Gesture learning and execution in a humanoid robot via dynamic movement primitives

Sammy Pfeiffer^a, Cecilio Angulo^{b,*}

^a PAL Robotics. Pujades 77-79, 08005 Barcelona, Spain ^bGREC Research Group, Universitat Politècnica de Catalunya. Pau Gargallo 5, 08028 Barcelona, Spain.

Abstract

A system for learning and executing gestures in a humanoid robot has been developed and implemented in this work. Gestures are represented via the use of dynamical movement primitives on the robotic platform REEM. Since agnostic knowledge is considered when designing trajectories, our approach can be easily extended to other robots. Implemented work involves recording of gestures using three different procedures, from the own robot, with the help of an user, and from external devices. Next, the dynamic movement primitives representing the motions are generated to describe trajectories that will finally be executed on the real humanoid robot. Several experiments are provided illustrating how knowledge is acquired by the robot, represented in the form of dynamical systems, generalized and reproduced from different starting conditions.

Keywords: Dynamic Movement Primitives, humanoid robot, knowledge acquisition, gesture representation

1. Introduction

Gestures design is a key point in human robot communication, as far as motion gestures are also important nonverbal signals in human communication (Hostetter et al., 2007; Radford, 2009). In fact, robot gestures can make difficult tasks easier, by reducing the perceived workload and increasing the task performance (Salem et al., 2011; Lohse et al., 2014). However, knowledge acquisition, like new designed gestures for robots with a high number of degrees of freedom, is currently a very tiring and challenging task in mobile robotics (Breazeal et al., 2005; Calinon and Billard, 2007). Besides, new represented movements are static in the sense that they are fixed to how they were initially designed, i.e. they will always start and end with the same pose of the robot, following the same trajectory (Tenorth and Beetz, 2009). This limited mechanical behaviour is a barrier when robots try to physically express social meaning with gestures, or even discover new knowledge fron them (Trafton et al., 2013; Moshkina et al., 2014; Puigbo et al., 2015).

By the use of learning by demonstration techniques (Argall et al., 2009) with dynamic movement primitives (Schaal, 2006; Ijspeert et al., 2013), motions are represented like dynamical systems and they can be generalized by modifying initial values and tuning parameters according to the environmental changes (Rückert and d'Avella, 2013). Hence, for instance, they can be executed from different initial and final poses with similar, but not the same, trajectory which eases the task of making robots more social. Therefore, this work is approaching cognition from a dynamical systems perspective. Unlike the computational approaches to cognition, sharing the idea that cognitive systems are a special kind of computer (Puigbo et al., 2015), the dynamical approach exploits the idea that cognitive systems are a special kind of dynamical system. Both approaches can be also merged in a hierarchical structure, with the computational approach linking

^{*}Corresponding author

Email address: cecilio.angulo@upc.edu (Cecilio Angulo)

states and the dynamical approach exeuting them on dynamic movement primitives.

Robots with a high number of degrees of freedom refers to the ones like REEM, a humanoid service robot, or REEM-C, a biped robot, by PAL Robotics, a technological company working in the research and development of robots. They have developed up to now three biped humanoid robots, REEM-A (2004), REEM-B (2006) and REEM-C (2013), and three other prototypes with a mobile base, REEM-H01 (2010), REEM-H02 (2012) and the current version of REEM, the one used in this work (see Figure 1). In this work, its latest REEM prototype (released in 2012), will be employed. It is a full-size humanoid service robot that can act as a receptionist, entertain and compliment guests, provide dynamic information and make presentations and speeches in care centers, hotels, museums, shopping malls or airports. A list of their characteristics includes: weight of 100 Kg height 1.70 m, battery autonomy for 8 hours, 34 degrees of freedom and 26 sensors.



Figure 1: PAL Robotics' robots evolution and the REEM robot in this study.

The rest of the paper is organized as follows: First, we introduce the dynamic movement primitives in short. Then, the software resources implied in the development are listed, as well as the total procedure implementation is explained in the next section. Experimental results with three different forms of knowledge acquisition are described in the next section. Finally, some conclusions and further research lines are offered.

