Detection and Control of Contact Force Transients in Robotic Manipulation without a Force Sensor

Karlsson, Martin; Robertsson, Anders; Johansson, Rolf

Published in:
2018 IEEE International Conference on Robotics and Automation (ICRA)

2018

Document Version:
Publisher's PDF, also known as Version of record

Detection and Control of Contact Force Transients in Robotic Manipulation without a Force Sensor

Martin Karlsson* Anders Robertsson Rolf Johansson

***Abstract*— In this research, it is shown that robot joint torques can be used to recognize contact force transients induced during robotic manipulation, thus detecting when a task is completed. The approach does not assume any external sensor, which is a benefit compared to the state of the art. The joint torque data are used as input to a recurrent neural network (RNN), and the output of the RNN indicates whether the task is completed. A real-time application for force transient detection is developed, and verified experimentally on an industrial robot.**

I. Introduction

In autonomous robotic assembly, mating of components in a sequence of assembly operations requires in each step a validation procedure. Without such validation, successful mating may not be verified, thus jeopardizing successive assembly steps and, eventually, the successful completion of the entire assembly. Moreover, the time required for validation will limit the speed of the assembly operation. To the purpose of such minimum-time validation, we propose a method based on analysis of force transients in controlled contact operations during robotic assembly. A method to detect force transients during robotic manipulation tasks, such as assembly tasks, is presented and evaluated. In [1], this was achieved by detecting contact force transients induced during the assembly, using a force/torque sensor. This detection reduced the assembly time, compared to using a force threshold. It also removed the necessity to determine any level of the force threshold, which would have required considerable engineering work and explicit robot programming. As compared to using position criteria, the approach in [1] allowed robots to switch between movements at the right moment, despite any position uncertainties, thus avoiding to push unnecessarily hard on work objects or to leave a task unfinished.

Here, we continue the work presented in [1], with the following extensions. In [1], a force/torque sensor was used to measure the contact force/torque. Such sensors and systems are usually expensive, with costs comparable to the robot itself. If attached to the wrist of the robot, it would introduce extra weight that the robot would have to lift and move. Further, some robot models do not support any seamless attachment of such sensors. If the force sensor would be attached to an object in the work space, *e.g.*, a table, this would imply restrictions on where the assembly could take place. In this work, we therefore propose a method based on robot joint torque measurements for the detection, thus avoiding the requirement of a force/torque sensor. This introduces a new difficulty; due to friction in the robot joints, some information is lost when using joint torques as compared to a force/torque sensor. The research presented in this paper also extends [1] by investigating whether a given detection model could generalize to tasks that involve new objects, from which no data have been used to determine the model. In order to investigate whether robot joint torques contain enough information to distinguish transients, we follow a machine learning approach; First, we gather data to determine a recurrent neural network (RNN), which is an artificial neural network specialized in processing sequential data [2], [3], and subsequently we evaluate the performance of the RNN on new data.

The procedure proposed does not rely on any assumption of what tasks or work objects are considered, as long as distinguishable joint torque transients are generated. In this paper, we evaluate the procedure on the manipulation scenarios shown in Figs. 1 and 2. These scenarios also serve as examples of how the procedure could be used in practice.

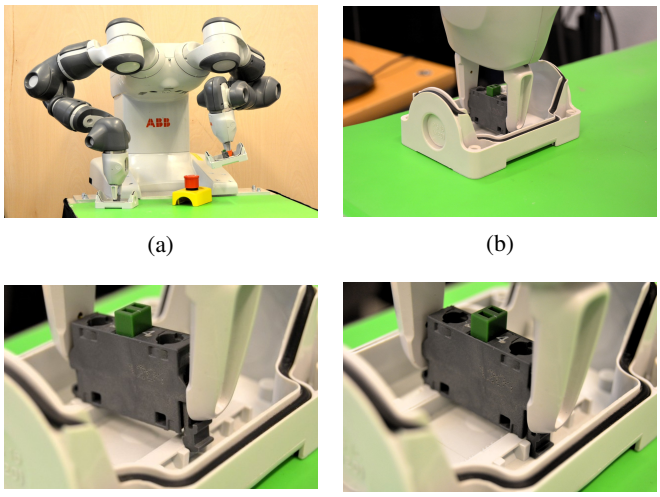
Machine learning for analyzing contact forces in robotic assembly was also applied in [4], where force measurements were used as input to an SVM, to distinguish between successful and failed assemblies. A verification system, specialized in snap-fit assembly, was developed in [5]. Similar to [1] and [4], a force/torque sensor was assumed in [5]. Such a requirement has been avoided in some previous research, by using internal robot sensors instead. For instance, a method to estimate contact forces from joint torques was presented in [6]. Further, force controlled assembly without a force sensor was achieved in [7], by estimating contact forces from position errors in the internal controller of the robot.

