Towards learning to balance with iCub

Jan Geukes[†]
TU Darmstadt

Moritz Nakatenus[†]
TU Darmstadt

Roberto Calandra
TU Darmstadt

Abstract

Balancing is important for many robotic applications, especially for humanoid robots like iCub. For improving the balancing behaviour of the iCub the skin can be used to make more precise measurements, because the sensors on the joints doesn't get exactly the orientation of the applied forces. Moreover the skin can be used to add an additional reflex behaviour. To learn such a behaviour with reinforcement learning we need to do tests in a simulator. In the simulator provided for the iCub, called Gazebo, is still no skin provided. This report deals with how we did include the skin in the simulator.

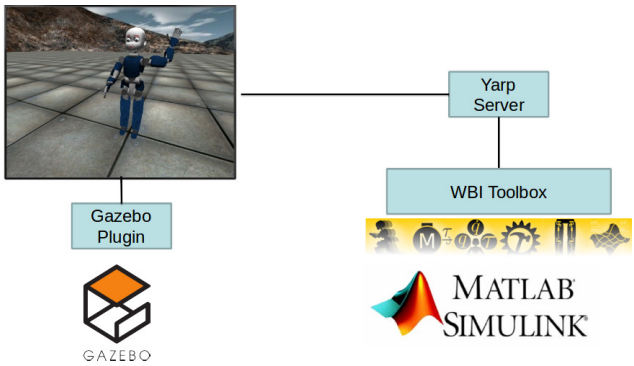


Figure 1: The composition of our framework

1 Introduction

Robot balancing is an important issue which is essential for several kinds of robots which are interacting with their environment, especially in new unknown scenarios. So the robot gets more stable and the probability to insure itself or humans surrounding is smaller. Solving such a problem is a challenge which is an important and ambitious topic in the current research. ICub is a project founded by the Italian Institute of Technology [1]. As base capabilities it can crawl, has image processing, object recognition and manipulation skills. The iCub looks like a four year old child and has a complex technology: It has 53 degrees of freedom, own artificial sensor skin with over six thousand force and torque sensors, a dual core machine within its head and much more. The skin is placed on hands, arms, legs and the torso. For cognition of its visual and acoustics environment iCub has two cameras and microphones[2]. To control iCub has its own software 'Yarp' [3] and to extend the robot there is more software avalaible which is open source. Yarp stands for 'yet another robot platform' and is constructed for the communication between modules of the robot. The 'whole body interface toolbox' uses Yarp and can get thereby sensor data from the robot which is important for creating the controllers, change torques of the robot and make the results visible on the Gazebo simulator. The WBI Toolbox is implemented in Matlab/Simulink and has some blocks for building the controllers. The WBI Toolbox was created due to the CoDyCo Project which is about cognitive understanding and motion interaction with multiple contacts[4]. For simulation of the robot we are using Gazebo 5. To run the balancing controllers on the Gazebo model of the iCub the WBI-Toolbox communicates with Gazebo via Yarp. We had problems with the framework because there are many bugs and inconsistencies (e.g. the gains depend on the hardware). Because of that we could not continue with our goal to improve the balancing controller. This is why we decided to just include the skin of the iCub in Gazebo. In this report we explain how we compute the sensor position data and how we used this data to add the skin to the iCub Gazebo model. Futhermore we report how we have written an improvement of the Yarp driver to get the skin data from the model and mention how we coded a plugin to send the skin data from the iCub model to Yarp.

† These authors equally contributed to this work.

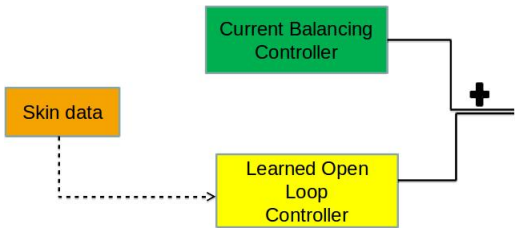


Figure 2: Idea to improve the balancing controller

2 Balancing with the iCub

The current version of the iCub is already able to balance. The approach controls the CoM of the robot and tries to estimate the force and torques which are applied on the robot. The Codyco project showed good results (see [5]) and the WBI Toolbox provides controllers for this kind of balancing. But these balancing controllers are rather slow in the way they react to pushes. One major problem is how to compute the external forces and its direction from the sensor input. Because the current controller wants to compensate the external forces. It is critical to compute the forces as exact as possible. And like Del Prete et al. explain it is not so easy to get the right position for the single sensor parts [6]. The main idea how to improve these controllers is to use just the skin data without explicit computing the force. So at the beginning we wanted to combine the current controller with an open loop controller. This new controller uses the skin data from the iCub and adds a certain amount to the current controller. In a way simulate some kind of reflex.

2.1 Technical Issues

In our work we suffered from the unstable WBI Toolbox because the results that we saw on the Gazebo simulator are depending on the hardware (the controllers are optimized to run on the real robot). Moreover, the controllers got a lot of bugs. For this reason the gains must be specified on the current system so our robot first falls on the ground or flies around triggered by some little impact. Trying to fix this costs us much time and we only managed that the robot does not fly around the simulator and stands on its position but the movements are still looking strange. So we decided to focus on how to implement the skin of the iCub in gazebo, because there is basically no skin simulation provided right now.