2. Dynamic movement primitives

Dynamic movement primitives (DMPs) is a method for trajectory control / planning derived from Stefan Schaal's lab. They were presented way back in 2006 (Schaal, 2006), and then updated in 2013 by Auke Ijspeert (Ijspeert et al., 2013). A detailed and very illustrative explanation about dynamic movement primitives can be found in (De-Wolf, 2014).

The starting trajectory defining DMPs is a point attractor dynamics equation:

$$\ddot{\mathbf{y}} = \alpha_{\mathbf{y}}(\beta_{\mathbf{y}}(g - \mathbf{y}) - \dot{\mathbf{y}}) \tag{1}$$

where y is our system state, g is the goal, and α and β are gain terms. This kind of system can be seen as a Proportional + Derivative control signal governing our system to the target. Now a forcing term is added, allowing us modify this trajectory:

$$\ddot{y} = \alpha_{y}(\beta_{y}(g-y) - \dot{y}) + f \tag{2}$$

How to define a nonlinear function f such that you get the desired behaviour is a non-trivial question.

In order to generalize and make the trajectory timeindependent, a new system, called canonical dynamical system, is introduced, having a very simple dynamics:

$$\dot{x} = -\alpha_x x \tag{3}$$

The canonical system starts at some arbitrary value, for instance $x_0 = 1$, and goes to 0 as time goes to infinity.

Hence, the forcing function f can be defined over a basis function ψ_i , defined on the canonical system x:

$$f(x,g) = \frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} x(g - y_0)$$
(4)

where y_0 is the initial position of the system and w_i is a weighting for the given basis function ψ_i . Usually, basis are defined like Gaussian functions centered at c_i , where h_i is the variance,

$$\psi_i = \exp\left(-h_i \left(x - c_i\right)^2\right) \tag{5}$$

so our forcing function is a set of Gaussian funtions that are 'activated' as the canonical system x converges to its target. Their weighted summation is normalized, and then multiplied by the $x(g - y_0)$ term, which is both a 'diminishing' and spatial scaling term.

Incorporating the *x* term into the forcing function guarantees that the contribution of the forcing term goes to zero over time, as the canonical system does. Moreover,

the spatial scaling means that once we have set up the system to follow a desired trajectory to a specific goal we would like to be able to move that goal farther away or closer in and get a scaled version of our trajectory.

Finally, generalization in terms of temporal scaling is also sought: to be able to follow the same trajectory at different speeds. Another term is added to our system dynamics, τ , like a temporal scaling term, giving temporal flexibility,

$$\ddot{y} = \tau^2 (\alpha_y (\beta_y (g - y) - \dot{y}) + f)$$

$$\dot{x} = \tau (-\alpha_x x)$$
(6)

To slow down the system you set τ between 0 and 1, and to speed it up you set τ greater than 1.

Now, we have a forcing term f making the system take a path as it converges to a target point, and temporal and spatial scalability. In order to set up the system to follow a specified path, we affect the system acceleration using the forcing term. So, from the desired trajectory (in fact the vector of desired points in the trajectory) $\mathbf{y}_d = \{\mathbf{y}_d(t_0), \dots, \mathbf{y}_d(t_n)\}$, the forcing term needed to generate this trajectory can be calculated as,

$$\mathbf{f}_d = \ddot{\mathbf{y}}_d - \alpha_v (\beta_v (g - \mathbf{y}) - \dot{\mathbf{y}})$$
(7)