II. Method

In this section, the proposed machine learning procedure to determine a force/torque transient detection model from training and test data is detailed. The assembly scenario used to acquire the data consisted of attaching a switch to a box, see Fig. 1. The objective for the robot was to move toward the box while holding the switch, thus pushing the switch against the box, until it snapped into place. The robot should

* The authors work at the Department of Automatic Control, Lund University, PO Box 118, SE-221 00 Lund, Sweden.
Martin.Karlsson@control.lth.se

The authors would like to thank Jacek Malec, Mathias Haage and Elin Anna Topp at Computer Science, Lund University, as well as Fredrik Bagge Carlson and Björn Olofsson at Dept. Automatic Control, Lund University, for valuable discussions throughout this work. The authors are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University. The research leading to these results has received funding from the European Commission's Framework Programme Horizon 2020 – under grant agreement No 644938 – SARAFun.



(a) (b) (c) (d)

Fig. 1: The ABB YuMi robot [8] was used in the experiments (a). The experimental setup for the switch assembly task is shown in (b) to (d). The robot gripper, box and switch are shown in (b). The robot grasped the switch, and attached it to the box by pushing downwards. The downward motion began in (c), where the switch was not yet snapped into place. It ended when the assembly was completed, in (d). These photos were taken during the experimental evaluation (see Sec. III), and the same setup was used for gathering training and test data (see Sec. II-B).

(a) (b) (c)

Fig. 2: Experimental setups for evaluation of the RNN model on objects that had not been used for acquiring test or training data. The pocket calculator and its battery cover are shown in (a), the spectacle case is shown in (b), and the power switch on the extension cord is shown in (c). For each setup, the robot was programmed to move the gripper downwards until a transient was detected, and subsequently move the gripper upwards.

detect the completion of the task automatically, stop moving toward the box, and possibly start a new movement.

A. Sequence model

An RNN [2], [3] was used as a sequence classifier. This choice is discussed in Sec. V. It had a sequence of joint torques as input, one single output indicating whether the sequence contained a given transient or not, one hidden layer, and recurrent connections between its hidden neurons. Each input torque sequence consisted of $T = n_{\text{pre}} + 1 + n_{\text{post}}$ time samples, where n_{pre} and n_{post} were determined as explained in Sec. II-C. In turn, each time sample consisted of $n_{\text{ch}} = 7$ channels; one per robot joint. The dimension of the hidden layer was chosen to be the same as the number of input channels, n_{ch}.

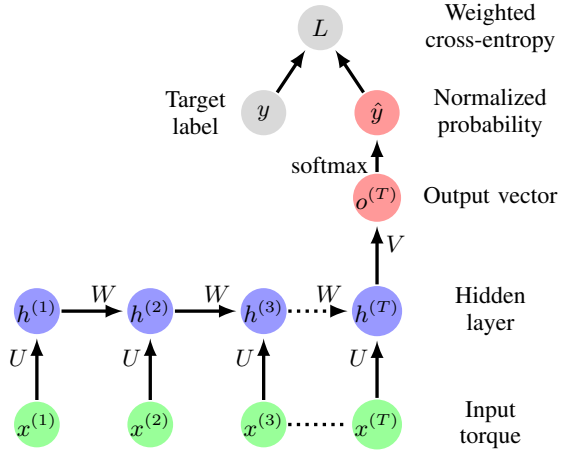


Fig. 3: The RNN visualized as an unfolded computational graph, where each node is associated with a certain time step. The biases b and c, as well as the activation function $\tanh(\cdot)$, are omitted for a clearer view, but the computations are detailed in (1) to (4). The input torque was used to determine the hidden state, which was updated each time step. The last hidden state was used to determine the normalized probability $\hat{y}$ of whether a transient was present in the time sequence or not.

