Manual for tactile object classification with the Jaco arm on the Doro robot sensorized with 3 OptoForce sensors

Philip Maus
Assistive Robotics Laboratory
BioRobotics Institute Pontedera

January 2021

Introduction

This document serves as a manual for the tactile object classification with the Jaco arm on the Doro robot sensorized with 3 OptoForce sensors. It gives an overview of the data acquisition in ROS, data augmentation scripts, the neural network training routine, and the object classification on real-life experimental data.

The presented concepts and routines were used in the development of the Master's Thesis "Study, Development, and Experimentation of Machine Learning Techniques in Grasping Tasks using Tactile Force Sensors in 3-Finger-Gripper" by Philip Maus in the Master's studies of Bionics Engineering of the University of Pisa, the article "Tactile-based Object Classification using Sensorized Gripper and Deep Learning approach", submitted to ICRA'21, and the paper "Data Augmentation of Tactile Time-Series Dataset for Object Classification" submitted to Autonomous Robots, both designed in the Assistive Robotics Laboratory of the BioRobotics Institute Pontedera, and with the main authors Philip Maus and Jaeseok Kim.

The recommended software setup is Python 3, ROS kinetic, Keras version 2.2.4, and TensorFlow version 1.14.

1 Data Acquisition

1.1 Hardware Setup

The hardware setup includes 3 OptoForce OMD-20-SE-40N sensors installed on the 3-finger-gripper of a Jaco 1 robotic arm, which is mounted to the Doro robot of the Robot-Era project [1]. The OptoForce OMD-20-SE-40N is a 3-dimensional optical force sensor with a resolution of 2.5 mN and a maximum sampling frequency of 1 kHz [2]. The Kinova Jaco arm is a 6 DOF carbon-fiber robotic manipulator with a payload of 1.3 kg and has a pre-mounted KG-3 gripper with 3 fingers [3]. The gripper is equipped with one actuator per finger, each with a

temperature sensor and rotational encoder [4]. Each finger is made up of a proximal and a distal phalanx. Figure 1 shows the hardware setup with the three force sensors mounted on the distal finger phalanxes via 3D-printed mountings.

Figure 1: Hardware setup with OptoForce 3D force sensors mounted on KG-3 gripper on Jaco1 arm

1.2 Software Setup

The spatial object exploration consists of a haptic glance, a single squeezing motion of the gripper. This grasp and the readback of the Jaco actuator position data and the OptoForce sensor force data is implemented via a ROS connection. The code for data acquisition is based on the ROS concept of actions. Actions allow sending asynchronous messages between two nodes, providing feedback and result.

All files are included in the **ROS** folder. The main functionality is guaranteed by the action client **record_data_action_client**, and the action server **record_data_action_server**, located in the **src** folder. The action client fulfills the function of calling the action server and therewith starting the data acquisition and defining the object label. The action server subscribes to the topics published by the OptoForce sensors and the Jaco arm. The force sensors publish the three-dimensional force values with a frequency of 100 Hz and filtered with a lowpass filter of 15 Hz. The Jaco arm publishes the finger position of the three-finger-gripper with a frequency of 100 Hz. The gripper actuation is based on an action server/action client base, in which the server is provided by the Kinova code and the data acquisition action server functions as finger actuation action client. Finally, the results, transferred to the data acquisition action client, are saved in a CSV file. The algorithm **merge_all_meas_data.py** in the **scripts** folder allows

to merge single data samples into one data set file, while adding the Euclidean force norm for every force sensor as an additional feature, resulting in the 15 features used for this entire work. The terminal commands required to record a measurement is:

- **roscore**
- **roslaunch kinova_bringup kinova_robot.launch kinova_robotType:=j2s6s300 use_urdf:=true** launch the Kinova arm with the required specifications and start the topic publishing process
- **roslaunch optoforce optoforce.launch** launch the OptoForce 3D force sensors and start the topic publishing process
- **roslaunch thesis start_record_data.launch** launch the action server record_data_action_server
- **roslaunch thesis call_record_ data.launch class_label:=0** launch the action client record_data_action_client, defining the object label under which the data sample will be saved. The objects with corresponding object labels can be found in the file **objects.pdf**.

