Towards an Intelligent System for Supporting Gesture Acquisition and Reproduction in Humanoid Robots

Agnese Augello
ICAR-CNR, Palermo, Italy
agnese.augello@cnr.it

Angelo Ciulla
ICAR-CNR, Palermo, Italy
angelo.ciulla@cnr.it

Alfredo Cuzzocrea
iDEA Lab, University of Calabria, Rende, Italy
alfredo.cuzzocrea@unical.it

Salvatore Gaglio
University of Palermo and ICAR-CNR, Palermo, Italy
salvatore.gaglio@unipa.it

Giovanni Pilato
ICAR-CNR, Palermo, Italy
giovanni.pilato@cnr.it

Filippo Vella
ICAR-CNR, Palermo, Italy
filippo.vella@cnr.it

Abstract

In this paper, an intelligent system for supporting gesture acquisition and reproduction in humanoid robots, which is based on the well-known Microsoft Kinect framework, is introduced and discussed in this paper. The idea that has inspired the paper is represented by endowing an humanoid robot with the capability to mimic the motion of a human user in real time. As a further extension, the latter amenity may serve as a basis for further gesture based human-robot interactions.

1 Introduction

Nowadays, the interaction between human beings and robots has become a very relevant issue in a wide range of applications (e.g., [15, 21, 19]). It is commonly agreed that communication between humans is based on both verbal and not verbal cues. A humanoid robot capable of interacting with people combining speech and gestures would dramatically increase the naturalness of social interactions. On the other hand, other studies like [18, 14, 8] consider *knowledge management techniques* (e.g., [8]) to improve this phase.

Furthermore, the Microsoft Kinect is a popular choice for any research that involves body motion capture. It is an affordable and low-cost device that can can be used for non invasive, marker-less tracking of body gestures. As an example, Baron et al. [5] controlled a Mindstorm NXT artificial arm with sensor Kinect, employing gesture recognition to regulate arm movement. Chang et al. [6] developed a Kinect-based gesture command control method for driving a humanoid robot to learn human actions, using a Kinect sensor and three different recognition mechanisms: dynamic time wrapping (DTW), hidden Markov model (HMM) and principal component analysis (PCA).

Meanwhile, Sylvain Filiatrault and Ana-Maria Cretu [12] used sensor Kinect to mimic the motion of a human arm to an NAO humanoid robot. In their case, the software architecture is based on three modules: Kinect Manager, Interaction Manager, and NAO manager. The Kinect Manager deals with the events and data captured by the Kinect. The class Kinect Transformer is used to get the Euler angles of the desired joints. The Interaction Manager is the intermediary between the Kinect and the robot and contains the repository for the joints used by the other two modules. The use of a joint repository of all articulations allows reducing the data to be processed as some joints are not needed. Finally, the NAO manager contains the static and dynamic constraints to apply to each one of the articulations, as well as the methods that allow the control of the robot movements.

To be sure that the robot has enough time to execute the gesture, a delay of 200 ms between one cycle and the next has been introduced. Itauma et al. [13] used a Kinect to

DOI reference number: 10.18293/DMSVIVA2020-017

teach an NAO robot some basic Sign Language gestures. The aim was teaching Sign Language to impaired children by employing different machine learning techniques in the process. Shohin et al. [16] used three different methods to make a robot NAO imitate human motion: direct angle mapping, inverse kinematics using fuzzy logic and iterative Jacobian.

In some cases, neural networks were used: Miguel et al. [17] used a Kinect sensor and a Convolutional Neural Network (CNN) trained with the MSRC-12 dataset [1] to capture and classify gestures of a user and send related commands to a mobile robot. The used dataset was created by Microsft and had 6244 gesture instances of 12 actions. To have gestures of the same length, without losing relevant information, the system used a Fast Dynamic Time Warping algorithm (FastDTW) to find the optimal match between sequences by non linearly warping them along the time axis. This resulted in all gestures normalized to sequences of 667 frames, with each frame having 80 variables, corresponding to the x,y,z values for each of the 20 joints, plus a separation value for each joint. The resulting 667x80 matrix is used as the input of the CNN, which classifies it in one of the 12 possible gestures. The CNN was trained using two strategies, combined training consisting of a single CNN to recognize all 12 gestures and individual training with 12 different CNN, each capable of recognizing only one gesture. The accuracy rates were 72.08% for combined training and 81.25% for the individual training.