Since the forcing term is comprised of a weighted summation of basis functions which are activated through time (4), an optimization technique like locally weighted regression (LWR) can be selected to calculate the weights w_i such that the forcing function matches the desired trajectory \mathbf{f}_d . The objective function to be minimized is

$$\sum_{t} \psi_i(t) (\mathbf{f}_d(t) - w_i(x(t)(g - y_0)))^2$$
(8)

and the solution is

$$w_i = \frac{\mathbf{s}^T \Psi_i \mathbf{f}_d}{\mathbf{s}^T \Psi_i \mathbf{s}} \tag{9}$$

where

$$\mathbf{s} = \begin{pmatrix} x_{t_0}(g - y_0) \\ \vdots \\ x_{t_N}(g - y_0) \end{pmatrix}, \quad \Psi_i = \begin{pmatrix} \psi_i(t_0) & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \psi_i(t_n) \end{pmatrix}$$
(10)

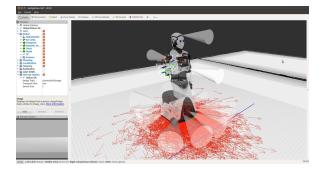


Figure 2: Rviz visualizer with the REEM robot. Different data from sensors is shown.

3. Software resources

This Section is mainly devoted to explain the integration effort of software existing tools. They will be shortly introduced with the aim to understand how the implemented system will work. The Robot Operating System (ROS) (Foundation, 2014) framework is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools to visualize and create applications.

3.1. Data acquisition and simulation

Software packages employed has been the following,

- *Rosbag*. A set of tools for recording from and playing back to ROS topics in a high performance way (Thomas, 2014).
- *YAML*. "YAML Ain't Markup Language" (Evans, 2014) is a human-readable data serialization format.
- *Rviz.* A 3D visualizer for displaying sensor data and state information from ROS (Hershberger et al., 2014) (see Figure 2).
- *REEM robot software*. REEM runs ROS. A couple of implemented packages (Robotics, 2014) that are important for this work are: the URDF implementation of the robot description and the play_motion package which stores previously known movements in a way points fashion and allows the user asking to reproduce these movements.

3.2. DMPs generation

Scott Niekum DMP's. A package called dmp (Niekum, 2014) has been developed by Scott Niekum which provides a general implementation of Dynamic Movement Primitives (DMPs) (Konidaris et al., 2013). This package exposes three services:

- learn_dmp_from_demo: Given a demonstration trajectory and DMP parameters, return a learned multidimensional DMP.
- set_active_dmp: Sets the active multi-dimensional DMP that will be used for planning.
- get_dmp_plan: Creates a full or partial plan from a start state to a goal state, using the currently active DMP.

3.3. Movement

The MoveIt! framework (Coleman et al., 2014) will be extensively used, so some parts of it will be outlined next.

Collision detection. Collision checking is performed taking into account the robot status (Pan et al., 2012). It makes use of the library Flexible Collision Library (Pan, 2014) from Willow Garage.

Kinematics. Forward kinematics and Jacobians are integrated. The default inverse kinematics plug-in for MoveIt! is configured using a numerical Jacobian-based solver.

Controller manager. Manager that takes care of all the available controllers and splits trajectories and synchronizes them.

Joint trajectory controller. Not developed as part of MoveIt! but MoveIt! takes for granted a joint trajectory controller (Tsouroukdissian, 2014) working underlying its controller manager.

4. Implementation

The different parts of the system and how they work will be explained in this Section. A general overview of a complete cycle of learning a gesture, which can be learnt from different sources, creating a trajectory and safely executing it can be seen in the diagram in Figure 3.

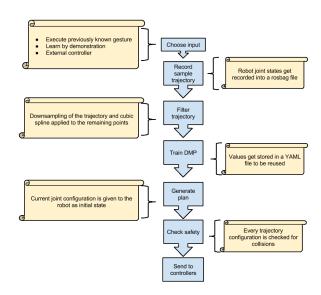


Figure 3: Diagram for the process from learning to executing a gesture.

4.1. Recording motions

Three main input sources will be employed to perform learning from demonstration.

Learning from previously known gestures. Gestures created by hand (i.e.: specifying joint configurations joint by joint manually and testing slowly) were played back and recorded. This approach implies an automated way of executing and recording these gestures.

Learning from robot demonstration. After creating an interface to lower the max current of the joints to effectively be able to move them by hand, motions were taught directly by moving the robot.

