Robot programming by manual guidance

Miha Deniša

December 2020

Abstract

The module for robot programming by manual guidance enables fast and intuitive programming of new robot tasks without using traditional robot programming systems. With manual guidance, called kinesthetic teaching, the operator physically guides the robot through the desired robot configurations and movements in order to program a sequence of robot movements that lead to a successful task execution. Manual guidance can be used for the programming of point-to-point movements or whole movements (trajectories). The first section of this manual presents the general overview of the approach and the required prerequisites. Hardware and software components of the module are presented in the second section, while instructions on using the graphical user interface are given in the last section.

Contents

1 Overview **2**
1.1 Programming of robot tasks 2
1.2 Kinesthetic teaching 3
1.3 Robots supporting kinesthetic teaching 3

2 Module components **4**
2.1 Robot Operating System (ROS) 4
2.1.1 ROS installation 5
2.2 Button interface 6
2.3 Graphical user interface 7
2.3.1 Installation procedure 7
2.4 Database 7

3 Data acquisition by kinesthetic teaching **8**
3.1 GUI configuration 9
3.2 Using the GUI 10
3.3 Point-to-point movements 11
3.4 Adaptation, execution, and Cartesian DMPs 11

1 Overview

To create a robotic workcell that is applicable to small batch size production and frequent changes in product demands, the cell must enable fast and easy programming of new robot tasks. With standard robot programming approaches, the programming is usually performed by experts at production sites. To support the programming process, many robots are equipped with teach pendants, which enable the programmer to control the robot through a relatively complex interface. Programming with a teach pendant is still relatively cumbersome and time consuming. An alternative is provided by a programming by demonstration (PbD) approach, where new robot operations are programmed by analyzing human demonstrations of the desired task [1]. Programming by demonstration enable non-expert users to generate new robot operations in a relatively short time. Kinesthetic teaching introduced in Section 1.2 is a form of programming by demonstration.

1.1 Programming of robot tasks

To program a new robot task, we typically need to program a sequence of robot movements and the associated operations that lead to a successful execution of the task. The associated operations typically involve control of robot end-effectors, e.g. grasp/release an object with a gripper, but can also relate to other peripheral equipment available in the workcell and not mounted on the robot's end-effector. As kinesthetic teaching is useful for the programming of robot movements, we focus in this manual on the programming of movements.

Some definitions are needed before we can explain the functionality of the module:

Robot configuration: a set of values specifying all robot joint positions.

Robot pose: position and orientation of the tip of the robot.

State of the robot, point: in this text, state of the robot / point refers to either robot configuration or pose.

Joint space trajectory: a dense or even continuous sequence of robot configurations and times at which the robot should reach the specified configurations.

Cartesian space trajectory: a dense or continuous sequence of robot poses and times at which the robot should reach the specified poses.

Point-to-point movement (motion): only the initial and the final robot configuration or pose and the duration of movement are specified. The in-between points are not specified. To generate the robot movement between the two specified points, the in-between robot configurations or poses are generated by computing a straight line or a higher order polynomial between the initial configuration and the final configuration, parametrized by time.

Whole movement (trajectory): the motion of the robot from the initial to the final configuration / pose is fully specified and does not necessarily follow a linear or higher-order polynomial.

Kinesthetic teaching / guidance is used to gather single points (robot configurations and poses) or dense sequences of points from which continuous trajectories can be generated.

1.2 Kinesthetic teaching

In kinesthetic teaching [2], the robot arm is held by a human demonstrator and guided in order to perform the desired task. The states of the robot during the task execution are recorded by using the robot's sensors. Kinesthetic teaching requires only limited user training and can be performed by users without in-depth knowledge about robotics. However, this does not mean that kinesthetic teaching can be performed effortlessly. The quality of the recorded data and consequently the quality of robot programs namely rely on the quality of the performed demonstrations.