Moreover, Unai et al. [20] developed a natural talking gesture generation behavior for $Pepper$ by feeding a Generative Adversarial Network (GAN) with human talking gestures recorded by a Kinect. Their approach in mapping the movements detected by Kinect on the robot is very similar to what we used, but while they feed the resulting values to a neural network (a GAN), we use the (filtered) values directly.

This paper reports the implementation of a system able to acquire and reproduce the gestures performed by a human during an interactive session. In our approach, we exploited a $MicrosoftKinect$ sensor to capture the motion data from a user and then, we have defined a mapping algorithm to allow a *SoftBank Pepper* robot to reproduce the tracked gestures as close as possible to the original ones.
In particular, we used the *OpenNi* driver for the *Kinect*, the NiTE 2.2 libraries for detecting the user skeleton, and the Kinetic version of ROS with the module pepper_dcm to provide package exchange and bridging between the computer and the robot and Ubuntu 16.04. We focused on the movements of the arms and the head, laying the basis for the extension of the same approach to the remaining parts. The extended version of this paper appears in [4].

2 The Proposed Solution

The developed system is structured in a set of modules, to increase versatility for future projects and to simplify possible extensions to the current project. Besides the Kinect itself, the first module is named *Viewer*, which extracts data frames (consisting of nine float values: three values for a joint position in 3D space, four values for quaternion from the origin and reliability values for both) from the Kinect and sends them in a pipe. The module also provides the feed of the Kinect camera with the overlay of the tracked user's skeleton. The pipe, long 8640 chars (64 chars for each joint value, 9 values for each joint, 15 joints total), is read by the second module, Gesture_Brain.

The *Gesture_Brain* module works both as a gateway for the ROS system [2] and as the module where actual data processing takes place. The gathered data cannot be used directly: a mapping is required to correctly associate each joint user position to the equivalent one in the Pepper robot. For this reason, the data is parsed and structured in a 15×9 float matrix, which is then separated into three matrices: one for coordinates, one for quaternions, and one for reliability values. In our algorithm, we decided to use only the first matrix for simplicity reasons, neglecting the quaternion matrix, and not performing any reliability check on the joints. We assume that the joint data is accurate enough for our purpose, as the Kinect already discards joints whose reliability values are too low. The joint position data is used to estimate Pepper joint angles, specifically shoulder pitch, shoulder roll, elbow roll and elbow yaw for both arms and head yaw for the head (there are three more joint angles that could be estimated, left and right wrist yaw and head pitch, but the Kinect is too imprecise to allow a good estimate, so they have been fixed to a value of 0). The details about this estimation are discussed in the next section.

After all required values are collected, we can use the ROS threads provided by the bridge pepper_dcm to send the joint angles to the robot. These threads consist of multiple joint angles divided into groups, each group representing a body part. As we are interested only in the movement of arms and head, we use three: head, left arm, right arm. The bridge reads the sent values and the time between each capture to dynamically calculate the gesture trajectory in real-time. This means that to allow the system to be as accurate as possible, the gesture should be executed quite slowly. The bridge itself was modified to activate the in-built Self Collision Avoidance (part of the NaoQi library) and to deactivate wait and breathe animations, as they interfere with the commands sent by the pepper_dcm.

```
[ Coordinates ] [ Quaternion ] [ C_conf ] [ Q_conf ]
```

Figure 1. Structure of a single row of the array sent by the Viewer module

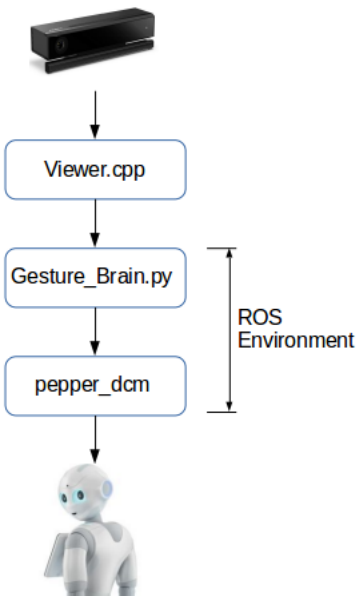

Figure 2. Structure of the algorithm

3 Mapping between User and Pepper

The *Pepper* robot has five Degree of Freedom for each arm (each one associated with a joint), unlike human beings who have seven. A mapping is thus required. From the *Kinect* the Cartesian coordinates for each joint, the quaternion for each segment (both referenced globally), and a reliability value for both are extracted. The bridge *pepper_dcm* uses Euler angles to communicate to the robot the new position of its joint angles. 3D space coordinates are thus used since quaternions have proven unsuitable. This is because the quaternions extracted do not represent the rotation from the previous frame, but rather the rotation from a reference quaternion. This leads to excessive inaccuracies once converted in Euler angles.