The service based interface requests the actual current value (in percentage of the maximum possible of the motor) and sets the desired value (see Figure 4).

Learning from external devices. By accepting end effector poses (as a 6D representing position and orientation) to learn from devices like Razer Hydra or Leap Motion motions were trained. These devices track the user movement of a tool, like a paddle, or directly a human part like the hand and report its position in the world. Also learn-

tartup	Head			Torso	
Diagnostics	head_1_joint		12	torso_1_joint	
sint	head_2_joint			torso_2_joint	
deo	Left arm			Right arm	
unctionalities	arm_left_1_joint	1111		arm_right_1_joint	
entences	arm_left_2_joint			arm_right_2_joint	
elmagine Food	arm_left_3_joint			arm_right_3_joint	
vent - Entertainer	arm_left_4_joint			arm_right_4_joint	
vent - DIPO	arm_left_5_joint			arm_right_5_joint	
vent - Receptionist	0.000 0.00				
vent - DIPO	arm_left_6_joint			arm_right_6_joint	
vent - Entertainer	arm_left_7_joint	1111		arm_right_7_joint	
event - Receptionist	Left hand			Right hand	
ogs	hand_left_thumb_jo	int		hand_right_thumb_joint	
aram	hand_left_index_joir	nt i i i i i		hand_right_index_joint	
lettings	hand_left_middle_jo	int		hand_right_middle_joint	
urveillance					
lovements					

Figure 4: Capture of the web interface to change current limits of motors.

ing from robots TF's¹ has been implemented (to also learn from play motion).

4.2. Generating DMPs

Using the services provided by Scott Niekum we can learn from the three sources of inputs explained in the previous subsection.

4.2.1. Improving recorded data

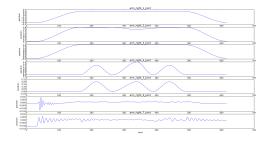
Recording the positions of the joints of the robot consists of subscribing to a source of information (called in ROS a topic) where the state of the joints is published constantly at a rate of 50 Hz. In real life this information has been found to be noisy in some cases. This noise makes the data useless to learn from. A few filtering methods were evaluated (mainly low-pass filters) and, for its ease of implementation and good results, a strategy consisting in down-sampling the signal to later on apply a cubic spline between the down-sampled points was chosen.

An example trajectory where arm_right_6_joint and arm_right_7_joint present jerkiness can be found plotted in Figure 5(a). The velocities recorded on that execution can be found in Figure 5(b), which confirm the continuous jumps in the direction of rotation of these joints.

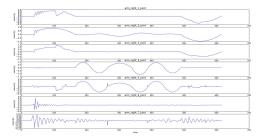
After deploying a $85\%^2$ down-sampling of the signal and then connecting the trajectory points with a cubic spline, we can observe the improved result in Figure 6.

¹The robot TF tree is the representation of all the robot parts and its current position in the space. This enables learning, for example, the trajectory of the tip of the hand.

²This ideal value was found after testing different values in the trade of similarity with the trained gesture and the smoothness of the resultant trajectory.



(a) Original captured trajectory positions.



(b) Original captured trajectory velocities.

Figure 5: Original captured trajectory showing evident jerkiness.

The trajectory is currently smoother, which makes the system able to generate trajectories that will don't make the robot shake.

4.2.2. Parameter tuning

There are some parameters to be tuned in the use case for learning from a demonstration and also when asking for a new plan from what we learnt.

DMP Parameters. In this work and after playing with several values for α and β parameters in the dynamics equation (1), they were left as in the documentation. For tuning the number of basis functions, it was empirically found that a value equivalent to 20 bases per every second of the duration of the recorded gesture gave the best results. To evaluate these values a plan with initial and final conditions coincident with the recorded ones in the original trajectory was generated. The similarity between the recorded trajectory and the generated one was analyzed.

Another strategy was evaluated which consisted of the quantization of how much movement had a trajectory to

minimize the number of basis functions. This strategy made the training faster but lead occasionally to sudden movements, which are dangerous for the robot's mechanical parts.