1.3 Robots supporting kinesthetic teaching

The main prerequisite for kinesthetic teaching of robot configurations and trajectories is a robot arm that can be freely guided around by a demonstrator. This can be achieved by putting the robot in a gravity compensation mode, where the weight of the robot is compensated by its motors and the robot arm can be guided with a minimal effort. By applying an external force to the robot by his/her own hands, the user can guide the robot kinesthetically without any additional sensing equipment. Gravity compensation requires the availability of torque values at each joint. They can be gathered in two different ways:

1. by measuring torque values at each joint using a torque sensor,
2. by measuring the electric current applied to the motor mounted at a joint and transforming the measured currents to torque values using a dynamic model.

The first approach results in more accurate torque values as we can avoid the transformation of currents to torque values with a dynamic model. Consequently, the gravity compensation works better and the robot arm can be guided through the desired motion with less effort. The quality of the gathered data is therefore better. However, the price of robot arms that include joint torque sensors is generally significantly higher than the price of robot arms without such sensors. Robot arm equipped with joint torque sensors include Kuka LBR iiwa and Franka Emika Panda.

To reduce the price of the robot, the second approach is often used. For example, Universal Robot UR series arms do not have joint torque sensors but support gravity compensation out of the box by transforming electric currents to torque values. This module is presented using this setup Universal Robot UR10 robot arm. An example demonstration using gravity compensation mode on a UR10 robot arm is shown in Fig. 1.

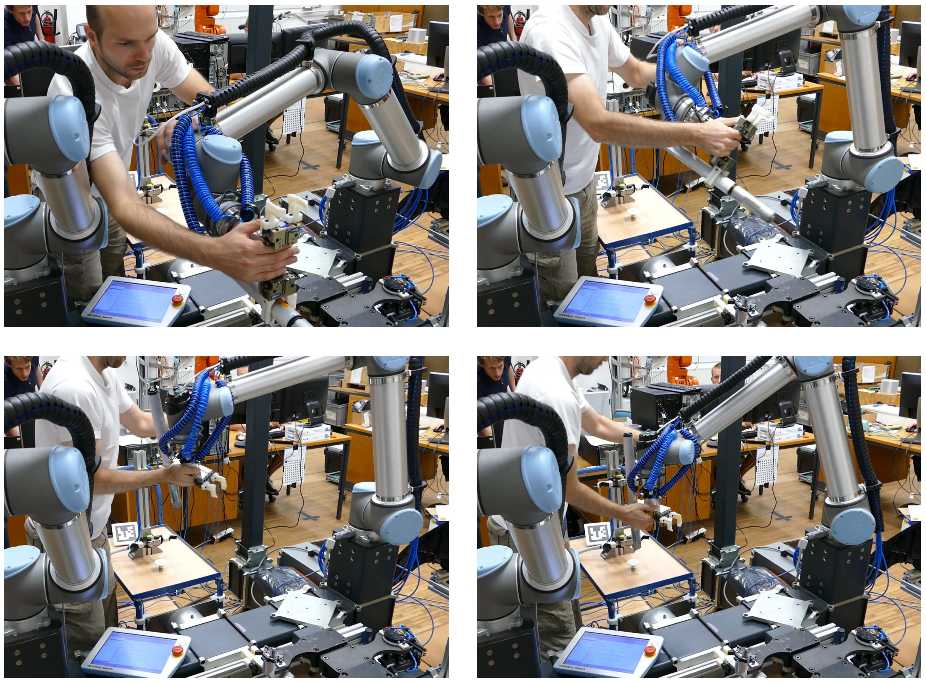


Figure 1: Kinesthetic guidance UR10 robot arm (from top-left to bottom-right).

2 Module components

This section presents all the components of this module, which consists of software and hardware components. Installation procedure for the software components is given. The module has been implemented for Universal robot UR10, but its adaptation to other robots that support gravity compensation is possible. Support for adaptation of the module to other robots can be provided by Jožef Stefan Institute on a contract basis[1].