Let $\overline{x}$, $\overline{y}$ and $\overline{z}$ be the unit vectors for each axis, that is:

$$\overline{x} = (1,0,0)$$
$$\overline{y} = (0,1,0)$$
$$\overline{z} = (0,0,1)$$

Let S_L , E_L and W_L be the coordinates of the shoulder, the elbow and the wrist of the left arm respectively, $\overline{S_LE_L}$ and $\overline{E_LW_L}$ are defined as:

$$\overline{S_LE_L} = E_L - S_L$$

$$\overline{E_LW_L} = W_L - E_L$$

$SR_L$is the supplementary to the angle between $\overline{S_LE_L}$ and $-x$ axis:

$$SR_L = \frac{\pi}{2} - arcos(\overline{S_LE_L} \cdot -x) \qquad (1)$$

$SP_L$is the angle between the projection of $\overline{S_LE_L}$ on zy plane and z axis, shifted in range to avoid the jump discontinuity at 180 and -180:

$$SP_L = \pi - mod_{2\pi}(\frac{3}{2}\pi + arctan(\overline{S_LE_L}_z, \overline{S_LE_L}_y) \qquad (2)$$

For values of SR_L close to $\frac{\pi}{2}$, SP_L become unstable. As such, in the algorithm is assigned a value of 0 for $SR_L >$ 1.3.

$ER_L$is the angle between $\overline{E_LW_L}$ and $\overline{S_LE_L}$, shifted by $\frac{\pi}{2}$:

$$ER_L = \frac{\pi}{2} + arcos(\overline{E_LW_L} \cdot \overline{S_LE_L}) \qquad (3)$$

$EY_L$is the angle between the projection of $\overline{E_LW_L}$ on zy plane and z axis, shifted in range for stability reasons, plus $-SP_L$:

$$EY_L = \pi - mod_{2\pi}(\frac{3}{2}\pi + arctan(\overline{E_LW_L}_z, \overline{E_LW_L}_y) - SP_L \qquad (4)$$

The right arm is almost the same as the left arm, the only difference is that some angles have the opposite sign. Let $\overline{HN}$ be the difference between the coordinates of the joints H (head) and N:

$$\overline{HN} = H - N$$

The head yaw HY is the angle between the projection of $\overline{HN}$ on the xz plane and the z axis:

$$HY = -arctan(\overline{HN}_z, \overline{HN}_y) - \frac{\pi}{2} \qquad (5)$$

3.1 Line of Best Fit

$Kinect$ joint detection is based on the shape of the user, which is redrawn at every frame. While calibrating the sensor helps to reduce the resulting jerkiness, there is still a significant amount of noise left. This noise can be approximately classified in two categories: a constant Gaussian noise caused by small alteration on the shape detected and large ”spikes” when the $Kinect$ fail to guess the position of one or more joints (especially common when part of the limb is outside of the frame or when two or more joints overlap). A simple way to compensate part of this noise is to use a line of best fit.

Given k points in (x, y) coordinates system, we must find the values c_0 and c_1 in the equation:

$$p(x) = c_0 x + c_1$$

that define the straight line minimizing the squared error:

$$E = \sum_{j=0}^{k} |p(x_j) - y_j|^2$$

in the equations:

$$\begin{gathered} x_0 c_0 + c_1 = y_0 \\ x_1 c_0 + c_1 = y_1 \\ \dots \\ x_k c_0 + c_1 = y_k \end{gathered}$$

The result is a smoother movement, especially when Kinect is not able to detect the precise coordinates of a given joint. This is because, given a disturbing signal, the line of best fit can be seen as an approximation of the tangent that the signal would have at that point if the noise were removed. This is not always true, especially when the signal changes rapidly, but it's close enough in most cases to give a generally cleaner movement.