Trajectory Parameters. From all the set of parameters, the only one that really needed some fine tuning was the time resolution of the plan in seconds. It was found that the robot controllers behave better with short steps in the trajectory so a value of 0.02 seconds was chosen. A lower value will slow all the needed computations and will don't add too much smoothness perceived in the execution in the robot.

4.3. Checking trajectories generation

To compare recorded data and generated data from the same initial and final points, position and velocity plots were made to check the system.

Position graphs. Firstly, it can be observed that no jerkiness exists in the improved recorded demonstration trajectory and this movement is quite slow and short (Figure 7(a)).

Next, we learn from that demonstration and we generate a plan from the same initial point to the same final point as learnt in the demonstration (Figure 7(b)).

Finally, the execution of this trajectory is shown in Figure 7(c). This trajectory is almost identical to the previous ones. There is a little difference between the plots depending on the exact timing of start and ending of the trajectories, also the down-sampled trajectories have lesser points but the main focus is to appreciate that the nature of the movement is still the same. The scale also varies minimally.

Velocity graphs. A similar systems check has been performed about velocities. Initially, the trajectory found in Figure 8(a) was recorded.

We don't down-sample the velocity as the DMP server learns only from position and time information. Figure 8(b) shows the velocities of the generated plan from the DMP server. Notice how they are smoother than the original ones.

Finally, the velocities of the resulting execution of the plan are depicted in Figure 8(c). It can be noted that they are a bit less jerky than the original ones.

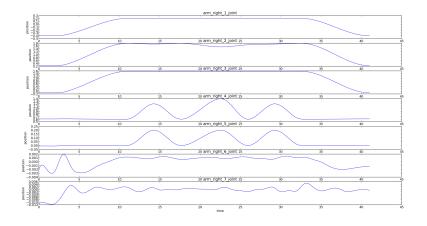


Figure 6: Final trajectory used as demonstration after down-sampling and using cubic splines.

4.4. Checking collisions

Data manipulation must be performed to be able to send plans to joint controllers. DMP plans include positions, velocities and times and they are robot agnostic, in the sense that they do not take into account the particular geometry of the robot. Hence, it is mandatory to check that provided trajectories are safe.

Using the service explained in Section 3.3 every trajectory point is checked to avoid robot collision with itself or the environment. If any point fails, the execution is not started.

The collision checking process is a computationally expensive operation. On real tests it was found that collision checking of a full generated trajectory took roughly 1/3 of the expected duration of the executed trajectory. For instance, a long trajectory called crowd_salute has a duration of 14 seconds. The computation of collisions took approximately 4 seconds. This trajectory is composed by about 600 points. Every collision checking call takes on average 0.007 seconds.

Such a large delay when asking for reproducing a gesture is not desirable so some research was completed trying to alleviate this issue. Down-sampling the trajectory to check collisions on only some points was considered but a calculation of the worst case movement discarded this approach: as every trajectory point is generated with 0.02 seconds of difference and a joint usually has its limit on 1 meter per second velocity, if all motors in an arm moved at the same time this would imply a movement of 0.14 meters in the end effector which has been considered too risky.

5. Results

Several experiences were implemented to illustrate the results. They show not only results from this work but also some other abilities of the robot (like voice recognition and text to speech) to perform the experiences more appealing and give a vision about how this work will be integrated.

5.1. Learning from play motion

In this demo the robot performs a pre-recorded movement that was created by hand which makes a typical waving motion. This movement is recorded and fed to the DMP server to learn from it. Next, a plan from the same starting pose and to the same ending pose is requested and then finally executed.

A video has been made public (Pfeiffer, 2014c), here some parts of the movement have been displayed as images (see Figure 9).

In this demo there is a difference in the timing during the execution of the learnt gesture. It is executed a bit faster than the original one because the recording step stops recording from the moment the robot reports it finished the motion, which may happen shortly before it actually finishes executing it. Hence, the experiment fails to show that the movement is almost identical playing both videos at the same time but on the other hand let us show that changing the timing of the motion still preserves the motion nature.