A general overview of all main components of this module is shown in Fig. 2.

2.1 Robot Operating System (ROS)

The module requires the installation of ROS (Robot Operating System) [3, 4], which provides the database MongoDB, high level communication, and other

[1]Contact Timotej Gašpar: timotej.gaspar@ijs.si

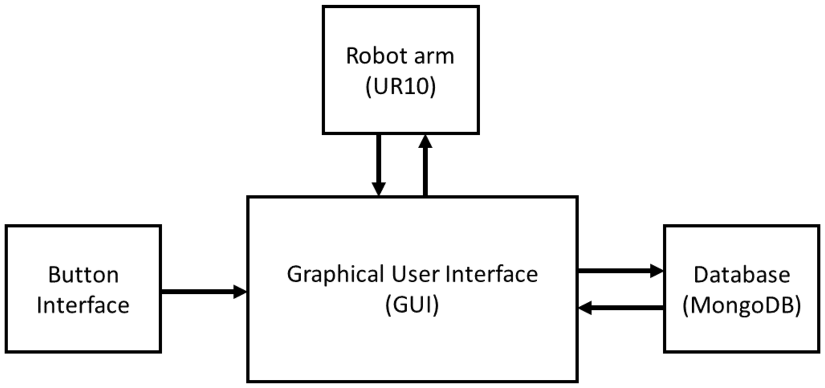


Figure 2: Overview of main module components. The GUI is at the core of the module. It receives triggers from the button interface, which are used to change the control mode of the robot, e.g. put the robot into gravity compensation mode, and initiate saving and reading the data from the database (MongoDB-based). The button interface is connected to the GUI via the robot's I/Os.

functionalities. While many features and tools are included in ROS, a subset of them is used to realize kinesthetic teaching within the cell: ROS nodes, MongoDB database, topics, services, action and parameters servers, messages, etc.

2.1.1 ROS installation

The ROS periphery interface is based on the Kinetic version of ROS. It can be installed on a machine running a Linux Ubuntu 16.04 LTS operating system. The minimal requirements for Ubuntu 16.04 are as follows:

- 2 GHz dual core processor or better
- 2 GB system memory
- 25 GB of free hard drive space
- Either a DVD drive or a USB port for the installer media
- Internet access is helpful

The tutorial covering the installation of Ubuntu 16.04 can be found at this link.

Once Ubuntu 16.04 is installed and running, ROS Kinetic can be installed by following instructions found at this link.

2.2 Button interface

To ease the acquisition of data for robot programming by kinesthetic teaching, a button interface can be mounted on the robot arm. It lowers the effort and shortens the time needed for robot programming by kinesthetic teaching. The buttons are used to trigger various functions that support kinesthetic teaching.

A 3D printed cover has been developed for this purpose. The cover houses 2 buttons and 2 switches. The cover suitable for Universal robot UR10 can be seen in Fig. 3. Each of the switches have a LED showing their status. The interface was developed to provide buttons that can be programmed for different actions. The default settings for this module maps the switch 1 to the gravity compensation control, switch 2 to the tool exchange system, while both buttons are used to trigger save events.

The tool exchange system is an optional hardware module which enables automatical exchange of the robot's end-effectors. It is composed of two parts, one permanently attached to the tip of the robot, the other to different end-effectors. Pneumatic, controlled via the robot's I/Os and the button interface, is used to attach and release the two parts of the exchange system.