Denote by $h^{(t)}$ the activation of the hidden units at time step t. The activation was defined recursively as

$$h^{(1)} = \tanh(b + Ux^{(1)}) \tag{1}$$

$$h^{(t)} = \tanh(b + Wh^{(t-1)} + Ux^{(t)}) \qquad t \in [2; T] \tag{2}$$

where U and W are weight matrices, both of size $n_{\text{ch}} \times n_{\text{ch}}$, b is a bias vector with dimension n_{ch}, and $x^{(t)}$ is the input at time t. Further, $\tanh(\cdot)$ represents the hyperbolic tangent function. After reading an entire input sequence, the RNN produced one output $o^{(T)}$ given by

$$o^{(T)} = c + Vh^{(T)} \tag{3}$$

where V is a weight matrix of size $2 \times n_{\text{ch}}$, and c is a bias vector with dimension 2. Finally, the softmax operation was applied to generate $\hat{y}$, a vector that represented the normalized probabilities of the output elements.

$$\hat{y} = \left[\frac{e^{o_1^{(T)}}}{e^{o_1^{(T)}} + e^{o_2^{(T)}}} \quad \frac{e^{o_2^{(T)}}}{e^{o_1^{(T)}} + e^{o_2^{(T)}}} \right]^T \tag{4}$$

Here, $o_i^{(T)}$ represents the i:th element of the output vector. If the first element of $\hat{y}$ was larger than the second, or equivalently, larger than 0.5, the data point was classified as positive, *i.e.*, it was indicated that the task was completed within the sequence. Vice versa, if the first element was less than than 0.5, the data point was classified as negative. The RNN architecture is visualized in Fig. 3.

B. Gathering of training data and test data

Training data and test data were obtained as follows. The right arm of the robot was used to grasp the switch, just above the box, as shown in Fig. 1. Thereafter, a reference velocity was sent to the internal controller of the robot, causing the robot gripper to move toward the box at 1.5 mm/s, thus pushing the switch against the box. Once the switch was snapped into place, the robot was stopped manually by the robot operator. The robot joint torques were recorded in 250 Hz. The torque transient, induced by the snap-fit, was labeled manually, and used to form a positive data point. This procedure was repeated $N = 50$ times, which yielded 50 positive data points. Data prior to each transient were used to form negative data points.

Given n_{pre} and n_{post}, a positive data point was formed by extracting a torque sequence, from n_{pre} samples previous to the peak value of the transient (inclusive), to n_{post} samples after (inclusive). Negative data points, with the same sequence length T as the positive ones, were extracted from torque measurements that ranged from a couple of seconds before the transient, until the positive data point (exclusive). The negative data points were chosen so that overlap was avoided. For each data point, the target was labeled as a two-dimensional one-hot vector y, where $y = [1 \;\; 0]^T$ represented a positive data point, and $y = [0 \;\; 1]^T$ represented a negative one.

Note that with the approach above, it was possible to extract several negative data points, but only one positive data point, for every assembly trial experienced by the robot.

Half of the positive and negative data points were used in the training set, and the other half was used in the test set.

C. Model training

Given the training set, the model parameters U, V, W, b, and c were determined by minimizing a loss function L. The training set contained much more negative data points than positive ones. If not taken into account, this type of class imbalance has been reported to obstruct the training procedure of several different classifiers. The phenomenon has been described in more detail in [9], [10], and should be taken into account when designing the loss function. Consider first the following loss function $\bar{L}$, which is the ordinary cross-entropy between training data and model predictions, averaged over the training examples.

$$\bar{L} = -\frac{1}{D}\sum_{d=1}^{D}\sum_{a=1}^{A} y_a^d \log \hat{y}_a^d \tag{5}$$

Here, a and d are indices for summing over the vector elements and training data points, respectively. This cross-entropy is commonly used as a loss function in machine learning [2], [11]. Due to the class imbalance in the present training set, it would be possible to yield a relatively low loss $\bar{L}$ by simply classifying all or most of the data points as negative, regardless of the input, even though that strategy would not be desirable.

In order to take the class imbalance into account, weighted cross-entropy was used as loss function. Denote by r the ratio between negative and positive data points in the training set, and introduce the weight vector $w_r = [r \;\; 1]^T$. The loss function was defined as

$$L = -\frac{1}{D}\sum_{d=1}^{D}\sum_{a=1}^{A} y_a^d \log \hat{y}_a^d \cdot w_r^T y^d \tag{6}$$

The RNN in Sec. II-A was implemented as a computational graph in the Julia programming language [12], using TensorFlow [13], [14] and the wrapper TensorFlow.jl [15].