5.2. Learning by demonstration

A video has been made public (Pfeiffer, 2014a) about the experiment. Here some parts of the movement can be seen as images (see Figure 10).

In this experience the robot starts by asking the user which joint group wants to use, the options are *right arm*, *right hand*, *left arm*, *left hand* or a combination of any of them. The user chooses one with a sentence similar to "I want to use right arm". The robot asks for confirmation and then lowers the current of the desired joint group, in this case *right arm*, so the user is able to move it freely.

Next, the user sets the initial position of the gesture and orders the robot to start recording the motion. REEM replies confirming it understood the order and starts recording.

The user performs a motion. When he/she wants stop the recording of the joint states, he/she says "*wait here*". The learning process starts.

Once the motion is learnt, which takes a time proportional to the number of joints involved in the learning and how long the motion was, REEM informs the user that it can move its arm to the position where he/she wants to start the learnt gesture.

The user sets this pose andlet REEM know it may compute the gesture from its current position (see Figure 11). REEM will give back the normal level of maximum current to the joints, check if the generated trajectory is safe and finally execute the motion (see Figure 12).

Once the motion is finished, REEM asks the user to set another initial position to generate a new gesture, entering in a loop until the user wants to stop it.

Whether the user sets an initial pose that provokes that the trajectory generated by the DMP is in collision at some point, the trajectory will don't be executed and the user will be informed.

5.3. Learning from end effector

A video has been made public (Pfeiffer, 2014b), here some parts of the movement are depicted.

This experience has been executed only on simulation for safety reasons. In this demo, the user designs a trajectory in the 6D world (i.e. in position and orientation of an end effector) using an external device, in this case a Razer Hydra controller. The input mechanism is generic so anything that can give a 6D pose can be used as an input. This demo has been tested also with a Leap Motion controller and using the own robot hand tip when executing a previously known gesture as the reference pose.

Once this trajectory is recorded, a DMP in the Cartesian space is learnt. Now, the system will ask for the inverse kinematics (IK) necessary for every point in the trajectory, this is done by giving to every call the previous robot state as the initial position to try to ensure the most continuous movement as possible. Whether the shown trajectory is quite simple, it usually works, but as the movement involves a big area there are many configuration changes and there are large shakes in the trajectory.

Once we have all the trajectory in the joint space (every end effector Cartesian pose was translated to a joint configuration in the joint group) we can skip the safety checking step that was used in the other demos as the IK calls ensure us that there is no collision in the given configuration.

6. Conclusions

A system for learning and executing gestures in a humanoid robot has been developed and implemented. It involves the integration of many layers of software from quite low level to very high level cognitive concepts.

Gestures are represented via the use of dynamical movement primitives on the robotic platform REEM. It has been demonstrated that the use of DMPs is a very handy way of learning motions for complex robots and it has been integrated in some experiences in an easy-to-use software. The REEM robot is currently able to learn not only gestures but also compose tasks by learning different steps of them.

As a future work, some issues could be expanded: trajectory checking could be faster; trajectories could be smoothly re-planned during their execution; training of

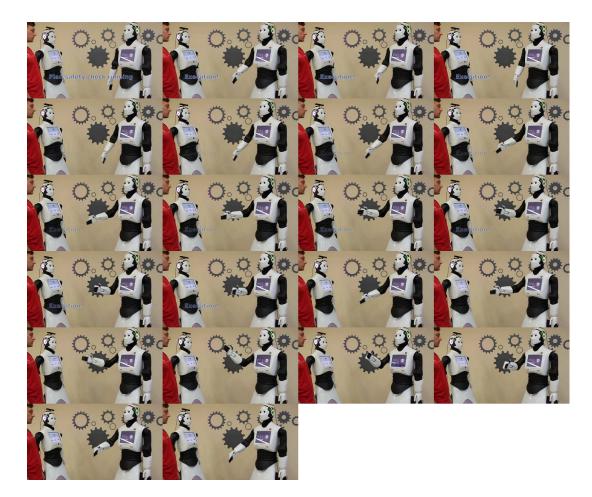


Figure 12: REEM performs the taught gesture.

gestures with an external device could be made safer and References also faster.