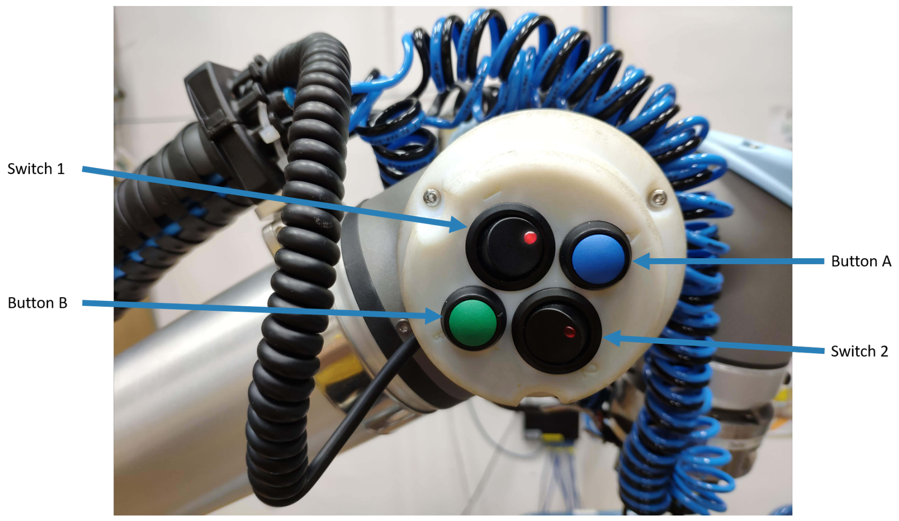


Figure 3: The button interface used for kinesthetic teaching. The button interface replaces one of the standard UR-10 joint covers. The button states are connected through the robot controller's I/Os.

As the buttons are connected through the robots's I/Os, their states can be read by observing the states on the robot. This is done through the ROS framework. Further specifics on the button interface are given in Section 3.

2.3 Graphical user interface

As presented in Fig. 2, the graphical user interface (GUI) is at the center of the kinesthetic teaching module. Through this interface, all the software components are called and triggered. The description of the GUI and the installation procedure are provided in this section. Section 3 describes how to use the GUI to gather data for robot programming by kinesthetic teaching.

2.3.1 Installation procedure

In order to use the GUI, it needs to be installed with required dependencies. To do that, a package called Helping Hand needs to be installed. This package contains various tools to facilitate kinesthetic teaching of robots. However, the main tool is the provided GUI. While installation instructions can be found on the repository, an overview is given here.

Before installing the package itself, ROS system needs to be installed. If not done before, follow the instructions given here. In addition, a working *catkin* workspace needs to be set up. If not done already, instructions can be found here.

To install the Halping Hand package perform the following steps:

1. Move to your catkin src directory and clone the repository:

   ```
   $ git clone https://github.com/tgaspar/helping_hand.git
   ```

2. Next, all of the other dependencies need to be set. For that wstool is used:

   ```
   $ wstool init
   $ wstool merge helping_hand/dependencies.rosinstall
   $ wstool up
   ```

   All of the cloned dependencies will now be in the src directory.

3. To query and install all libraries and packages perform the next command:

   ```
   $ rosdep install --from-paths . --ignore-src
   --rosdistro kinetic
   ```

4. Finally use the catkin command to build the package:

   ```
   $ cd ..
   $ catkin build
   ```

2.4 Database

The third main component of the kinesthetic teaching module is the database, which is the back-end of the module. The data acquired by the kinesthetic teaching module should be accessible throughout the entire workcell software framework, which is based on ROS. For this reason the `mongodb_store` ROS package is integrated, as it allows all ROS nodes in the network to access the database. The MongoDB database [5] usually runs on the same computer that runs the ROS core. After the data for robot programming have been acquired by kinesthetic teaching module and stored in the database, the required motions to execute the task can be computed. The program to control the robot to execute the desired task can then be compiled.

In our system, captured trajectories are represented as dynamic movement primitives (DMPs) [6]. They are used to encode the trajectories in a compact way and enable their modulation if needed. They are based on a set of nonlinear differential equations with a well-defined attractor system. Instead of storing the full set of points and their time components, a low number N of weights, which define the shape of the trajectory are stored. The number of weights define the fidelity of the reproduction and can be changed prior to recording. In addition, the final point and the duration of the trajectory is also stored. Further details are omitted at this point, and the reader is referred to [7, 6]. They are implemented both for Cartesian space control and joint space control. Single points can also be saved either as Cartesian poses or robot joint configurations and are used to generate point-to-point movements.