3.2 Modes of Operation

Besides mimicking the user movement, the Gesture_Brain module also has some additional features implemented to increase the breadth of experiments that can be performed with the system or to help with future projects. The behavior of the program is managed by the input arguments. These are, in order: mode, mirror_flag, json_file_name, LAjpos, RAjpos, Hjpos. The first one determines which of the three different modes of operation will be used (default 0), the second one determines if the mirror mode is activated or not (default false), the third defines the name of the text file used to record (in mode 0 and 1) or read (mode 2) the gestures (the default value is NULL, that is no recording) and set a flag (record_flag) to 1, the fourth, fifth and sixth ones are used to determine the pose to use in mode 1 (as default, the robot will spread its arms parallel to the ground, in a pose that in animation is known as "T-pose"). More in details, the modes of operation of the main program are:

- Mode 0 or "Mimic Mode", is the default mode and makes the robot mimic the movements of the user. The record flag makes it, so the output is not just sent to the ROS publishers, but recorded in a JSON(JavaScript Object Notation) file, to be reproduced later. If the mirror flag is active, every movement is mirrored. In case both the record and mirror flags are active, the mirrored movement will be recorded and saved in the specified txt file.
- Mode 1, or "Pose Mode", make the robot execute a pose (defined at the beginning by the value of the given arguments) that the user must try to emulate. A distance algorithm calculates how close is the user pose to that of the robot, evaluated separately for the head, right upper arm, left upper arm, right forearm, and left forearm. If the user pose keeps all body parts below their respective thresholds (defined separately for each boy part), the program will communicate the success and shut down. The record flag makes it, so the distance values returned are written in a file, while the mirror flag makes it so the user must try to mirror the pose shown.
- Mode 2, or "Playback Mode", consists of reproducing a previously recorded gesture. The mirror flag, even if selected, doesn't have any effect on the algorithm. The name necessary to activate the record flag is used as the name of the file with the gesture to execute.

As an example, an experiment that was conceptualized consisted in using the Pepper robot to show a specific pose that the user must replicate as closely as possible. The experiment envisaged the use of both the normal mode and the mirror mode.

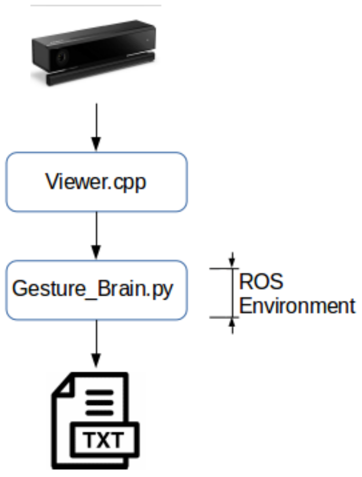

Figure 3. Structure of the algorithm when recording (the text file can be either the recorded gesture in mode 0 or the record of distance values in mode 1)

4 Conclusions and Developments

The system illustrated in this paper is capable of detecting the user poses with the Kinect with sufficient accuracy. The first experiments show that the reproduced movements are precise and smooth; the mirroring is accurate; the pose

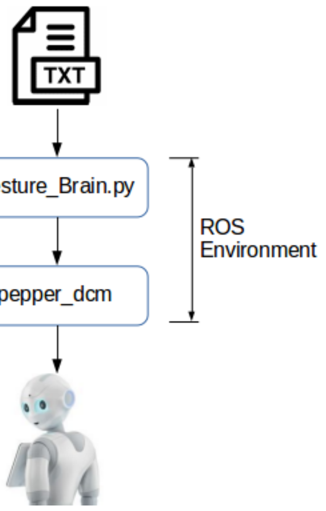

Figure 4. Structure of the algorithm in mode 2 (the txt file in this case is the coding of a previously recorded gesture)

evaluation is coherent; Furthermore, the recording and execution of the gestures are very close to the real-time movements. However, sometimes, certain positions cannot be reliably detected, due to imprecise behavior of the Kinect output when joints overlap each other, and to excessive reliance on the silhouette to detect the human body and the lack of joints in key points of the detected skeleton (like the hands). There is also an environmental factor, like lightning and positioning, that can make accurate user detection problematic. Currently, we are setting up two experiments: the first one is to make the robot autonomously capable of acting both as an instructor and a learner of the Semaphore Flag Signalling System [3], exploiting the gesture mirroring features; the second one is to make the robot capable of both encoding and decoding simple sentences from natural language to the flag semaphore system and vice-versa.

In future works, we plan to extend our framework as to deal with novel and emerging *big data trends* including performance (e.g., [10, 7]), and privacy and security (e.g., [9, 11]).