Acknowledgements

We would thanks to PAL Robotics for letting us work on this project with robot REEM in their headquarters and to all the great people therein that have been interested in this work.

This research was supported in part by the PATRICIA Research Project (TIN2012-38416-C03-01), funded by the Spanish Ministry of Economy and Competitiveness.

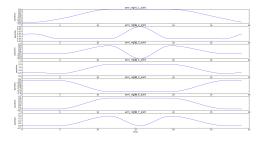
- Argall, B.D., Chernova, S., Veloso, M., Browning, B., 2009. A survey of robot learning from demonstration. Robotics and Autonomous Systems 57, 469-483.
- Breazeal, C., Kidd, C., Thomaz, A., Hoffman, G., Berlin, M., 2005. Effects of nonverbal communication on efficiency and robustness in human-robot teamwork, in: Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, pp. 708-713. doi:10.1109/IROS.2005.1545011.
- Calinon, S., Billard, A., 2007. Incremental Learning of Gestures by Imitation in a Humanoid Robot, in: Pro-

ceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 255-262.

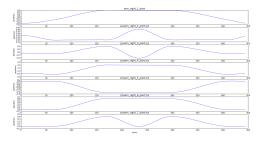
- Coleman, D., Sucan, I.A., Chitta, S., Correll, N., 2014. Reducing the barrier to entry of complex robotic software: a MoveIt! case study. Computing Research Repository abs/1404.3785. URL: http://arxiv. org/abs/1404.3785.
- DeWolf, T., 2014. Dynamic movement prim-The basics. itives part 1: URL: http: //studywolf.wordpress.com/2013/11/16/ dynamic-movement-primitives-part-1-the-basics?nd smooth trajectory computation in cluttered envi-
- Evans, C.C., 2014. YAML official website. URL: http: //www.yaml.org/.
- Foundation, O.S.R., 2014. ROS official website. URL: http://www.ros.org/.
- Hershberger, D., Gossow, D., Faust, J., 2014. rviz ROS wiki. URL: http://wiki.ros.org/rviz.
- Hostetter, A.B., Alibali, M.W., Kita, S., 2007. I see it in my hands eye: Representational gestures reflect conceptual demands. Language and Cognitive Processes 22, 313-336. URL: http://dx.doi.org/10.1080/ 01690960600632812,
- Ijspeert, A.J., Nakanishi, J., Hoffmann, H., Pastor, P., Schaal, S., 2013. Dynamical movement primitives: Learning attractor models for motor behaviors. Neural Computation 25, 328-373.
- Konidaris, G., Kuindersma, S., Niekum, S., Grupen, R., Barto, A., 2013. Robot learning: Some recent examples, in: Proceedings of the Sixteenth Yale Workshop on Adaptive and Learning Systems, Center for Systems Science, Department of Electrical Engineering, Yale University. pp. 71–76.
- Lohse, M., Rothuis, R., Gallego Perez, J., Karreman, D.E., Evers, V., 2014. Robot gestures make difficult tasks easier: the impact of gestures on perceived workload and task performance, in: CHI '14 : proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA. pp. 1459-1466.

