. A photo of the experimental setup is given in Fig. 1.

B. Snap detection

Three different choices of classifiers were used for the initial training phase, namely LS, boosting, and the combination of LS and boosting. These classifiers were used as they were both quite computationally cheap, and that they had no extra parameters to tune. The first parameter to be decided was which channels in the measured force/torque data to use. The result on validation data for all combinations of channels is displayed in Fig. 3. It can be seen that the variation in the performance decreases when the number of channels is increased, which is intuitive as more information becomes available. For the LS classifier, however, the performance seems to deteriorate when the number of channels becomes large. This is probably due to the classifier being over-fitted, and thus failing on validation data. The best choice for one to four channels was used for further training.

The next parameter to decide was n_{post}, the number of samples to use after the snap. The result on validation data when this parameter was varied from 0 to 30 for the different choices of channel selections is displayed in the left diagram in Fig. 4. The training based on the combination of LS and boosting has been omitted from the figure to increase readability. The stars ($*$) show the cost according to (1), and the rings ($\bigcirc$) the total performance measure (10). The constant p_1 was chosen as 0.09, and it was chosen such that it gave a quite low value for n_{post} for this case. The LS classifier behaves as expected, i.e., the performance increases with n_{post}. The boosting classifier, on the other hand, first shows a performance degradation before giving perfect classification for $n_{post} \geq 14$. For all cases the n_{post} given by the performance measure was a low value.

The final parameter to decide was n_{pre}. The result from varying this parameter from 0 to 30, with the other parameters decided by the previous experiments, is displayed

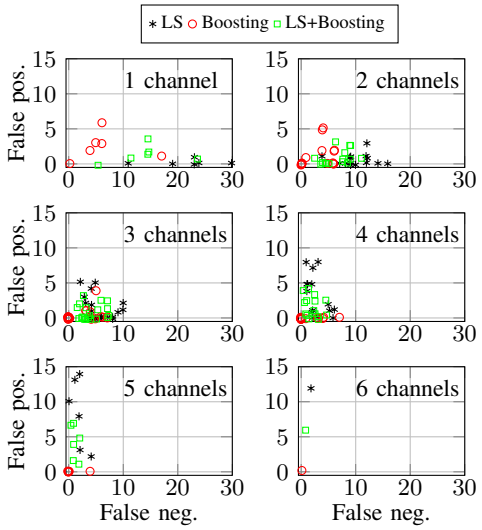

Fig. 3. The classification result when all different combinations of channels were tested. A small random perturbation has been added to every marker, such that it should be possible to see where multiple configurations led to the same result.

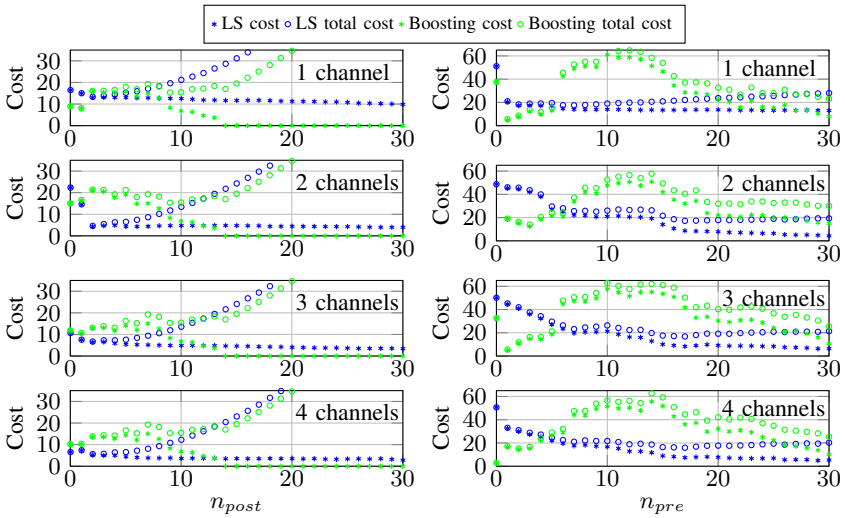

Fig. 4. The result when the number of samples after the transient, n_{post} (left column), and before the transient, n_{pre} (right column), was varied. The cost refers to the cost function (1) with $cost_{FP} = cost_{FN} = 1$. The total cost also contains the extra penalty term for large n_{post} or n_{pre}.