The values of n_{pre} and n_{post} were determined using both the training set and the test set as follows. All positive data points available were used, and $r = 20$ times as many negative data points. Starting with $n_{\text{pre}} = n_{\text{post}} = 1$, the model was trained using the training set, and its performance was measured using the test set. Subsequently, both n_{pre} and n_{post} were increased by 1, and the training and evaluation procedure was repeated. This continued until perfect classification was achieved, or until the values of n_{pre} and n_{post} were large; (30 was chosen as an upper limit, though it was never reached in the experiments presented here). Thereafter, n_{pre} was kept constant, and it was investigated how much n_{post} could be lowered with retained performance. This was done by decreasing n_{post} one step at a time, while repeating the training and evaluation procedure for each value. Once the performance was decreased, the value just above that was chosen for n_{post}. This way, the lowest possible value of n_{post} was found, that resulted in retained performance.

Once n_{pre} and n_{post} were determined, new model parameters were obtained by training on a larger data set, with $r = 100$. The reason for using a lower value for the other iterations, was that it took significantly longer computation time to use such a large data set.

Due to the class imbalance in the test set, ordinary classification accuracy, as defined by the number of correctly classified test data points divided by the total number of test data points, would not be a good model performance measurement. Instead, the F-measurement [16] was used, defined as

$$F_1 = 2\frac{PR}{P+R} \tag{7}$$

where P is the precision, *i.e.*, the number of correctly classified positive data points divided by the number of all data points classified as positive by the model, and R is the recall, *i.e.*, the number of correctly classified positive data points divided by the number of all data points that were truly positive. The value of F_1 ranges from 0 to 1, where 1 indicates perfect classification.

III. Experiments

The ABB YuMi robot [8], shown in Fig. 1, was used for experimental evaluation. To facilitate understanding of the experimental setup and results, a video is available on [17]. First, the performance of the RNN obtained as described in Sec. II was evaluated on the switch assembly scenario. Since the test set was used to determine the hyper parameters

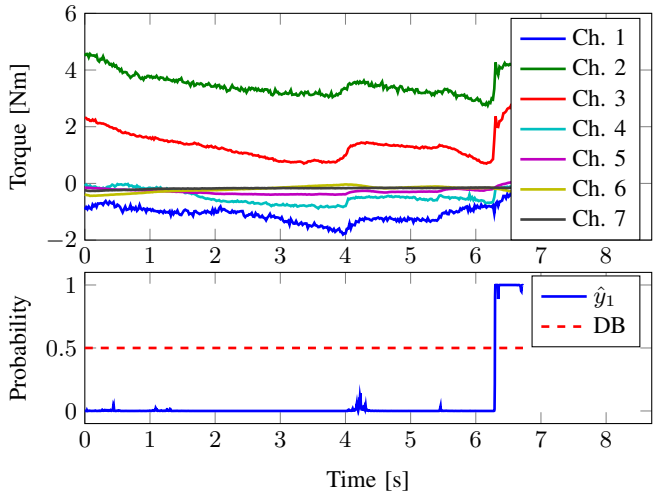


Fig. 4: Data from one of the experiments. The robot joint torques (upper plot) were used as input for the RNN. These are represented by one channel (Ch.) per joint. The first element of the RNN output ($\hat{y}_1$ in the lower plot) was close to 0 before the assembly was completed, and increased to close to 1 once the task was finished. Completion was indicated when $\hat{y}_1$ was above the decision boundary (DB, at 0.5) for the first time. Thus, for detection purposes, the RNN output generated after this event was not relevant.

of the RNN, *i.e.*, n_{pre} and n_{post}, it was necessary to gather new measurements to evaluate the general performance. The experimental setup was similar to that in Sec. II-B, except that the measured torque sequences were saved and classified by the RNN, instead of just saved to the training and test sets. The robot was programmed to first move its gripper down, thus pushing the switch against the box. Once a transient was recognized, it was programmed to stop its downward motion, and instead move the box to the side. The assembly was repeated 50 times, to evaluate the robustness of the proposed approach. The experimental setup is visualized in Fig. 1.

