DE GRUYTER OPEN

Print ISSN: 1312-2622; Online ISSN: 2367-5357
DOI: 10.1515/itc-2017-0023

Wireless Kinect-NAO Framework Based on Takagi-Sugeno Fuzzy Inference System

A. Lekova, A. Krastev, I. Chavdarov

Key Words: *Teleoperation; telerobotics, Takagi–Sugeno fuzzy system; view invariance; Microsoft Kinect V2; NAO robot.*

Abstract. *In the context of learning new skills by imitation for children with special educational needs, we propose Wireless Kinect-NAO Framework (WKNF) for robot teleoperation in real time based on Takagi-Sugeno (T-S) Fuzzy Inference System. The new solutions here are related to complex whole-body motion retargeting, standing body stabilization, view invariance and smoothness of robot motions. The raw depth Kinect data are fuzzified and processed by median filter. The joint angles estimation for motion mapping of Human to NAO movements is based on fuzzy logic and featured angles rather than direct angles are calculated by Inverse Kinematics due to differences in the human and robot kinematics. During the joint angles calculation nonlinearities are observed as a result of ambiguity of Kinect 3D joint coordinates in different offsets. NAO kinematic limitations and nonlinearities in workspace are decomposed and linearly approximated by T-S fuzzy rules of zero and first order that have local support in 2D projections. To prevent the robot to fall down, the center of mass is considered in order NAO to stay within a support and safe polygon. The feasibility of the proposed framework has been proven by real experiments.*

1. Introduction

Imitation involves a child's ability to copy others and helps children to learn new things and movements. Unfortunately, children with special educational needs often have difficulty with imitation. They lack the ability to share a focus on humans, however are attracted to robots and computerized technologies. Therefore, we seek for joyful play environment exploiting assistive technologies to enhance children's ability to imitate. In the context of the project [1], a Microsoft Kinect sensor [2] is used for motion-sensing and mediating the process of doing things by teleoperation to replicate the human movements on the robot. Nowadays, teleoperation (or also motion retargeting) is achieved by sensors on the human or by external observations over time. Since the marker based motion capturing systems are expensive, require careful calibration and are hard to use in daily life, a lot of work has been done to study imitation by external observations for extracting 3D poses from an image sequence. Tracking the human motion is an attempt a kinematic model of the robot to be recovered from the video sequences and to be an input for the kinematic modules of the robot. The teleoperation process has two stages: the operator's calibration stage and motion mapping to the robot. During the complex whole-body motion retargeting from video observations, considering the positions of the endeffectors and the center of mass over the support polygon to avoid falls of the robot are the most important aspects in direct imitation. Another concerns are that the robot may have insufficient degrees of freedom to satisfy the desired motion or circumstances of redundancy where unique solution is not guaranteed.

1.1. Problem Formulation

In the present study, Kinect V2 sensor is used for teleoperation of Aldebaran NAO humanoid robot V 2.1.4 [3]. These two technologies used for imitation in the context of a play as learning environment are expected to be easily set-up at day-care centers or schools from people without engineering skills. So, the calibration of their integration should be easy. Observing the changes in the human and robot movement we faced a variety types of problems during design and implementation coming from the following requirements: (1) need of Kinect data smoothing and filtering; (2) need of real-time operation; (3) smooth functions for motion retargeting without false sudden shifts that cause the robot to move abruptly; (4) view-invariance and scalability of the Inverse Kinematics solutions; (5) NAO standing stability.

Motion retargeting should be online and in real-time. There are two Aldebaran's Inverse Kinematics (IK) functions on the robot side to control the arm's joints and move the hand point to a given position [3] by passing the coordinates of a hand end positions using transformation matrices or direct angle mapping to pass parameters to control of arm's joints. However, Aldebaran's IK function works correctly only after passing the orientation of the hand point from Kinect. Additionally, the noisy Kinect readings cause continuously changing joint angles and results in abrupt movement of the arms even in steady state. The Kinect joint positions data are not perfectly precise, meaning that they are scattered around the correct joint positions in each frame and are accurate within a centimetre range, not millimetre [4]. When the functions using these data are not smooth (e.g. in calculating forearm size that participate in formulae for angles) the sudden Kinect spikes will cause the robot to move abruptly while the user is barely changing pose. During the whole body retargeting, the torso and/or leg joints have to be controlled in parallel with other joints in order to generate and stabilize consistent motions.

1.2. Existing Solutions

A variety of approaches how robots to imitate human motions have been proposed, however most of them consider only upper-body motions, while the legs are neglected.

We are focused only on reported studies presenting real-time systems for NAO teleoperation using visual-based motion observations, mainly obtained from Kinect sensor. The noise and variations of Kinect readings are taken care by the margin of error [5] or by filtering the Kinect data [6]. The double exponential smoothing filter [4] is the most used filter for smoothing Kinect data [6]. During calibration a complex reference coordination between Kinect and robot coordinate frames requires a lot of code and skills that therapists or educators do not have. Often the operator stays at predefined area in front of the Kinect view [7] or complex transformation matrices are used for calibration between Kinect and NAO coordinate systems since different areas of the input space require different compensations and scale independence.

Inverse kinematics is often used for the joint angle calculation, however, it needs relatively high computational cost for the optimization calculation and in case of redundancy a unique solution is difficult to be found. An adaptive neuro-fuzzy inference system (ANFIS) for motion mapping is proposed in [5], where three methods to imitate human *upper body motion* are implemented on a NAO robot and compared: (1) direct angle mapping method (2) IK using fuzzy logic and (3) IK using iterative Jacobian. The direct method requires the coordinates of three joints (shoulder, elbow and wrist) to determine NAO angles. However, continuously changing angles for the positions result in jerky movement. The IK using ANFIS method requires only two joint coordinates and is found to be more efficient and fast because the fuzzy system is trained a priory and there are not many computations involved in mapping the coordinates of the end-effector to the joint angles. However, the training of the ANFIS can take a long time depending on the amount of training data. Moreover, the first method is used to train ANFIS and training phase could not be performed from inexperienced persons. Solving the IK problem iteratively using the Jacobian pseudo-inverse requires two joints as well, but is found to be inefficient because a lot of iterations are required at each step and the response of the robot is very slow. This method also gets stuck in singularities. The authors in [8] presents a different type of real-time inverse kinematics based retargeting system to map human upper limbs motions (tracking by Kinect sensor) safely and smoothly to robot's joints. It considers motion similarity between end-effector motions and between angular configurations. Additional constraints are proposed and solved in the projected null space of the Jacobian matrix.