The following message types are used to store the gathered data:

- `sensor_msgs/JointState`; this message type is used to store joint space positions
- `geometry_msgs/Pose`; this message type is used to store Cartesian space poses
- `robot_module_msgs/JointSpaceDMP`; this message type is used to store a trajectory encoded in joint space DMP
- `robot_module_msgs/CartesianSpaceDMP`; this message type is used to store a trajectory encoded in Cartesian space DMP

3 Data acquisition by kinesthetic teaching

The main component of the kinesthetic skill acquisition module is the GUI. It is used by the user for single point or whole movement acquisition. If a single point is stored in the database it can later be used for generating point-to-point movements. For the acquisition of whole robot movements, a dense or even continuous sequence of robot points and corresponding times is stored. It can be used to record Cartesian space trajectory or a joint space trajectory. The trajectory is post processed to an extent, as is being encoded as a Dynamic Movement Primitive (DMPs). This sections presents a guide for configuring and using the GUI. In addition, it provides some further information regarding usage of single points and whole movements.

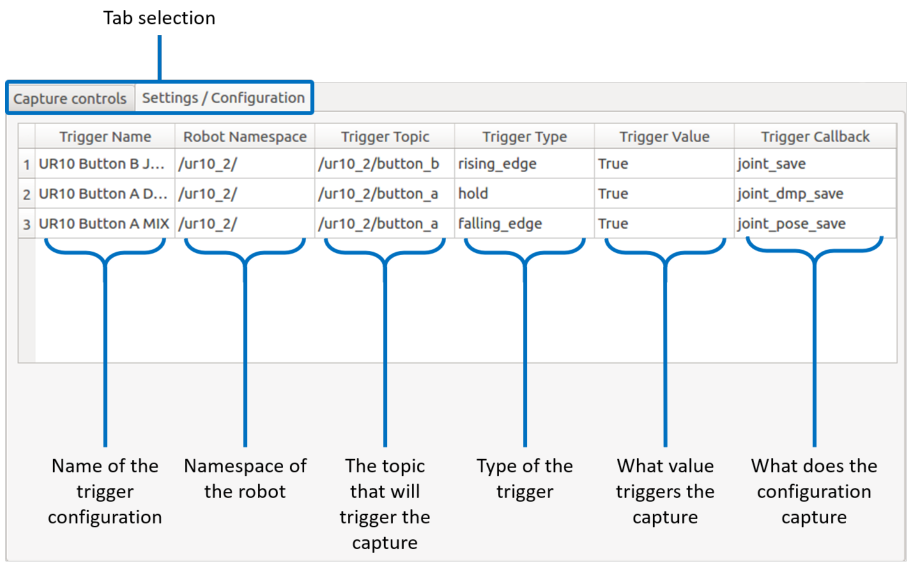
3.1 GUI configuration

This module uses the button interface to trigger save events by default. Nonetheless, to allow the user to have custom signals that trigger custom save events, configuration file is used. This YAML file tells the GUI which topic (of type `std_msgs/Bool`) triggers which save event. The default configuration file can be found in the `helping_hand_gui/conf` folder and on the Helping Hand repository here.

For the default UR10 module option, no changes to the configuration file should be necessary. If changes are needed, e.g. different topics are used to listen to buttons on the robots, the file needs to be modified. The following example configuration file defines the values that can be changed:

```
example_config:
trig_name:  "<Name of the configuration>"
robot_ns:  "<Robot's namespace>"
trig_topic:  "<Topic of type std_msgs/Bool>"
trig_type:  "<rising_edge, falling_edge or hold>"
trig_value:  true/false
trig_callback:  "<joint_save, joint_pose_save, joint_dmp_save
or cart_dmp_save>"
joint_topic:  "<Needed in every case where joints are saved>"
```

Further instructions on how to modify the configuration file are located here.