Thereafter, the same RNN model was tested on manipulation tasks that involved objects not used during model training, in order to test its generalizability. Again, the robot was programmed to move its gripper down until a transient was detected. Once a transient was detected, it was programmed to move its gripper upwards. Three tasks were considered: snapping a battery cover into place on a pocket calculator, closing a spectacle case, and pushing a power switch on an extension cord. The setup for each task is shown in Fig. 2. For each of these tasks, 50 attempts were made for validation.

IV. Results

The performance of the RNN on the test set, for different values of the hyper parameters, is shown in Table I. The abbreviations are as follows: number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The hyper parameters were increased until $n_{\text{pre}} = n_{\text{post}} = 5$, for which perfect classification was obtained. Then, n_{post} was decreased until the performance decreased at $n_{\text{post}} = 2$. With this value of n_{post}, larger values of n_{pre} were tested (see second last row in Table I), which did not yield perfect classification for any values, *i.e.*, $F_1 < 1$. Thus, $(n_{\text{pre}}, n_{\text{post}}) = (5, 3)$ was chosen for the final model.

TABLE I: RNN performance on the test set, for different values of the hyper parameters. The row with the lowest value of n_{post} that yielded perfect classification is marked in **blue**. With these values, the model was trained and tested again, but now with more negative data points (see last row, marked in **red**).

n_{pre}	n_{post}	TP	TN	FP	FN	P	R	F_1
1	1	20	500	0	5	1	0.80	0.89
2	2	22	500	0	3	1	0.88	0.94
3	3	23	498	2	2	0.92	0.92	0.92
4	4	23	500	0	2	1	0.92	0.96
5	5	25	500	0	0	1	1	1
5	4	25	500	0	0	1	1	1
5	**3**	**25**	**500**	**0**	**0**	**1**	**1**	**1**
5	2	22	500	0	3	1	0.88	0.94
[6; 30]	2	-	-	-	-	-	-	< 1
5	**3**	**25**	**2500**	**0**	**0**	**1**	**1**	**1**

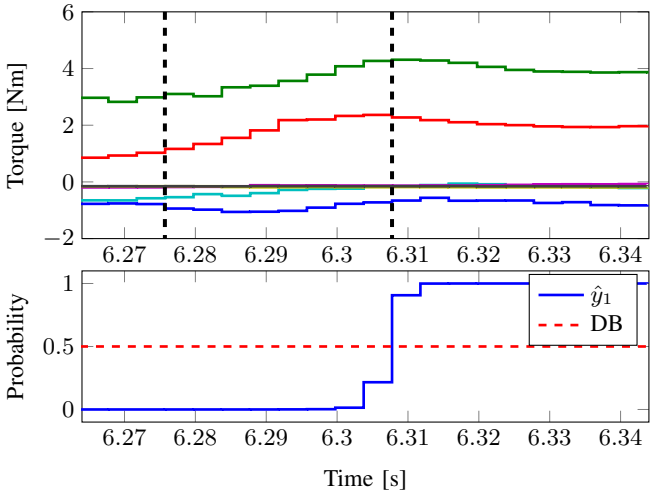


Fig. 5: Same data as in Fig. 4, but zoomed in on the time when the task finished. The legend of the upper plot is absent for better visualization, but can be found in Fig. 4. The transient was detected at time $t = 6.308\,\text{s}$. The first torque sequence to be classified as positive was that within the vertical dashed lines in the upper plot.

After training, the RNN detected all 50 snap-fits in the switch assembly experiments correctly, without any false positives prior to the snap. The torque data and RNN output from one of the trials are shown in Figs. 4 and 5. The other trials gave qualitatively similar results. The RNN also detected each transient successfully, without any false positives, for each trial of the tasks that involved objects not used for model training, shown in Fig. 2. Data from one trial of each task are shown in Fig. 6, and the other trials gave qualitatively similar results.