The data acquisition process requires around 10 seconds, with the terminal displaying status updates in the process. The measurement results in a data sample saved as **label_timestamp.csv**. Each sample consists of 50 time steps and 12 features, the force in x, y, and z-direction for every force sensor, and the position of the three gripper fingers.
All the code is implemented in ROS kinetic.

1.3 Data Set Recording

Tactile grasping data are acquired in two different ways. The first is a training data set, **000_training_data.csv**, entirely recorded by the author of this work and consisting of 100 measurements of each object, concluding in 2000 samples. The second way is the collection of real-life experimental data, **000_real_life_data.csv**, which was recorded by 12 unbiased experimenters. It represents the real-life conditions of variations in placement and object-handling among humans and will, therefore, be used to determine the NN model classification success rates. The data consists of labeled measurements of the objects.

For the real-life experimental data, each experimenter was asked to pick 10 objects from the object set. Repetitions and an empty grip are allowed. For every measurement the experimenter was instructed to place the object in a way so it is in contact with the sensors during the entire grasping process, to transfer the full contact forces to the force sensors. If possible the object should be grasped in between all 3 fingers. The object is to be placed on the lower two force sensors with its full weight from the start. The ideal placement for non-symmetrical objects is not defined, the experimenter is asked to place the object in as many different orientations as possible following the given instructions. To create the training data set, the author followed the same instructions, varying position and orientation to guarantee a broad training space.

2 Data Augmentation

Data augmentation enables the creation of artificial data samples based on the recorded ones. This allows for expanding the data set to train NNs more efficiently. For each sample of the collected data set, new samples with the same label are created. The training data set is, with a size of 2000 samples on a 20-class-classification task, rather limited. Neural networks are known to require a high number of samples to be trained reliably due to the big extent of parameters.

Additionally, the more complex the NN model is, the more samples are needed. Therefore data augmentation seems essential to achieve an acceptable classification performance.

The 5 basic data augmentation methods used in this work are:

- **Jitter** A random Gaussian noise with zero mean and a standard deviation of 0.05 is added to the data set.
- **Scaling** The time series data for every feature is multiplied with a random factor with mean value of 1 and standard deviation of 0.1. This can be considered as applying constant noise to the samples.
- **Magnitude Warping** Each feature time data series is convolved with a smooth random curve created with the cubic spline method with 4 knots around the value 1 with a standard derivation of 0.05.
- **Time warping** The time intervals between the samples are smoothly distorted, thereby causing significant signal parts to slightly shift in their location.
- **Cropping** One subsample/time window of each sample is cropped out, the remaining data is set to zero.

The methods and code used to perform the calculations are strongly inspired by T. T. Um et al. [5]. Overall, this work uses 28 augmented datasets, consisting of single and combined augmentation methods, following this scheme:

- The data sets **augm_method_samples_4000_augm_data** include the 2000 original data samples plus 4000 data samples artificially generated with the corresponding augmentation method.
- The data sets **augm_method_samples_8000_augm_data** include the 2000 original data samples plus 8000 data samples artificially generated with the according augmentation method.
- The data sets **augm_method1_augm_method2_samples_4000_augm_data** include the 2000 original data samples plus 4000 artificially generated data samples of augmentation method 1 plus 4000 artificially generated data samples of augmentation method 2.
- The data sets **augm_method1_augm_method2_added_samples_8000_augm_data** include the 2000 original data samples plus 8000 artificially generated data samples of BOTH augmentation methods 1 and 2, so all the newly generated samples undergo all the the augmentation methods.
- The data sets **augm_method1_augm_method2_augm_method3_added_samples_8000_augm_data** include the 2000 original data samples plus 8000 artificially generated data samples of ALL THREE augmentation methods 1, 2, and 3.