Tab selection
Capture controls
Settings / Configuration
Name of the trigger configuration
Namespace of the robot
The topic that will trigger the capture
Type of the trigger
What value triggers the capture
What does the configuration capture

	Trigger Name	Robot Namespace	Trigger Topic	Trigger Type	Trigger Value	Trigger Callback
1	UR10 Button B J...	/ur10_2/	/ur10_2/button_b	rising_edge	True	joint_save
2	UR10 Button A D...	/ur10_2/	/ur10_2/button_a	hold	True	joint_dmp_save
3	UR10 Button A MIX	/ur10_2/	/ur10_2/button_a	falling_edge	True	joint_pose_save

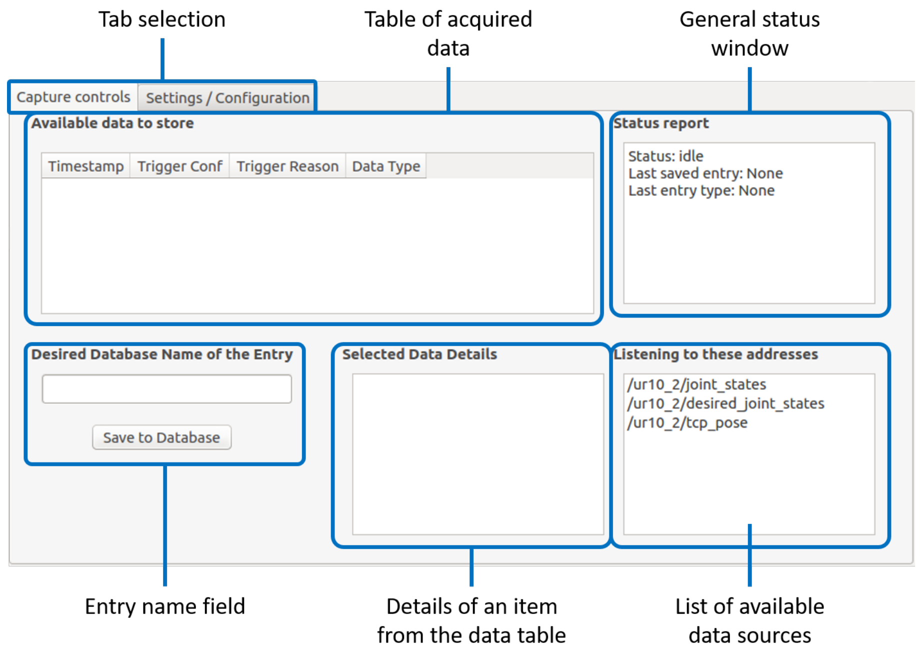
Figure 4: Kinesthetic teaching GUI configuration tab. Default settings are displayed in this example.

3.2 Using the GUI

To launch the GUI we suggest you use the provided launch file:

```
$ roslaunch helping_hand_gui gui.launch
```

The GUI has two tabs: the main tab called *Capture controls* and the secondary one called *Settings/Configuration*. The secondary one can be seen in Fig. 4 and is used to check the configuration of triggers defined in the YAML configuration file.



Status: idle
Last saved entry: None
Last entry type: None

/ur10_2/joint_states
/ur10_2/desired_joint_states
/ur10_2/tcp_pose

Figure 5: Kinesthetic teaching GUI main tab.

The primary tab, seen in Fig. 5, is used to capture points and/or movements during kinesthetic teaching. In order to be able to move the robot by hand, gravity compensation should be turned on (switch 1 by default). The user can then move the robot to a desired configuration or over a desired trajectory. Buttons on the robot, related to triggers, are used by default to store the desired pose/movement. Once triggers, as defined in the configuration file, will be caught, the corresponding data will be stored in the buffer. If, for example, the default configuration file is active, the button A is used for joint trajectory and joint pose recording. As can be seen in Fig. 4 line 2, showing the default settings, the GUI will continuously capture robot configurations while button A is active, and stores the corresponding trajectory in the buffer once it is released. As seen in line 3, just the current configuration will be stored in the buffer using the trigger type `falling_edge`, i.e. once the button is released.