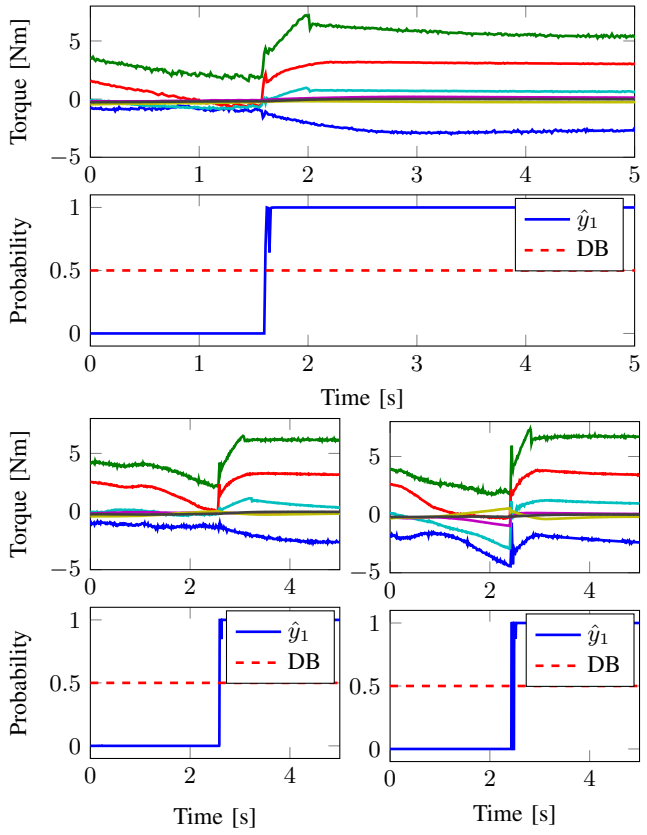


Fig. 6: Experimental data from the manipulation tasks visualized in Fig. 2: snapping the battery cover into place on the pocket calculator (upper), closing the spectacle case (lower left), and pushing the power switch on the extension cord (lower right). The organization of each plot is the same as in Figs. 4 and 5.

V. Discussion

There are several alternatives to RNNs for classification. Using an RNN is motivated as follows. Two less complicated models, matched filter and logistic regression, were tried initially, without achieving satisfactory performance on test data. RNNs are specialized in processing sequential data, and the torque measurements used in this work were sequential. Thanks to the parameter sharing of the RNN, it is possible to estimate a model with significantly fewer training examples, than would be needed without parameter sharing. Compared to models that are not specialized in sequential data, *e.g.*, logit models, SVMs, and ordinary neural networks, the RNN is less sensitive to variations of the exact time step in which some information in the input sequence appears. An RNN can also be generalized to classify data points of sequence lengths not present in the training set, though this was not used in this present work. General properties of RNNs are well described in [2].

Compared to [1], the approach proposed here had three new benefits. A force sensor was no longer required, the detection delay was reduced, and the computation time for model training was shortened. In [1], $n_{\text{post}} > 10$ (corresponding to > 40 ms) was required for perfect classification, whereas our proposed method required $n_{\text{post}} = 3$ (12 ms). The training time of the RNN was in the order of minutes on an ordinary PC, which was an improvement compared to days in [1]. Further, this research extended [1], [18] by verifying generalization of the classification model to manipulation of new objects, that had not been used to generate training and test data. It is expected that this generalization is limited to similar tasks. Therefore, the idea is that the training procedure described in Sec. II should be gone through whenever a detection model for a completely different task is required.

The concept of weighted loss to compensate for class imbalance in machine learning has been evaluated in [9], and successfully applied to a deep neural network in [19]. Equation (6) extends (5) by the factor $w_r^T y^d$, which evaluates to r for positive data points, and to 1 for negative ones.

In the experiments presented here, the decision boundary for $\hat{y}_1$ was set to 0.5, meaning that a task was classified as completed when the estimated probability of completion was above 50 %. However, the meaning of $\hat{y}_1$ is intuitive, and the decision boundary can be adjusted to a desired level of confidence.

Given a certain contact force/torque acting on the end-effector of the robot, the corresponding joint torques depend on the configuration, as well as gravity and friction. In order to generalize the detection to other robot states than the one used in the training, it would be a good idea to estimate the contact forces/torques, by first modeling the gravity and friction-induced torques, and subsequently compensating for these. In a complete setup, the validation procedure described here could be combined with more sophisticated motion controllers, such as dynamical movement primitives [20]–[22]. Whereas the evaluation of the model was done in real-time in this work, the RNN training was performed offline. It therefore remains as future work to create a user interface that allows for an operator to gather training data and test data, label them, and run the training procedure. It is also important to shorten the reaction time of the robot, *i.e.*, to further reduce the value of n_{post}. This could be done by including more sensors. For instance, the snap-fit assembly generates a sound, easily recognized by a human. Adding a microphone to the current setup, would therefore add information to the detection approach. In turn, this could be used to decrease the required amount of training data, improve robustness of the detection, or detect the transient earlier.