As it can be seen in the above studies, the legs are neglected. However, during the complex whole-body teleoperation motion retargeting has to be feasible and stable. Direct joint angles mapping is typically impossible because of the differences in the human and robot kinematics, as well as the different weight distribution. Some approaches how to stabilize the balance and standing poses are presented below. The proposed in [9] approach uses a compact human model and considers the positions of endeffectors, as well as the center of mass as the most important aspects to imitate. This system actively balances the center of mass over the support polygon to avoid falls of the robot, which would occur when using direct imitation. For every point in time, the system generates a statically stable pose. In [10] authors propose to use a Particle Filter (PF) for joint angle imitation. PF provides a reasonable solution with a less computational cost to realize a real-time imitation of a humanoid robot through observation of human demonstration. However, PF does not provide the standing stability of the humanoid robot. Authors propose a simple strategy for controlling the hip and ankle joints to provide a reasonable standing stabilization so that it keeps the center of mass within the supporting polygon. Three relatively simple equations stabilize the robot successfully where knee-pitch and hip-pitch angles are estimated by the PF. The idea is that the length between the hip and knee joints is almost equal to the one between the knee and ankle joints and the triangle that consists of hip, knee and ankle joints is an isosceles triangle. If the torso leans forward while all joint angles are fixed except the hip joints, the center of mass moves forward and eventually NAO will fall down forward. Thus, the calculated gain for ankle pitch joint compensates for it by adding an offset. The imitation problem presented in [11] is formulated as finding the projection of a point from the configuration space of a human's poses into the configuration space of a humanoid. An optimal projection is defined as the one that minimizes a back-projected deviation among a group of candidates, which can be determined in a very efficient way. Thus effective projections can be obtained by using sparse correspondence. Authors claim that the overhead of the proposed method by generating these sparse correspondence samples for motion imitation is very light and it fits well for different real-time applications. The quality of a projected configuration is evaluated by two metrics referring to corresponding ground truth and the degree of maximum absolute deviation. Nierhoff in [12] draws special focus on a complete approach to keep the shape of the resulting motion consistent in the three domains of planning, control and reasoning. Different from conventional approaches which favor a strict separation, this work aims at a tight coupling to keep the optimality properties consistent for the entire process chain. Nierhoff presents a common prioritized IK scheme and shows how it can be combined with Laplacian trajectory editing for a continuous re-planning of the desired trajectory. A lot of simulation results and issues have been dedicated to computational complexity and possible improvements to increase the speed of the method.

To summarize, the methods for motion retargeting often suffer from smoothness in motion retargeting, certain levels of latency due to computational overhead or a large set of correspondence samples to search. Continuously changing Kinect 3D positions used in the direct method to determine NAO angles result in jerky movement. Teleoperation is fast if learning algorithms are used because the system is trained a priory and not many computations are involved in the motion retargeting, however the training can take a long time depending on the amount of training data. Solving the IK problem iteratively using the Jacobian matrixes are found to be inefficient because a lot of iterations are required at each step and the response of the robot is very slow. Moreover,

Jacobian method blocks in singularities or in circumstances of redundancy when unique solution is not guaranteed.

To the best of our knowledge we didn't find related works for Kinect–robot teleoperation based on depth data pre-processing and motion retargeting by fuzzy logic. The Takagi-Sugeno (T-S) fuzzy systems [15] are mainly used for robot control, not for approximation reasoning over sensor data, as we use it. We decided to apply it in original way in order to make a trade-off between latency and precision in teleoperation by approximation reasoning. T-S fuzzy inference system is chosen as universal approximator since it presents a low time response using a set of simple functions that require low CPU and memory resources.

1.3. Design Criteria

Considering the above problem requirements, we propose a Wireless Kinect-NAO Framework (WKNF) for teleoperation in real-time based on Takagi-Sugeno Fuzzy Inference System (T-SFIS). The wireless framework connects middleware on Kinect side and NAO robot side to transmit data to robot actuators. NAO desktop applications are used in original way in the design of fuzzy rules, while the NAO *Whole Body Balancer* APIs are instantiated to generate and stabilize consistency in the desired motions. IK and T-SFIS algorithms are online and lightweight in order not to block the wireless connection to robot Python scripts in real time.

During the design and implementation of WKNF we found out several innovative solutions related to complex whole-body motion retargeting that requires standing body stabilization, view invariance and smoothness of robot motions. WKNF works well online and is view invariant considering the parallax effect to normalize the calculated angles and distances. The smoothness in motion retargeting is ensured by fuzzifying of the raw depth Kinect data and afterward processing by median filter. Distances and angles are calculated by vector algebra. The joint angles estimation for motion mapping of Human to NAO movements is based on fuzzy logic and featured angles rather than direct angles are calculated by IK. Featured angles are stable in the vision area of Kinect and at least one of the joints establishing the vectors is not quickly changing joint, resulting in less scattered Kinect readings. We exploit trigonometric functions that stop amplifying the data noise. During the joint angles calculation we observed nonlinearity in mapping of the 3D Kinect to NAO angles in different offsets. This is as a result of ambiguity of Kinect 3D joint coordinates in different offsets and parallax effect. Such nonlinearities are decomposed and linearly approximated by T-S fuzzy rules of zero and first order that have local support in 2D projections according to offsets of body parts or NAO kinematic limitations. We partition the 3D input space and decompose the mapping in several projections such that linear approximations of unknown mapping relation identify the number and parameters of fuzzy sets and fuzzy rules. For controlling the torso and leg joints in parallel we analyse the interdependencies among these joints in Choregraphe – a desktop application that allows to create animations and behaviours [14]. The changes of joint values in time are presented by curves in Choregraphe timeline editor. Another desktop application is Webots for NAO [15] that offers a safe place to test behaviours before playing them on a real robot. Connecting Choregraphe and Webots allows a simulation of NAO motions in a virtual world (*see figure 9*). If the virtual robot falls down, when the joint interdependencies and the leg balance with center of mass are taking into account in the python scripts on the robot side, NAO stays within a support and safe polygon.

2. Proposed Solutions

In WKNF for motion retargeting we have to determine the joint angles for the robot actuators to set a desired trajectory on the basic of positions and orientations of the robot end-effectors. Fuzzy logic is adopted for estimation of joint angles for the NAO robot to imitate the human demonstrated posture. It provides a reasonable solution with a less computational cost to realize a real-time motion retargeting on a robot through observation of human demonstration. Since, the T-SFIS does not provide the standing stability and there is a high risk of NAO to fall down if it follows the joint angles estimated by T-SFIS, we proposes a simple solution in real-time how to control the leg joints for the standing stabilization.