References

[1] Msrc-12 dataset, `https://www.microsoft.com/en-us/download/details.aspx?id=52283`.
[2] Ros kinetic, `http://wiki.ros.org/kinetic`.
[3] Semaphore flag signalling system, `https://en.wikipedia.org/wiki/Flag\_semaphore`.
[4] A. Augello, A. Ciulla, A. Cuzzocrea, S. Gaglio, G. Pilato, and F. Vella. A kinect-based gesture acquisition and reproduction system for humanoid robots. In *Computational Science and Its Applications - ICCSA 2020 - 20th International Conference, Cagliari, Italy, July 1-4, 2020, Proceedings*, 2020.
[5] G. Baron, P. Czekalski, D. Malicki, and K. Tokarz. Remote control of the artificial arm model using 3d hand tracking. In *2013 International Symposium on Electrodynamic and Mechatronic Systems (SELM)*, pages 9–10. IEEE, 2013.
[6] C.-w. Chang, C.-j. He, et al. A kinect-based gesture command control method for human action imitations of humanoid robots. In *2014 International Conference on Fuzzy Theory and Its Applications (iFUZZY2014)*, pages 208–211. IEEE, 2014.
[7] G. Chatzimilioudis, A. Cuzzocrea, D. Gunopulos, and N. Mamoulis. A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement. *J. Comput. Syst. Sci.*, 79(3):349–368, 2013.
[8] A. Cuzzocrea. Combining multidimensional user models and knowledge representation and management techniques for making web services knowledge-aware. *Web Intelligence and Agent Systems*, 4(3):289–312, 2006.
[9] A. Cuzzocrea and E. Bertino. Privacy preserving OLAP over distributed XML data: A theoretically-sound secure-multiparty-computation approach. *J. Comput. Syst. Sci.*, 77(6):965–987, 2011.
[10] A. Cuzzocrea, R. Moussa, and G. Xu. Olap*: Effectively and efficiently supporting parallel OLAP over big data. In *Model and Data Engineering - Third International Conference, MEDI 2013, Amantea, Italy, September 25-27, 2013. Proceedings*, pages 38–49, 2013.
[11] A. Cuzzocrea and V. Russo. Privacy preserving OLAP and OLAP security. In *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*, pages 1575–1581. 2009.
[12] S. Filiatrault and A.-M. Cretu. Human arm motion imitation by a humanoid robot. In *2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings*, pages 31–36. IEEE, 2014.
[13] I. I. Itauma, H. Kivrak, and H. Kose. Gesture imitation using machine learning techniques. In *2012 20th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2012.
[14] M. C. Lau, J. Anderson, and J. Baltes. A sketch drawing humanoid robot using image-based visual servoing. *Knowledge Eng. Review*, 34:e18, 2019.
[15] C. A. Monje and S. M. de la Casa Díaz. Modeling and control of humanoid robots. *Int. J. Humanoid Robotics*, 16(6):1902003:1–1902003:3, 2019.
[16] S. Mukherjee, D. Paramkusam, and S. K. Dwivedy. Inverse kinematics of a nao humanoid robot using kinect to track and imitate human motion. In *2015 International Conference on Robotics, Automation, Control and Embedded Systems (RACE)*, pages 1–7. IEEE, 2015.
[17] M. Pfitscher, D. Welfer, M. A. d. S. L. Cuadros, and D. F. T. Gamarra. Activity gesture recognition on kinect sensor using convolutional neural networks and fastdtw for the msrc-12 dataset. In *International Conference on Intelligent Systems Design and Applications*, pages 230–239. Springer, 2018.
[18] P. Regier, A. Milioto, P. Karkowski, C. Stachniss, and M. Bennewitz. Classifying obstacles and exploiting knowledge about classes for efficient humanoid navigation. In *18th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2018, Beijing, China, November 6-9, 2018*, pages 820–826, 2018.
[19] S. Saeedvand, H. S. Aghdasi, and J. Baltes. Robust multi-objective multi-humanoid robots task allocation based on novel hybrid metaheuristic algorithm. *Appl. Intell.*, 49(12):4097–4127, 2019.
[20] U. Zabala, I. Rodriguez, J. M. Martínez-Otzeta, and E. Lazkano. Learning to gesticulate by observation using a deep generative approach. *arXiv preprint arXiv:1909.01768*, 2019.
[21] A. Zhang, I. G. Ramirez-Alpizar, K. Giraud-Esclasse, O. Stasse, and K. Harada. Humanoid walking pattern generation based on model predictive control approximated with basis functions. *Adv. Robotics*, 33(9):454–468.