VI. Conclusion

In this work, we have addressed the question of whether robot joint torque measurements could be used for detection of force/torque transients in robotic manipulation. We have shown that such detection is possible, which is the main contribution of this paper. In contrast, the concept of RNNs is well known from previous research, and was used in this paper for the purpose of evaluation, as well as to exemplify how joint torques could be used for detection in a practical

setup. First, training and test data were gathered and labeled. Then, these were used to determine the parameters of the detection model. Finally, a real-time application for transient detection was implemented and tested, both on the assembly task used to generate training and test data, and on new tasks, that had not been used to determine the model parameters. The method presented seems promising, since the resulting model had high performance, both on the test data and during the experiments. A video that shows the functionality is available on [17].

REFERENCES

[1] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson, "Detection of contact force transients in robotic assembly," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 26–30, Seattle, WA, May 2015, pp. 962–968.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, Cambridge, MA, 2016, visited on 03/06/2017. [Online]. Available: `http://www.deeplearningbook.org`.

[3] A. Graves, "Neural networks," in *Supervised Sequence Labelling with Recurrent Neural Networks*, Springer, Berlin Heidelberg, Germany, 2012, pp. 15–35.

[4] A. Rodriguez, D. Bourne, M. Mason, G. F. Rossano, and J. Wang, "Failure detection in assembly: Force signature analysis," in *IEEE Conference on Automation Science and Engineering (CASE)*, August 21–24, Toronto, Canada, Aug. 2010, pp. 210–215.

[5] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 7–12, Vilamoura, Portugal, Oct. 2012, pp. 356–363.

[6] M. Linderoth, A. Stolt, A. Robertsson, and R. Johansson, "Robotic force estimation using motor torques and modeling of low velocity friction disturbances," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 3–7, Tokyo, Japan, Nov. 2013, pp. 3550–3556.

[7] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson, "Force controlled robotic assembly without a force sensor," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 14–18, St. Paul, MN, May 2012, pp. 1538–1543.

[8] ABB Robotics. (2017). ABB YuMi, [Online]. Available: `http://new.abb.com/products/robotics/yumi` (visited on 01/23/2016).

[9] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[10] N. Japkowicz, "The class imbalance problem: Significance and strategies," in *International Conference on Artificial Intelligence*, Las Vegas, Nevada, June 26–29 2000.

[11] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer Science & Business Media, New York, 2013.

[12] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *arXiv:1411.1607*, 2014.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[14] TensorFlow. (2018). An open-source software library for machine intelligence, [Online]. Available: `https://www.tensorflow.org/` (visited on 02/14/2018).

[15] J. Malmaud. (2017). A Julia wrapper for TensorFlow, [Online]. Available: `https://github.com/malmaud/TensorFlow.jl` (visited on 03/09/2017).

[16] G. Hripcsak and A. S. Rothschild, "Agreement, the F-measure, and reliability in information retrieval," *Journal of the American Medical Informatics Association*, vol. 12, no. 3, pp. 296–298, 2005.

[17] M. Karlsson. (2017). Detection of contact force transients from robot joint torques, Dept. Automatic Control, Lund University, [Online]. Available: `https://www.youtube.com/watch?v=TE1q5lQr4nk` (visited on 02/14/2018).

[18] M. Karlsson, "On motion control and machine learning for robotic assembly," TFRT–3274–SE, Dept. Automatic Control, Lund University, Lund, Sweden, 2017.

[19] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-task learning and weighted cross-entropy for DNN-based keyword spotting," *Interspeech 2016*, pp. 760–764, 2016.

[20] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[21] M. Karlsson, A. Robertsson, and R. Johansson, "Autonomous interpretation of demonstrations for modification of dynamical movement primitives," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 29–June 3, Singapore, 2017, pp. 316–321.

[22] M. Karlsson, F. Bagge Carlson, A. Robertsson, and R. Johansson, "Two-degree-of-freedom control for trajectory tracking and perturbation recovery during execution of dynamical movement primitives," in *20th IFAC World Congress*, July 9–14, Toulouse, France, 2017, pp. 1959–1966.