2.1. Processing Kinect Body Data to Solve Inverse Kinematics Task

We analyse the Kinect depth and body stream data and identify the 3D positions of upper body limbs over time. The important joints for motion retargeting of upper limbs are left and right shoulder, elbow, wrist and hand. During the movement, the length and angles between each joint are changing at each frame and some of them are considered as important features to map the Kinect angles to angles of robot actuators. To reduce the spikes, we apply Median filter and we use it only for joints that are more often in "Inferred State", such as hand tips and thumbs. This filter doesn't introduce latency because it doesn't take advantage of the statistical distribution of data or noise.

First we have tried to solve the problem for motion retargeting using direct angle analytical method. However due to its poor performance and ambiguous results, we searched for unique angles that can feature the movement by θ_1 to θ_5 over time and map them to NAO actuator angles. Featured angles are stable in the vision area of Kinect and the joints forming the vectors are not quickly changing joints, resulting in less scattered Kinect readings. The extra joints we use for motion retargeting are head, thumb and tip (*see figure 1a*).

The angle between two vectors defined by the three joints P^{th}, Q^{th} and R^{th} with coordinates: (x_p, y_p, z_p) $(x_q, y_q,$

z_q) and (x_r , y_r , z_r), then the two vectors are $\overrightarrow{PQ}$ nd $\overrightarrow{QR}$. The angle between them is calculated by using equation (1).

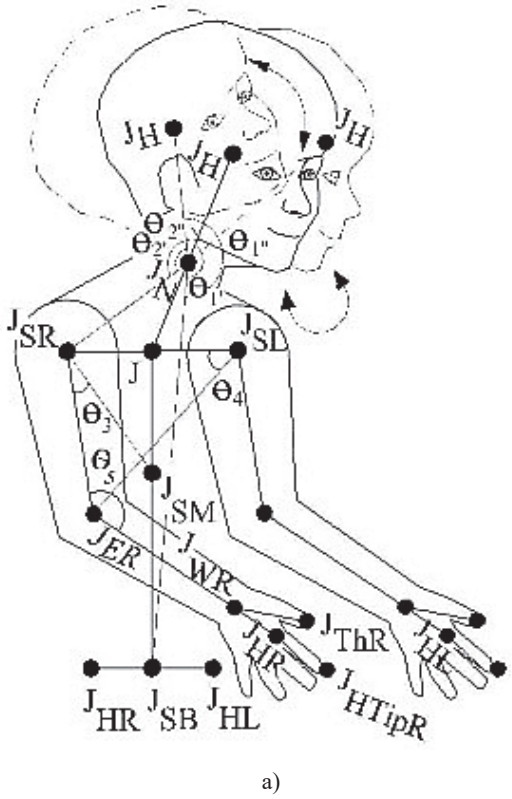
$$(1)\quad \theta_{pqr} = \cos^{-1}\left[\frac{\overrightarrow{PQ}\cdot\overrightarrow{QR}}{\|\overrightarrow{PQ}\|\,\|\overrightarrow{QR}\|}\right]$$

where $\overrightarrow{PQ}\cdot\overrightarrow{QR}$ is the dot product (equation 2) and $\|\overrightarrow{PQ}\|$ and $\|\overrightarrow{QR}\|$ are the lengths (equation 3)

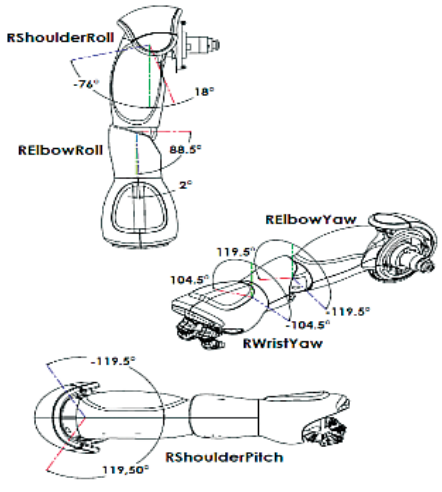
$$(2)\quad \overrightarrow{PQ}\cdot\overrightarrow{QR} = (x_q - x_p)*(x_r - x_q) + (y_q - y_p)*(y_r - y_q) + (z_q - z_p)*(z_r - z_q)$$

$$(3)\quad \|\overrightarrow{PQ}\| = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2 + (z_q - z_p)^2}$$

During the implementation we established that the noisy Kinect readings result in noisy calculations for distances (we use Euclidean formula) and 3D angles, different in the different area of the input space. One of the reasons is that operations to calculate body part sizes and relative positions such as addition, subtraction and multiplication, amplify the noise. Trigonometric functions which are typically used for calculating joint angles affect the noise in different ways. We use arccosine that results in small radian values and even decrease the noise. For unavoidable dependency on offsets of body parts, we implement fuzzy rules that have local support in slit planes in 2D projections (*see figure 4*). Another reason for incorrect calculations is the observed parallax effect.



a)



b)

Figure 1. a) Important joints for motion retargeting of upper limbs; b) 3D model of the NAO right upper limb

2.2. View Invariance and Scalability

Parallax arises due to change in viewpoint occurring to motion of the observer. Since the visual angle of an object projected onto the retina decreases with distance, this information can be combined with previous knowledge of the object's size (etalon) to determine the absolute depth of the object. By measuring angles, and using geometry, one can determine the length of an object – *x* if *r* is the radius between the object and the eye. From the Kinect's eye perspective, the changes in sizes a Kinect would see when the user move far and close we also explain with motion parallax effect. Consequently, the depth coordinate Z affects angle calculations because the lengths of body parts participate in the formulae. However, if we have an etalon (size(s) taken during the Kinect calibration stage) we can measure the change in the angle relative to human position in front of Kinect and to correct angles on the Kinect side according to current visual angle (α_{cur}) using equations (4). In WKNF, the 2D distances for closeness are normalized either by the depth value Z (for tip and thumb joints) or by dividing to αcur (for bigger limbs such as hip and elbow).

$$(4)\quad \alpha_{etallon} = x/r*180/\pi; \qquad \alpha_{curr} = x_{curr}/r_{curr}*180/\pi$$
$$\alpha_{correct} = \alpha_{etallon} - \alpha_{curr}$$

2.3. The Need of Fuzzy-based Reasoning

Kinect and NAO have different coordinate systems, as well as NAO has joint limitations (*see figure 6a*). Because of the context of the application (play for children) we made a trade-off between the precision of motion retargeting, easy calibration and NAO response delay. We applied approximate reasoning to handle Kinect noise readings and provide a universal approach for motion mapping instead to do a complex calibration between Kinect and NAO coordinate systems.