$cost_{FP} = cost_{FN}$

Initial training	Least-squares				Boosting				Least-squares + Boosting			
Channels	$[\psi]$	$[x\ \theta]$	$[x\ \phi\ \theta]$	$[x\ y\ \phi\ \theta]$	$[\psi]$	$[\theta\ \psi]$	$[x\ \phi\ \psi]$	$[x\ z\ \phi\ \psi]$	$[\psi]$	$[x\ \theta]$	$[x\ z\ \psi]$	$[x\ y\ \theta\ \psi]$
n_{pre}	7	17	17	17	1	3	1	0	1	17	2	1
n_{post}	2	2	2	2	1	0	1	0	1	2	2	1

$cost_{FP} = 10 cost_{FN}$

Initial training	Least-squares				Boosting				Least-squares + Boosting			
Channels	$[\psi]$	$[y\ \psi]$	$[y\ \theta\ \psi]$	$[x\ y\ \theta\ \psi]$	$[\psi]$	$[z\ \psi]$	$[y\ z\ \psi]$	$[y\ z\ \phi\ \psi]$	$[\psi]$	$[y\ \psi]$	$[y\ \theta\ \psi]$	$[x\ y\ \theta\ \psi]$
n_{pre}	3	5	7	7	8	4	5	8	3	9	4	2
n_{post}	1	2	3	3	15	15	16	15	14	14	14	14

TABLE I

RESULT OF INTIAL TRAINING. THE TOP PART SHOWS THE SYMMETRIC CASE AND THE BOTTOM PART THE ASYMMETRIC CASE.

in the right diagram in Fig. 4. The boosting classifier still has some problems in the middle region, while the LS classifier shows a monotone-like performance increase. The performance measure constant p_2 was chosen as 0.5, such that it gave a slowly increasing cost for growing n_{pre}. The final parameter choices are summarized in the upper part of Table I.

The considered scenario is in some sense asymmetric, as was concluded in Sec. III-D. To see the effect of including the asymmetry, the entire training procedure was performed one more time, with $cost_{FP}$ being 10 times as large as $cost_{FN}$. The final parameter choices are listed in the lower part of Table I. It can be seen that the parameters for the initial training with LS are similar to the symmetric case, but that n_{post} is much larger for the other initial training methods. The difference is that the LS classifier was very careful and made almost no false positive classifications, but the other two initial training methods had some false positives for low values of n_{post}. Only a few false positives gave a large cost, so large that not even the quadratic penalty could force a low value of n_{post}. For the initial training with LS, the quadratic penalty was the dominating term as there were mostly false negatives, and the result was a low value for n_{post}.

Parameters for the SVM classifiers were found by gridding the parameter space, as described in Sec.III-F.4. The best classifiers for all choices of number of channels and initial training are listed in Table II. It can be seen that the best performance was achieved with the asymmetric approach, giving perfect classification. The main difference between the two approaches was the choice of n_{post} versus n_{pre}. In the symmetric case, n_{post} was small and n_{pre} usually larger, resulting in a classification with only a small delay. In the asymmetric case, however, n_{post} was large and n_{pre} small when boosting and the combination of LS and boosting was used for the initial training, giving a longer delay, but on the other hand giving perfect classification. Depending on sampling time, this delay might be of more or less importance. In this scenario, with a sampling time of 4 ms, the extra delay in the asymmetric case becomes 40-50 ms, which is quite short and this suggests that the penalty for choosing n_{post} might have been a bit too large.

$cost_{FP} = cost_{FN}$

	Least-squares initial training				Boosting initial training				Least-squares + Boosting initial training			
	n_{ch}	P	l	Average cost	n_{ch}	P	l	Average cost	n_{ch}	P	l	Average cost
Primal SVM	4	0.0182	-	3.65	3	1.33	-	7.0	2	0.294	-	3.85
Dual SVM	2	6.91	67.17	1.80	4	7.97	7.97	2.80	2	6.91	67.17	1.80

$cost_{FP} = 10cost_{FN}$

	Least-squares initial training				Boosting initial training				Least-squares + Boosting initial training			
	n_{ch}	P	l	Average cost	n_{ch}	P	l	Average cost	n_{ch}	P	l	Average cost
Primal SVM	2	0.571	-	10.60	2	0.00750	-	5.80	4	0.0330	-	3.20
Dual SVM	2	320.75	668.57	3.00	2	28.63	537.00	0.0	1	33.00	320.75	0.0

TABLE II

RESULT OF SVM TRAINING. THE TOP PART SHOWS THE SYMMETRIC CASE AND THE BOTTOM PART THE ASYMMETRIC CASE.