All information concerning data augmentation can be found in the **Data augmentation** folder. The algorithm **data_augmentation.py** is used to create the augmented data sets:

- augm_method_samples_4000_augm_data
- augm_method_samples_8000_augm_data
- augm_method1_augm_method2_added_samples_8000_augm_data

- augm_method1_augm_method2_augm_method3_added_samples_8000_augm_data

The training data set is loaded and preprocessed, which includes object weight compensation and normalization of each feature over all samples to mean value 0 and standard deviation 1. In the main function of the code, the settings can be adjusted. The variable **augm_type** defines the type of augmentation, and **rep** the number of augmented samples created for each data sample in the original data set (therefore, rep=4 results in 8000 augmented data samples).

The algorithm **load_combine_datasets.py** allows merging existing data sets into one file, thereby creating the data sets of type augm_method1_augm_method2_samples_4000_augm_data.

All augmented data set files can be found in the archive file **augmented_datasets** under the following link: bit.ly/3iaOvDO. The settings to be determined in main are the methods of the files which are meant to be merged in one. Both algorithms are based Python 3.

3 Hyperparameter Search

To reach the best possible performance, the hyperparameters of a neural network have to be found before training the models. The training data set of 2000 samples are split into 80/10/10 training, validation, and test set for the hyperparameter optimization. Hyperopt [6] with the TPE algorithm and a validation accuracy loss function is run for 1000 trials for MLP and 500 for the other NNs. An early stopping parameter of 50 is deployed.

The folder **Hyperparameter search** contains a python file for the search for each NN. The input is the training data set, and the output a csv file in the folder **results**. Each result file contains the best value for each hyperparameter, the search duration, and the information for every trial conducted during the search.

4 Neural Network Training

The core task of this work is training the neural network models with the recorded data. The corresponding code can be found in the folder **Neural Networks** in the file **NN_train.py**. The data with which the NN is supposed to be trained is loaded and if needed preprocessed. The data set can be the original training data or an augmented data set. The file provides the code for the neural network models MLP, LSTM, CNN, CNNLSTM, ConvLSTM, and deep CNN (D-CNN). The NN training is conducted with a stratified 10-fold cross-validation, resulting in an 80/10/10 split for training/validation/test data. The hyperparameters found in section 3 are used for training. An early stoppage criterion based on the validation loss valley is deployed, with a patience parameter between 30 and 50 epochs. As result, the model file and test accuracy of every epoch are saved as well as the history file and the accuracy and loss over epoch plot for the model of the fold with the highest test accuracy. Additionally, the confusion matrix averaged over all folds is saved. All those result files can be found via bit.ly/3iaOvDO in the archive file **NN_training_results**. In the main function, the training data file, the NN type, and the data augmentation type can be defined.
The NN model training requires Keras version 2.2.4 and TensorFlow version 1.14.

5 Real-life Data Set Prediction

The success rate of a NN model is defined as the performance of the model applied to the 120 real-life experimental data samples recorded by 12 experimenters. Compared to the accuracy,

determined by the NN models on the test fold of the training data set, the success rate represents a real-life variety in object placement and orientation.

The folder **Predict real-life** contains all the necessary information. The script **predict_real_life_dataset.py** predicts the real-life success rate. The script first plots all the test accuracies of the different NN models trained on all available data sets averaged over all 10 models, with standard deviation. After this, the 120 real-life data samples are loaded and object weight compensation and data normalization is executed. An overview of the number of occurrences of each object in the real-life data is given. Finally for each trained NN-model the real-life success rate, average and standard deviation over all 10-fold cross-validation models, is computed, and the confusion matrix is saved in the folder **real-life success rates**.
The hyperparameter search requires Keras version 2.2.4 and TensorFlow version 1.14.

References

[1] F. Cavallo et al., "Development of a Socially Believable Multi-Robot Solution from Town to Home", *Cognitive Computation*, Vol. 6, p.954-967, 2014.
[2] OnRobot A/S, "OMD-20-SE-40N Datasheet", 2018.
[3] Kinova Inc., "Kinova Jaco^{TM}Prosthetic robotic arm User Guide", 2018.
[4] Kinova inc., "Kinova^{TM}Gripper KG series User Guide", 2018.
[5] T. T. Um et al., "Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks", *arXiv:1706.00527v2*, 2017.
[6] J. Bergstra, D. Yamins, D. D. Cox, "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms", *Proceedings SciPy*, 2015.