Fuzzy logic is a superset of classical logic with the introduction of "degree of membership". Uncertainties are presented as fuzzy sets (A_i), which are often expressed by words and interpreted by their membership functions μA. We exploit the Takagi-Sugeno (T-S) fuzzy model as a universal function approximator [15]. Its structure consists of rules in the form

$$(5)\ R_i : IF\ \ x\ \ is\ A_i\ THEN\ \ y_i = a_0^i + \sum_{k=1}^{n} a_k^i x_k$$

where $x = (x_1, x_2,...,x_n) \in D$ is a vector representing the inputs defined on D. A_i is a fuzzy set defined on certain domain (D); y_i is a scalar output corresponding to rule i; a^i_k are the consequence parameters associated to rule i. $i \in \{1,..., p\}$, where p is the number of rules. For a zero-order T-S model, the output level y is a constant ($a_0 = a_k = 0$). The rules are aggregated and defuzzified by using the fuzzy-mean formula

$$(6)\ y(x) = \sum_{i=1}^{p} \mu_{A_{ik}}(x) * y_i \Big/ \sum_{i=1}^{p} \mu_{A_i}(x)$$

where $\mu A_{ik}(x)$ is the degree of fulfilment of i-th rule.

$$\mu_{A_i}(x) = \mu_{A_{ik}}(x_1, x_2,..., x_n) = \mathop{T}_{k=1}^{n'} \mu_{A_{ik}}(x_k)$$

where n' is the number of input variables in i-th rule ($n' \leq n$), $\boldsymbol{T}$ is a type of t-(co)norm as minimum, product, etc.

2.4. Generating Fuzzy Rules

During the design we identify premise and consequence parameters and write a fuzzy-rules base (FRB) for mapping angles from Kinect to NAO coordinate frames over time. We use fuzzy trapezoidal membership functions (simple and fast for calculation), where the upper base of a trapezoid takes care of small scattering, i.e. ignoring the scattering in centimetre range that cause the robot to move abruptly while the user is barely changing pose. We observed nonlinearity during the mapping for some of the 3D Kinect to NAO angles in different offsets. For instance, during the mapping of the angle values to move the hand to a given location, NAO right shoulder roll angle (*see figure 3b*) has constant values in different hand-body offsets, while the 3D Kinect angle θ4 is changing in dependence on distances between 2D joint positions of the right hip and elbow. The proposed solution is to partition the 3D input space in several projections such that a linear approximation of unknown mapping relation is possible in these ranges with accepted error less than 5%. The goal is to obtain the premise and consequence parameters of T-S and the required model error ε. If the error is less than 5% zero-order T-S is used, otherwise first-order, second, etc., till the error gets below 5%. The less rules used (i.e. coarse fuzzy partition) the bigger error. The first step is in several 2D projections to identify the group of states having similar linear approximation of mapping, and to distinguish these states as fuzzy sets. Their membership functions will classify the Kinect angle in degrees into these fuzzy sets. The second step is to find the local approximation linear solutions and to aggregate and defuzzifying them by equation (6). In the next Section, how to design T-SFIS fuzzy rules is illustrated.

2.5. Stabilizing Consistency and Balance in the Desired Motions

We proposes a simple solution in real-time how to control the leg joints for the standing poses stabilization, balance, redundancy and task priority during the movements of the whole body. If the upper joints can be controlled individually, the torso or leg joints have to be controlled in parallel with other joints. For instance, the hip and knee joints are of higher importance in squatting, however other joints as upper limbs and ankles also need to be controlled for stability of NAO properly. We found out an original way how to determine feature angles to facilitate the deriving of T-S fuzzy rules by observing the changes of joint values in time presented by curves in Choregraphe timeline editor. Such curves for pose squat, sit down and ankle back are presented in Figure 7. In the last Section we illustrate how we solved the problem for standing stabilization by incorporating in the python scripts on robot side the NAO *Whole Body Control (WBC) APIs* [16]. Thus we stabilize the motions generated by *Joint control APIs* (they control directly the position of the robot joints) and adapt NAO's behaviour to the situation. WBC is a Generalized Inverse Kinematics (GIK) problem which deals with Cartesian or Joint control, balance, redundancy and task priority. GIK problem is written as a quadratic program which is solved every 20 ms using the C++ open source library [17].

The classical form of a quadratic program is

$$(7)\qquad \min\ \ \frac{1}{2}\left\|Y - Y^{des}\right\|_Q^2 \qquad such\ \ as\ \ \begin{cases} AY + b = 0 \\ CY + d \geq 0 \end{cases}$$

Y: Unknown vector;
Y^{des}: Desired but not necessarily feasible solution;
Q: Quadratic norm;
A, b, C and d: Matrices and vectors which express linear equality and inequality constraints.

For the robot NAO, the unknown vector in equation (7) is composed of velocity of torso (3 translations and 3 rotations) and velocity of all the articulated joints. The equality constraints concern keeping feet in a plane or fixed. The inequality constraints concern joint limits and balance. The Center of Mass is constrained to stay within the support polygon. Y^{des} is composed of Cartesian desired trajectories and joints desired trajectories. The solution obtained is feasible since it fulfils all the constraints and is a compromise between the desired motions [16]. Initial motion is modified to the closest motion which respects balance and/or foot state. For instance, it is really difficult to keep the feet flat by joint control only of *HipYawPitch* joint (the only way to rotate the foot of NAO). However, using with *WBC* the generated motion is stable. The main goal of Balance Constraint is to maintain the Center Of Mass (COM) of NAO inside the support polygon that depends on supported leg – both feet, left or right foot.

3. Implementation and Evaluation

3.1. Technical Specifications

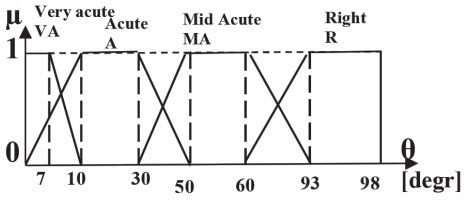
Kinect V2 sensor is connected to a laptop with Intel(R) Core (TM) i7-5500U CPU@ 2.40 GHz and transmits sensor data to a software application running on the laptop, built in C#, referencing Microsoft Kinect library (SDK 2.0) and performing data pre-processing and motion retargeting. Apart from the robot hardware and Kinect sensor, no other hardware is used. Three middleware are connected: Kinect SDK2 [2], NAO.NET on the Kinect side [18] and Naoqi 2.1.4 on the robot side [3]. Apart from those middleware, the framework is built only in C# on the Kinect side and Python scripts on the robot side. These codes run in parallel - first running on a laptop and connected to the Kinect sensor to provide body joint tracking and fuzzy logic processing, and second running on the robot side - waiting to receive data from Kinect side via Wi-Fi and use it to make the move of the robot actuators. The latency of fuzzy reasoning (how much time it takes for robot output to catch up to the actual human joint position when there is a movement in a joint) is not introduced in CPU time it takes for executing the T-SFIS. In general, the delay depends on the number of the simultaneously open Python scripts and the number of per frame passing parameters to these scripts. The reasonable trade-off between precision and latency is proven by real experiments (*figure 2*), where we infer the angles for the robot actuators over time by fuzzy reasoning via numerically processing of the information in the fuzzy rules.