All the data currently in the buffer can be seen in the *Table of acquired data* section of the GUI. If the buffered data is selected, details about it will be shown on the GUI. In order to store the selected buffered data, the *Save to Database* button is used. The desired name can be typed in the entry name field. If no name is typed the automatically generated name, under which the buffered data was selected, will be used. Once saved, the selected data will be removed from the buffer and *Table of acquired data.* Additional information is provided to the user via the *General status window* and through the list of available data sources.

3.3 Point-to-point movements

In order to expand the stored points into useful skills, point to point movements can be generated from them. As this is beyond the scope of this module details are omitted and two possible trajectory type generations are just briefly described.

Two points, saved as robot configurations in joint space, can be used to generate a movement in joint space following a trapezoidal velocity profile. This approach can be used to generate movements at well-defined velocities. The movements are guaranteed to be executable on the robot, if the speed limit is set accordingly, while being as fast as possible. For these type of movements, inputs need to be defined for each joint θ: initial joint position θ_i, final joint position θ_f, desired joint velocity $\dot{\theta}_c$, and desired joint acceleration $\ddot{\theta}_c$. While the first two are read from the database, the last two should be defined by the user. They must be specified within the allowed minimum and maximum joint velocity and acceleration.

Points, saved as robot poses in Cartesian space, can be used to generate straight line & SLERP trajectories following the minimum jerk velocity profile. This approach is useful to generate precise paths from the initial to the final point in Cartesian space. It will generate a straight line trajectory in Cartesian space with zero initial and final velocities and accelerations in a manner that minimizes the jerk throughout the motion. The following parameters are needed to generate these trajectories: initial and final position $\mathbf{p}_i,\ \mathbf{p}_f \in \mathbb{R}^3$, initial and final orientation $\mathbf{q}_i,\ \mathbf{q}_f \in S$, and the desired travel time T. While most of these parameters can be read from the database, the desired travel time needs to be defined by the user.

3.4 Adaptation, execution, and Cartesian DMPs

As the trajectories are encoded as DMPs, some post processing is possible. The trajectory can be executed at different speeds, and start and end positions can be altered. As DMPs provide a parameterized smooth representation of a the trajectory, it can be further smoothed by reducing the number of basis function.

While the standard implementation of DMPs are used for trajectories in joint space, the Cartesian Space Dynamic Movement Primitives (CDMP) are intended to handle Cartesian space robot motions where the orientations are represented with unit quaternions. CDMPs should be used when trajectories in Cartesian space are needed, e.g. to generate motions relative to the assembled workpiece. This way, the same recording, encoded as a CDMP, can be used to support assembly at different object locations. This module uses CDMPs as introduced in [6]. This implementation avoids direct integration of the orientation parameters, which require ad-hoc normalisation of the resulting orientation quaternion and thus generating a consistent trajectory for orientation in SO(3) manifold. To execute recorded trajectories, additional tools are needed to decode them from DMPs back into a time sampled trajectory.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, pages 1371–1394. Springer, Secaucus, NJ, USA, 2008.

[2] Micha Hersch, Florent Guenter, Sylvain Calinon, and Aude Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467, 2008.

[3] ROS (Robot Operating System). https://wiki.ros.org. Accessed: 2019-12-29.

[4] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[5] mongodb_store Package Summary. https://wiki.ros.org/mongodb_store. Accessed: 2019-12-29.

[6] Aleš Ude, Bojan Nemec, Tadej Petrič, and Jun Morimoto. Orientation in Cartesian space dynamic movement primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004, Hong Kong, 2014.

[7] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural Computation*, 25(2):328–373, 2013.