- Moshkina, L., Trickett, S.B., Trafton, J.G., 2014. Social engagement in public places: A tale of one robot, ACM, Bielefeld, Germany. pp. 382–389. doi:10. 1145/2559636.2559678.
- Niekum, S., 2014. DMP ROS wiki. URL: http://wiki. ros.org/dmp.
- Pan, J., 2014. FCL official GitHub. URL: https:// github.com/flexible-collision-library/fcl.
- Pan, J., Zhang, L., Manocha, D., 2012. Collision-free
- ronments. International Journal of Robotics Research 31, 1155-1175.
- Pfeiffer, S., 2014a. Learning from demonstration using DMPs. URL: http://youtu.be/Q0J2D9R7m5E.
- Pfeiffer, S., 2014b. Learning from external controller with DMPs. URL: http://youtu.be/h4JSr7aVxvI.
- Pfeiffer, S., 2014c. Learning from previously known gestures using DMPs. URL: http://youtu.be/1_ gYRG4d-g4.
- Puigbo, J., Pumarola, A., Angulo, C., Téllez, R.A., 2015. Using a cognitive architecture for general purpose service robot control. Connect. Sci. 27, 105–117. URL: http://dx.doi.org/10.1080/ 09540091.2014.968093, doi:10.1080/09540091. 2014.968093.
- Radford, L., 2009. Why do gestures matter? sensuous cognition and the palpability of mathematical meanings. Educational Studies in Mathematics 70, 111-126. URL: http://dx.doi. org/10.1007/s10649-008-9127-3, doi:10.1007/ s10649-008-9127-3.
- Robotics, P.A.L., 2014. REEM ROS wiki. URL: http: //wiki.ros.org/Robots/REEM.
- Rückert, E., d'Avella, A., 2013. Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems. Frontiers in Computational Neuroscience 7, 138. doi:10. 3389/fncom.2013.00138.

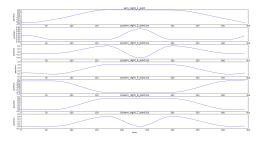
- Salem, M., Rohlfing, K., Kopp, S., Joublin, F., 2011. A friendly gesture: Investigating the effect of multimodal robot behavior in human-robot interaction, in: RO-MAN, 2011 IEEE, pp. 247–252. doi:10.1109/ ROMAN.2011.6005285.
- Schaal, S., 2006. Dynamic movement primitives a framework for motor control in humans and humanoid robotics, in: Kimura, H., Tsuchiya, K., Ishiguro, A., Witte, H. (Eds.), Adaptive Motion of Animals and Machines. doi:10.1007/4-431-31381-8_23.
- Tenorth, M., Beetz, M., 2009. Knowrob knowledge processing for autonomous personal robots, in: Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pp. 4261–4266. doi:10. 1109/IROS.2009.5354602.
- Thomas, D., 2014. rosbag ROS wiki. URL: http://wiki.ros.org/rosbag.
- Trafton, J.G., Hiatt, L.M., Harrison, A.M., Tamborello, F., Khemlani, S.S., Schultz, A.C., 2013. Act-r/e: An embodied cognitive architecture for human robot interaction. Journal of Human-Robot Interaction 2, 30–55. doi:10.5898/JHRI.2.1.Trafton.
- Tsouroukdissian, A.R., 2014. joint trajectory controller ROS wiki. URL: http://wiki.ros.org/joint_ trajectory_controller.



(a) Down-sampled and cubic splined trajectory position.

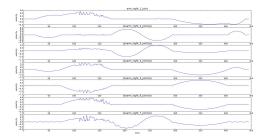


(b) Generated by the DMP server trajectory position.

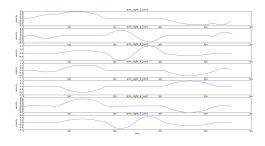


(c) Executed trajectory from the generated by the DMP server trajectory, in position.

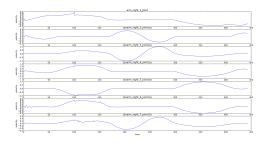
Figure 7: Checking the system. Improved demonstration, generated and executed trajectory position.



(a) Original captured trajectory velocity.



(b) Generated by the DMP server trajectory velocity.



(c) Executed trajectory from the generated by the DMP server trajectory, in velocity.

Figure 8: Checking the system. Improved demonstration, generated and executed trajectory velocity.



Figure 9: REEM performing the original wave motion and the same one generated by using DMPs.



Figure 10: User teaches a gesture by moving the robot arm.



Figure 11: User sets different initial positions for executing the gesture.