3.2. Kinect Side

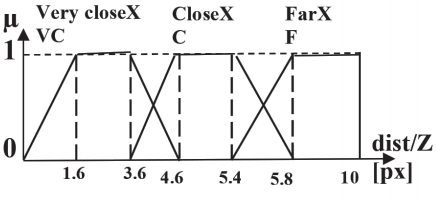
On Kinect side we process the 3D coordinates of joints at each consecutive frame. The applied Median filter output is the median of the last N inputs (joint positions). It latency depends on N and we use it only for N=20 Kinect frames. We calculate the joint angles by analytical IK per frame. The Kinect tracking algorithm operates in 3D camera space, however for some offsets (such as closeness), limb sizes and angles we use the Kinect Coordinate Mapping to project 3D points from camera space to a row/column location in the depth space with origin x=0, y=0 corresponding to the top left corner of the depth image. Thus, we operate only with positive values for joint coordinates.

Modelling of uncertainties consists of information about linguistic variables, domains, constraints as fuzzy sets (*figure 3*) and fuzzifiers. The values of constraints, normalized in the range [0-1], take part in the premise parameters in IF-THEN fuzzy rules, such as Kinect angles and/or 2D distances. The rules map the input values to the output space in terms of implication relation between fuzzy sets in "IF" and functions in "THEN" parts. Fuzzified input data trigger one or several rules in the fuzzy model to calculate the result. Two type of functions for motion mapping of Kinect angles to NAO space in "THEN" parts are designed: function approximation using first-order T-S model and zero-order T-S model.

Figure 2. Real experiments with children



a) for angle θ_3



b) for distances in XZ slit

Figure 3. Premise parameters

To cope with the nonlinearity of the observed Kinect angles in X and Y split planes where the 3D Kinect angles vary (see *figure 4*), we implemented the proposed solution to partition the 3D input space and decompose the mapping of Kinect to NAO in the slit planes in XZ and YZ projections. We use first-order T-S fuzzy rules with linguistic values in premise formed from the semantically close values for angles in different slit planes according to the closeness of the hand to the body. Distances are normalized by current parallax angle. Premise parameters participating in the fuzzy rules are presented in *figure 5*. Consequences are the equations of the trend lines in decompositions.

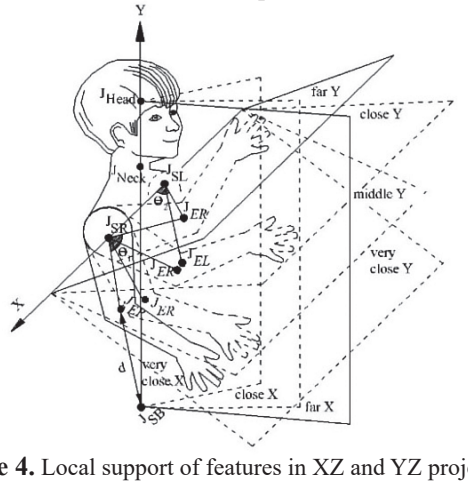


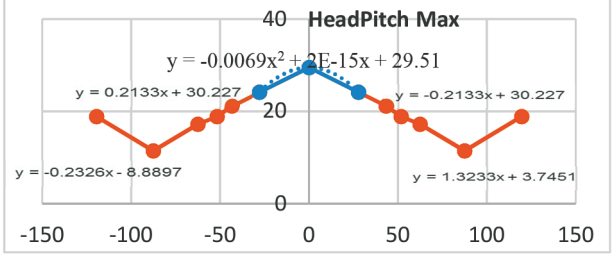
Figure 4. Local support of features in XZ and YZ projections

```
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule(-4.4322
* theta4 + 641.34, new string[] { “All”, “very_closeY” }));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule( -
1.3333* theta4 + 190 , new string[] { “RightObtuse”,
“med_closeY” }));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule(-
5.6129 * theta4 + 834.95, new string[] { “Obtuse”, “med_closeY”
}));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule(-74,
new string[] { “RightObtuse4”, “med_closeY” }));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule( -
0.8333* theta4 + 115, new string[] { “RightObtuse”, “closeY” }));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule(-
2.6761 * theta4 + 394.57, new string[] { “RightObtuse3”, “closeY”
}));
        TSFS_RSR.addRule(new TagakiSugenoFuzzyRule(-
2.5037 * theta4 + 366.96, new string[] { “All”, “farY” }));
```

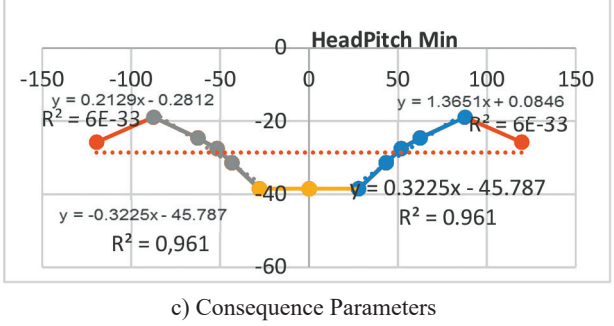
Figure 5. Kinect θ4 mapping to NAO RShoulderRoll angle

HeadYaw	HeadPitch Min	HeadPitch Max
-119.52º	-25.73 º	18.91 º
-87.49 º	-18.91 º	11.46 º
and so on		

a) Anti-collision limitation due to Aldebaran Table [3]



b) Consequence Parameters



c) Consequence Parameters

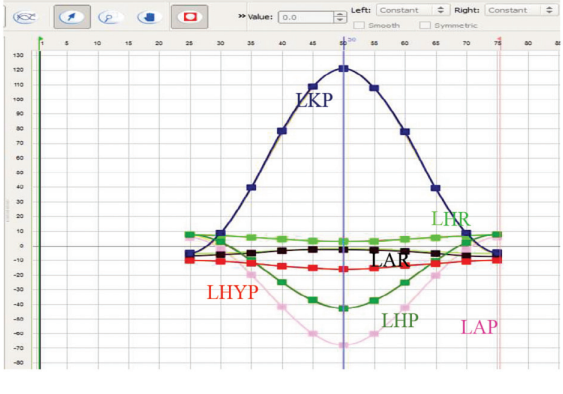
Figure 6. Anti-collision limitations:
HeadYaw /HeadPitch