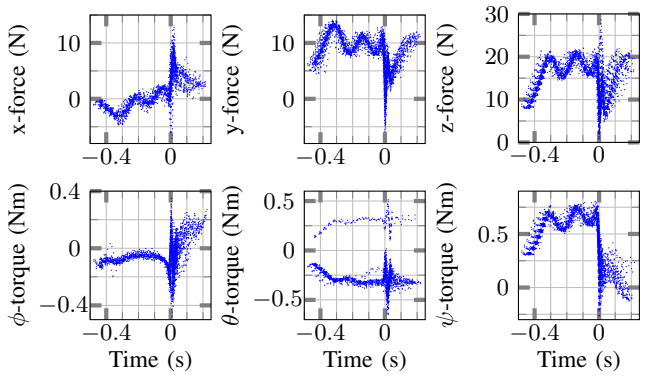

Fig. 5. The recorded data from the slide motion. All sequences have been shifted such that the transient occurs at $t = 0$ s. The force/torque directions refer to the feature coordinates defined in Sec. II.

The results from the SVM training give no unanimous answer to how many channels that should be used, as there were classifiers performing the best in all cases, with one to four channels used. A feasible strategy could be the one taken in this paper, i.e., evaluate a number of different choices of number of channels and choose the one performing best. The best option for the initial training was the combination of LS and boosting, giving the best performance in all cases except for the primal SVM classifier in the symmetric case.

The reduction in assembly time for using the transient detection compared to the threshold detector earlier used was approximately $0.31\,\mathrm{s}$, with a standard deviation of $0.064\,\mathrm{s}$.

C. Slide motion transient

The training procedure was applied also to the slide motion transient, see the recorded transients in Fig. 5. Based on the results from the snap detection, the combination of LS and boosting was used for the initial training. The same parameter values for p_1 and p_2 were used. For this transient, both the symmetric and the asymmetric case gave low values for n_{post} and somewhat larger for n_{pre}. The final SVM training resulted in perfect classification for both the primal and the dual variants. This is an indication of that this transient was easier to detect than the snap.

The reduction in assembly time for using the transient detection compared to the threshold detector earlier used was approximately $0.18\,\mathrm{s}$, with a standard deviation of $0.041\,\mathrm{s}$.

D. Real-time implementation

Classifiers for the two transients were implemented and used for executing the assembly operation. The classifiers chosen for implementation were both dual SVM classifiers that resulted in perfect classification for training and validation data. The asymmetric formulation was used, and the previously used thresholds were used as backup classifiers. The number of support vectors was 35 for the slide motion classifier and 53 for the snap motion classifier. As was mentioned in Sec. III-A, all available switches were used for training/validation, so the experiments were performed using a subset of these switches, namely two switches from the training data set and two from the validation data set.

An unforeseen problem occurred when the classifiers were both used in the assembly operation. The shape of the force/torque signals directly following the slide motion transient were similar to the snap motion transient, cf. Figs. 2 and 5. These data were not used for training the snap classifier, and it was therefore confused when faced to these data and the result was quite many false positive classifications right in the beginning of the snap motion. One solution to this problem would be to include these data for training. Another solution was to wait for the slide motion transient to die away before the snap classifier was activated. In Fig. 5 it can be seen that the slide motion transient has died away after around $0.1\,\mathrm{s}$, and in Fig. 2 it can be seen that it takes at least $0.3\,\mathrm{s}$ from the snap motion is started until the snap occurs. The time to wait was therefore set to $0.1\,\mathrm{s}$.

The assembly operation was performed 20 times. None of the transients were missed and no false positive classifications were made. The total reduction in assembly time by using the classifiers was in average $0.49\,\mathrm{s}$ (standard deviation $0.070\,\mathrm{s}$), which meant a reduction of the total time to insert the switch in the bottom box with approximately $15\,\%$.

V. DISCUSSION

The classifiers considered in this paper were supposed to replace simple threshold detectors that have been used before. The two investigated state transitions showed that there is time to gain, i.e., production can be sped up. The advanced classifier can also be used for improving the robustness, or enabling detection with a lower force at the same robustness as the threshold detectors.

The procedure described in Sec. III-C to pick out background sequences to include into the training data was applied as using all background sequences for training would be too computationally expensive. On the other hand, it was computationally cheap to perform validation on all the background sequences. That made it possible to start with a small set of background sequences, and include more if needed, as this will lead to a more robust classifier.

One of the design goals of the presented procedure was to reduce the number of channels used. This was done by investigating all possible combinations of channels and in the end use the one giving the best result. An alternative could be to employ Principal Component Analysis (PCA), and instead find one or more linear combinations of channels to use. PCA would pick out combinations of channels with large variance, but this might not be the important information for doing classification. It remains as future work to investigate this approach.