Decomposition is used also to handle kinematic limitations due to potential shell collision. They are implemented by fuzzy rules as well. As it can be seen in *figure 6a*, at the head level the Pitch motion range is limited according to the Yaw value. First we map the angle HeadYaw by T-S of zero order and then HeadPitch Min and Max according to the first order T-S functions presented in *figure 6b* and *6c*). They show how collision relations between HeadYaw and HeadPitch angles are approximated linearly in four semantically closed groups (forming the premise parameters), the trend lines (forming consequence parameters for each semantic group), their equations and squared errors. If the error is not acceptable, the first step is repeated until deriving premise parameters. Having premise and consequence parameters, we write fuzzy rules for linearly approximation of nonlinearity. Two examples for fuzzy rules of zero and first-order T-S derived from *figure 6c*) are presented below.

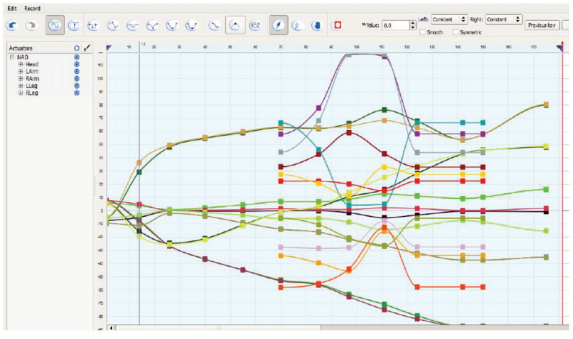
R_3: If $\mu_{\theta 1}$ is acute and $\mu_{\theta 1}$ is right then $\Theta_1 = 40$
R_4: If $\mu_{\theta 1}$ is acute and $\mu_{\theta 1}$ is obtuse then $\Theta_1 =$
$= 0.3225\ \Theta_2 - 45.787$

By using first-order T-S rules rather than zero order, we reduce the number of fuzzy sets used, respectively the number of fuzzy rules in the whole system.

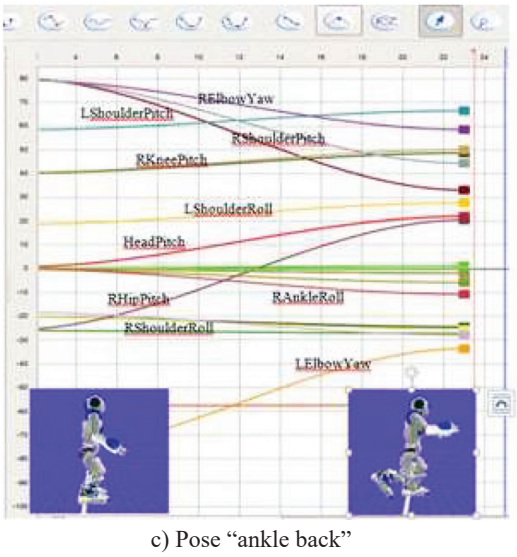
After estimating the torso and leg joints by TS-FIS, we first have tried to control them directly, however nonlinearity from the Kinect measures in different area of the input space resulting in wrong calculated 3D angles and correspondingly inconsistent NAO motions and breakdown. Instead of partitioning the 3D input space and decomposing the mapping of Kinect to NAO angles in slit XZ and YZ planes to model the compensations and scale independences, we tried to find how these joints are controlled in parallel in Choregraphe in order to generate and stabilize consistent motions. We derived the interdependencies among these joints from the Choregraphe timeline editor to understand which one is the feature joint angle to be estimated by TS-FIS. Observing some motion behaviour in time, we choose one or two angles with max deviation to be estimated by TS-FIS and calculate the rest torso/leg angles as functions of feature angle(s). For instance, during the squatting (*figure 7a*) the knee pitch angle is changing to highest degree, then ankle pitch and hip pitch angles. We choose left knee pitch angle as a feature angle and then find functions to calculate the rest (*figure 8*). By analogy to anti-collision relations, these functions might be approximated linearly (if the squared error is bigger than 5%) in several semantically closed groups to form the premise parameters, while the trend lines form the consequence parameters for each semantic group. Part of the pseudo code for TS-FIS on the Kinect side and interdependencies among leg joints are given in the *Appendix 1*. The algorithm how to derive whole-body relations from Choregraphe is presented in *Appendix 2*.



a) Pose “squatting”

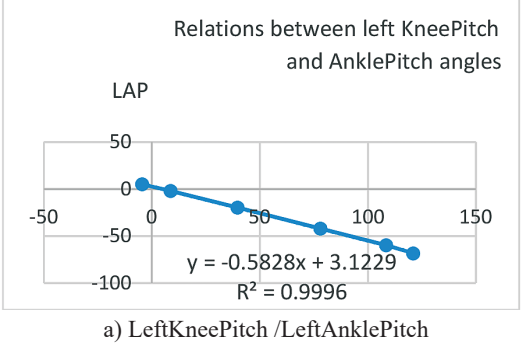


b) Pose “sit down”

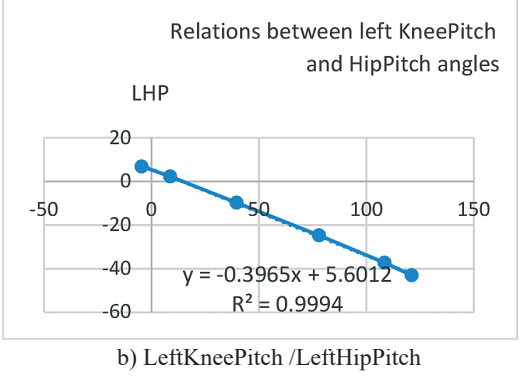


c) Pose “ankle back”

Figure 7. Curves for poses in Choregraphe timeline editor



a) LeftKneePitch /LeftAnklePitch

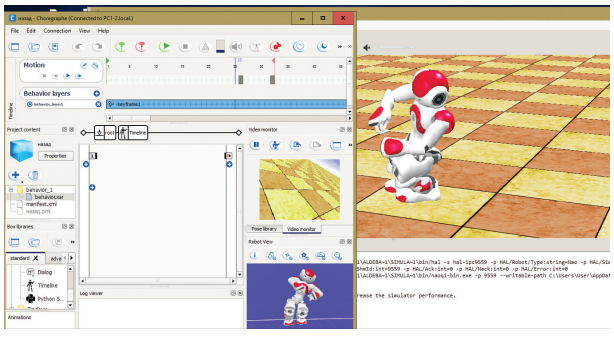


b) LeftKneePitch /LeftHipPitch

Figure 8. Relations for torso and legs angles during squatting

3.3. Testing NAO Behaviour in a Virtual World

Testing behaviours in a virtual world by connecting Choregraphe and Webots allows to understand joint interdependences to avoid real robot falling down (*see figure 9*). For instance, when we move our knee up, the torso, legs and feet joints have to be controlled together in order to design stabilize animation. However, in a result of our experiments if NAO is a little bit unbalanced in Webots, it is enough to instantiate the whole body balance and center of mass APIs in the python scripts. Thus, the behaviour of the real robot is adapted to the situation.



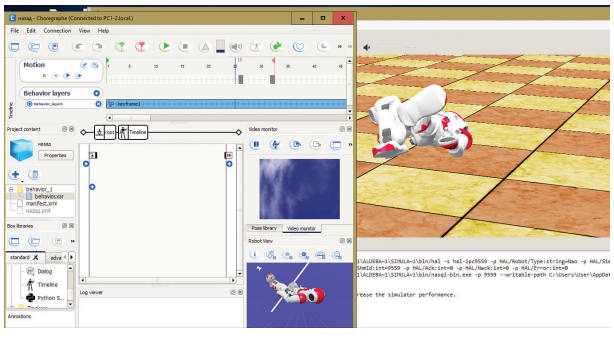


Figure 9. Testing behaviours in a virtual world by connecting Choregraphe and Webots Aldebaran applications

Animation mode might be used in Choregraphe on a real robot (*figure 10*). The algorithm is similar to the described one in *Appendix 2* with the only one difference in

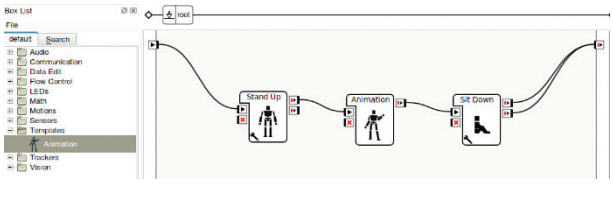


Figure 10. Creating movements from scratch using Animation in Choregraphe

step 4. After creating movements from scratch by stiffening off all joints and changing motor positions manually, the current joint positions are stored in several key frames by pressing NAO start button.

3.4. Robot Side

To stabilize the standing positions we deployed WBC – a feasible solution, since it fulfils all the constraints and is a compromise between the desired motions. Initial motion is modified to the closest motion which respects balance and/or foot state. We illustrate how to incorporate in the python scripts on robot side the NAO Whole Body Control API [16]. The used methods (given in *Appendix 3* in bold) in class *ALMotionProxy* are:

- *ALMotionProxy::wbEnable()*
- *ALMotionProxy::wbFootState()*
- *ALMotionProxy::wbEnableBalanceConstraint()*

The feasibility and real time operation of the proposed WKNF has been proven by videos in [1], Section Results> Games for motor and cognitive rehabilitation> Imitation. The overall latency is less than 100 milliseconds that is suggested for developers.

4. Conclusion

The proposed wireless Kinect-NAO framework for teleoperation has been implemented and tested with children and adults. Its feasibility and usability have been proven by real experiments with required accuracy, view invariance, whole-body consistency and response in real time. Structured expressions of natural language as linguistic IF-THEN rules, flexible fuzzy sets and easy for implementation wireless connection to NAO middleware allow the framework to be tried by others without detailed knowledge for video processing, inverse Kinematic task in Robotics and fuzzy logic. The proposed approach for NAO teleoperation using Kinect sensor is general enough to be applied to other humanoid robots.

Acknowledgments

This research is inspired by the EU COST Action No. TD1309 „Play for Children with Disabilities (LUDI)" and supported by the EEA grants, BG09 N D03-90/27.05.2015.

References

1. http://iser.bas.bg/METEMSS/en/.
2. http://www.microsoft.com/en-us/kinectforwindows.
3. Aldebaran NAO Humanoid Robot. URL: https://www.ald.softbankrobotics.com/en/cool-robots/nao
4. Skeletal Joint Smoothing White Paper. URL: http://msdn.microsoft.com/en-us/library/jj131429.
5. Mukherjee, S. et all. Inverse Kinematics of a NAO Humanoid Robot Using Kinect to Track and Imitate Human Motion. Int. Conf. RACE 2015, 18-20 February 2015, Chennai, India, 1-7.
6. Wenbai, C. et all. Human's Gesture Recognition and Imitation Based on Robot NAO. – *Signal Processing, Image Processing and Pattern Recognition*, 8, 2015, No. 12, 259-270.
7. Rodriguez, I. et al. Humanizing NAO Robot Teleoperation Using ROS. 14th IEEE-RAS Int. Conf. on Humanoid Robots, 18-20 November 2014, Madrid, Spain, 179-186.
8. Alibeigi, M., S. RabieeMajid, N. Ahmadabad. Inverse Kinematics Based Human Mimicking System Using Skeletal Tracking Technology. – *Journal of Intelligent & Robotic Systems*, 85, 2017, No. 1, 27–45.
9. Koenemann, J, F. Burget and M. Bennewitz. Real-time Imitation of Human Whole-body Motions by Humanoids. IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, 2806-2812.
10. Kondo, Y., S. Yamamoto and Y. Takahashi. Real-time Posture Imitation of Biped Humanoid Robot Based on Particle Filter with Simple Joint Control for Standing Stabilization. 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), Sapporo, 2016, 130-135.
11. Shuo, J., D. Chengkai, L. Yang, C. L. W. Charlie. Motion Imitation Based on Sparsely Sampled Correspondence. Technical Report on arVix.org.
12. Nierhoff, T. Real-time Robotic Motion Control and Adaptation in Constrained Environments. PhD Thesis, 2015 http://mediatum.ub.tum.de/doc/1238415/1238415.pdf.
13. Aldebaran Choregraphe. http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html.
14. Aldebaran Webots Overview. http://doc.aldebaran.com/1-14/software/webots/webots_index.html.
15. Takagi, T., M. Sugeno. Fuzzy Identification of Systems and its Applications to Modelling and Control. – *IEEE Trans. Sys. Man Cyber*., 1985, 15:116–132.
16. Aldebaran Documentation, NAOqi Motion, Whole Body Control. http://doc.aldebaran.com/2-1/naoqi/motion/control-wholebody.html.
17. Ferreau, H. J., H. G. Bock and M. Diehl. An Online Active Set Strategy to Overcome the Limitation of Explicit MPC. – *IEEE International Journal of Robust and Nonlinear Control*, 2008, 816-830.
18. Gamal, T. NAO.NET. http://www.codeproject.com/Tips/1002530/NAO-NET-Python-programs-to-NET.