The complexity of the classifiers has been kept low, in the sense of using as little data as possible. Aside from keeping the training time from increasing and also making classification faster, it will give some robustness. In a scenario like this, the training data size will be quite small, at least initially. A too complex classifier will then tend to model also the noise, and therefore being a worse classifier when faced to new data. The parameters used for choosing the complexity of the classifiers (p_1 and p_2) in the example scenario may, however, have given too much emphasis on low complexity. This is especially the case for when boosting was used as the initial classifier for the snap, when very few samples before and after the snap were used for the classifier.

In a production setting, all erroneous classifications should be saved, perhaps also the successful ones. These data should be used to redo the training of the classifier to improve performance. With more data available, maybe also the complexity of the classifier can be increased.

Automating a task like the one considered in this paper is difficult. Every new task will be different, with different requirements. But a systematic approach, as was presented, will be a step towards a complete solution. The experiments performed show a proof of concept that it works.

Performing the training procedure takes a really long time, which may take in the order of days on a standard computer, depending on the grid size used in the SVM training. But the training procedure is possible to parallelize and thus to use multiple machines. It would therefore be perfect as a cloud service. Record some training data, send them to the cloud, and receive a classifier in response. With large computational resources it might further be feasible to vary more parameters, e.g., other kernels for the dual SVM classifier, as well as other types of classifiers.

VI. CONCLUSIONS

A systematic procedure for finding machine-learning based classifiers to detect transients in force/torque data was presented, but it would also be feasible to use the method with any other time-series data. A force-controlled assembly task was used as an experimental case to show that the method worked. In the implementation of the assembly task, the classifiers were used to replace simple threshold detectors that were used before. The replacement resulted in the total assembly time being reduced with $15\,\%$.

REFERENCES

[1] C.M. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2006.

[2] A. Blomdell, G. Bolmsjö, T. Brogårdh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, and J.J. Wang. Extending an industrial robot controller–Implementation and applications of a fast open sensor interface. *IEEE Robotics & Automation Magazine*, 12(3):85–94, 2005.

[3] A. Blomdell, I. Dressler, K. Nilsson, and A. Robertsson. Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers. In *Proc. ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, pages 62–66, Anchorage, Alaska, USA, May 2010.

[4] S. Cho, S. Asfour, A. Onar, and N. Kaundinya. Tool breakage detection using support vector machine learning in a milling process. *Int. J. of Machine Tools and Manufacture*, 45(3):241–249, 2005.

[5] Inc. CVX Research. CVX: Matlab software for disciplined convex programming, version 2.0. `http://cvxr.com/cvx`, August 2012.

[6] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *Int. J. Robotics Research*, 26(5):433–455, 2007.

[7] E. Di Lello, T. De Laet, and H. Bruyninckx. Hierarchical Dirichlet Process Hidden Markov models for abnormality detection in robotic assembly. In *Proc. NIPS 2012 Workshop on Bayesian Nonparametric Models (BNPM) For Reliable Planning And Decision-Making Under Uncertainty*, Lake Tahoe, Nevada, USA, Dec 2012.

[8] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. Thirteenth Int. Conf. Machine Learning*, pages 148–156, Bari, Italy, July 1996.

[9] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[10] Y.W. Hsueh and C.Y. Yang. Prediction of tool breakage in face milling using support vector machine. *The International Journal of Advanced Manufacturing Technology*, 37(9):872–880, 2008.

[11] A. Rodriguez, D. Bourne, M. Mason, G.F. Rossano, and J.J. Wang. Failure detection in assembly: Force signature analysis. In *IEEE Conf. Automation Science and Engineering (CASE)*, pages 210–215, Toronto, Canada, Aug 2010.

[12] J. Rojas, K. Harada, H. Onda, N. Yamanobe, E. Yoshida, K. Nagata, and Y. Kawai. A relative-change-based hierarchical taxonomy for cantilever-snap assembly verification. In *Proc. Int. Conf. Intelligent Robots and Systems (IROS)*, pages 356–363, Vilamoura, Portugal, Oct 2012.

[13] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson. Force controlled assembly of emergency stop button. In *Proc. Int. Conf. Robotics and Automation (ICRA)*, pages 3751–3756, Shanghai, China, May 2011.

[14] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson. Robotic assembly of emergency stop buttons. In *Proc. Int. Conf. Intelligent Robots and Systems (IROS)*, page 2081, Tokyo, Japan, Nov 2013. (Video: http://www.youtube.com/watch?v=7JgdbFW5mEg)