Appendix 1

```
//L Knee Pitch RULES
TSFS_LKP = new TSFSystem(); //initialize a new Tagaki Sugeno Fuzzy System type

TSFS_LKP.addRule(new TSFRule(-4.6, new string[] { new TrapezMembershipFunction(120, 168, 175, 180); })); //L knee up
TSFS_LKP.addRule(new TSFRule(55, new string[] { new TrapezMembershipFunction(90, 115, 120, 160)})); //half
TSFS_LKP.addRule(new TSFRule(120, new string[] { new TrapezMembershipFunction(50, 50, 90, 115)}));// L knee down

double NAO_LKP = TSFS_LKP.CrispOutput(new FuzzySet[] { new FuzzySet(FuzzyVariables[], KinectKneePitchL)});

// interdependencies among leg joints
double NAO_LAP = -0.5828*NAO_LKP + 3.1229;
double NAO_LHP = -0.3965 * NAO_LKP + 5.6012;
double NAO_LAR = 0.0317*NAO_LKP - 6.3346;
double NAO_LHYP = -0.0436*NAO_LKP - 10.443;
double NAO_LHR = -0.0381*NAO_LKP + 6.9073;
```

Appendix 2

Algorithm for deriving whole-body relations in Choregraphe

1. In a new Project drag from the box library the desired pose and connect to the main onStart;
2. Drag from Templates a Timeline and connect the timeline box to the first box onStopped output. Right click the box and edit to the desired pose;
3. Double click the timeline box and drag the desired pose in it and connect to start
4. Go to the key frame bar and insert new key frame at the desired time on the motion section. By right click store the current joints positions in that key frame with options for the whole body, head, arms or legs;
5. Go to the play controls (the green button) and start for a while the NAO motion, then stop it (by the red control button). Make the next key frame and store the current joint values;
6. Repeat the Step 5 until reaching the end of the pose. The number of recorded key frames depends on the desired precision;
7. To analyse curves use the "pen" button on the motion section to go to the Choregraphe Timeline Editor where by "sinusoid" button to see the participated joints 2D graphics over time.

Appendix 3

```
// Python script for motion mapping of Kinect feature angles to NAO joint angles

def StiffnessOn(proxy):
  pNames = "Body"
  # the collection of all joints
  pStiffnessLists = 1.0
  pTimeLists = 1.0
  proxy.stiffnessInterpolation(pNames,
  pStiffnessLists, pTimeLists)

def main(robotIP,Angle1,…,Angle36):

  # Init proxies.
  try:
    motionProxy = ALProxy("ALMotion", robotIP, 9559)
  except Exception, e:
    print "Could not create proxy to ALMotion"
    print "Error was: ", e

    # Set NAO in Stiffness On
  StiffnessOn(motionProxy)

  # Activate Whole Body Balancer
  isEnabled  = True
  motionProxy.wbEnable(isEnabled)

  # Legs are constrained in a plane
  stateName  = "Fixed"
  supportLeg = "Legs"
  motionProxy.wbFootState(stateName, supportLeg)

  # Constraint Balance Motion
  isEnable   = True
  supportLeg = "Legs"
  motionProxy.wbEnableBalanceConstraint(isEnable, supportLeg)

 # KneePitch angleInterpolation
  # Without Whole Body balancer, foot will fall down
 names = ["RAnklePitch","LAnklePitch","RKneePitch","LKneePitch","RHipYawPitch","LHipYawPitch","RHipRoll","LHipRoll","RHipPitch","LHipPitch","RAnkleRoll","LAnkleRoll"]
  angleLists = [ [Angle1, … Angle36] ]
  timeLists  = [ [3.0, 6.0, 9.0], ..,[3.0, 6.0, 9.0]]
  isAbsolute = True
  motionProxy.angleInterpolation(names, angleLists, timeLists, isAbsolute)

  # Deactivate Whole Body Balancer
  isEnabled  = False
  motionProxy.wbEnable(isEnabled)
```

```
if __name__ == "__main__":

    if len(sys.argv) > 1:
            Angle1 = float(sys.argv[1])
            ...
            Angle36 = float(sys.argv[36])
            robotIP = sys.argv[37]
main(robotIP, Angle1,...,Angle36)
```

Manuscript received on 28.02.2017

Anna Lekova, *PhD, Professor and Head of the Interactive Robotics and Control Systems Department at Institute of Robotics, Bulgarian Academy of Sciences. She received her MSc in Computer Science (1988) and her PhD in CAD/CAE/CAM from the Technical University – Sofia (1995). Her research interests are in fuzzy-logic for pervasive human-robot interactions, gesture recognition, vision-based motion sensing devices, image processing and pattern recognition, telerobotics. She was coordinator of the EEA Grants project METEMSS (2015-2016) with partners from the South-West University of Blagoevgrad, Bulgaria and University of Stavanger, Norway. In the frame of the project this paper was developed.*

Contacts:
Institute of Robotics, BAS
Bl. 2, Acad. G. Bonchev St.
1113 Sofia, Bulgaria
tel: +359 887435648
e-mail: alekova.iser@gmail.com

Aleksandar Krastev, *PhD, Assistant Professor in the Interactive Robotics and Control Systems Department at Institute of Robotics, Bulgarian Academy of Sciences. He receives his MSc in 2004 in Town Ecology, from the Department of Ecology of the University of Forestry – Sofia and his PhD in 2013 from the Institute of Systems Engineering and Robotics – BAS. His recent research interests are in robotic systems and interactive human-robot interfaces.*

Contacts:
Institute of Robotics, BAS
Bl. 2, Acad. G. Bonchev St.
1113 Sofia, Bulgaria
tel: +359 2 870 3361
e-mail: aikrastev.iser.bas@gmail.com

Ivan Chavdarov, *PhD, Associate Professor. He received his MSc in Mechanical Engineering Science (1991) and his PhD in Robotics from the Technical University – Sofia (2004). Institute of Systems Engineering and Robotics (ISER) – BAS. His research interests are in Mechatronics, robotic systems, 3D fast prototyping.*

Contacts:
Institute of Robotics, BAS
Bl. 1, Acad. G. Bonchev St.
1113 Sofia, Bulgaria
tel: +359 9792422
e-mail: ivan_chavdarov@dir.bg