3 Skin of the iCub

The skin of the iCub is composed of thousands of small tactile elements, called taxels. These taxels are grouped into triangles like you can see in Figure 3. Up

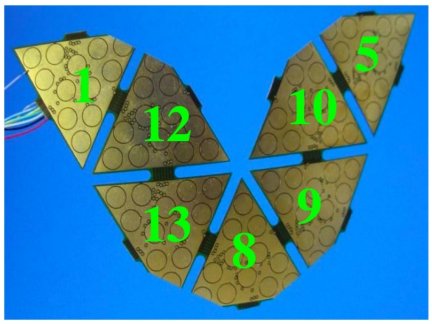


Figure 3: Structure of the skin from [7]

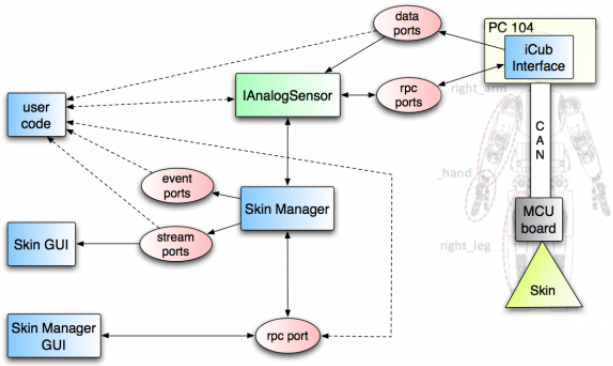


Figure 4: Skin data process from [7]

to 12 such triangles shape a patch. There are skin patches for most body parts of the iCub.

The ouput of the skin data is a byte per taxel. So it can have values between 0 and 255. In the case of raw skin data information 255 means high pressured and low values signals low impact on the skin sensor [8]. The user can get this information via Yarp ports. The output is a vector with up to 192 values per patch. A list of the current output ports can be found in [7]. The real iCub also needs to deal with thermal drift compensation and different thresholds for each taxel. Therefore there are some ports which expose filtered skin data for each patch, these are called compensated skin data. Furthermore it is important to know that not all values of such an output vector are used for measuring the pressure, some are used to transport configuration information or some are just zero. How all the values should be interpreted can be found at [7]. The complete processing step for the skin data of the iCub is shown in Figure 4.

3.1 Skin Gui

In the original iCub Simulator you can use an element called SkinGui. It displays a 2D representation of the skin. After starting the program you get a new Yarp port. The next step is to connect this port with a Yarp

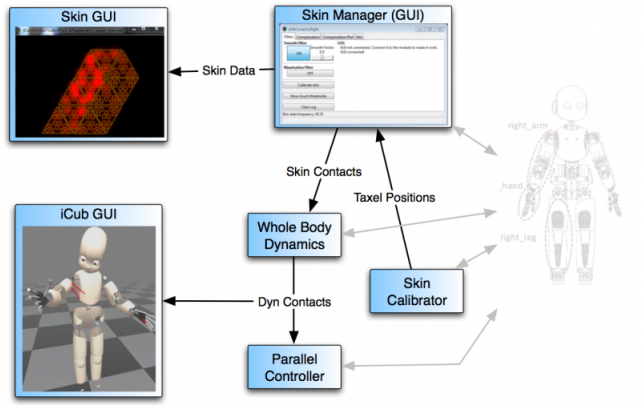


Figure 5: Skin Gui from [9]

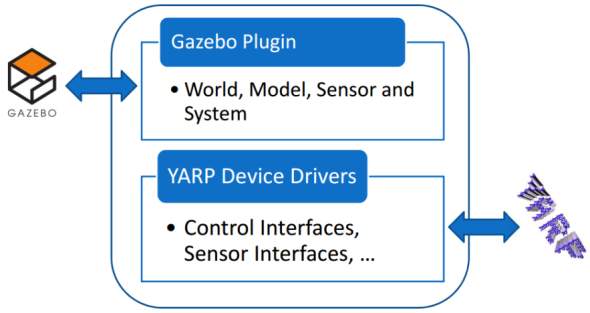


Figure 6: Model Gazebo Yarp Plugin from [14]

port which contains the skin data. In order to read and interpret the skin data in the right way, SkinGui uses some configuration files and templates ([9]). SkinGui can use either raw skin data or compensated skin data.

4 Gazebo Yarp Plugin

As described above Yarp is the main control unit for the iCub. In order to use the Gazebo Simulator it is necessary to define a connection between these two elements. Hodicky et al, descried a basic framework how to achieve this connection [10]. Additional to define the architecture of this gazebo-yarp-plugins they also implement a few plugins. For example to use force and torque sensors within the simulation [11]. Our task was to understand this architecture and then implement such a gazebo-yarp-plugin for our skin simulation. The goal was to do it in such a way, that we are as close as possible to the real iCub output.

4.1 Gazebo Yarp Plugins Structure

In general three steps are needed in order to get the Gazebo sensor data working together with Yarp. First the current model of your robot has to be modified. Second you have to write a Gazebo Plugin class and last a Yarp device driver has to be implemented. There are several different sensors which are part of Gazebo. The behaviour of these sensors is defined within the Gazebo plugin class. The connection between model, in our case a SDF file and the Gazebo plugin and the Yarp device is organised by the 'plugin' tag as well as the 'yarpConfigurationFile' tag [12]. The third part a Yarp device is the implementation of an interface which connects the sensors in the Gazebo simulation with the Yarp control unit and provides the possibility to access the sensor data with a Yarp port [13]. The main idea is to use Gazebo sensors and its plugins, then Yarp device drivers create a connection to Yarp. The final output of such a gazebo-yarp-plugin should be a port in the Yarp network.

4.2 Gazebo Yarp Plugins Implementation

In our implementation we use the contact sensors which are given in Gazebo. These sensors provide information for each contact. You can get the positions of the contacts. Furthermore information about the two involved collision objects are provided as well as applied forces during a collision [15]. Moreover an URI pointing to a configuration file has to be added between the YarpConfigurationFile tags. This file consists of several key value pairs which are needed to initialize the the gazebo-yarp-plugin.

After modifying our model in the next step a Gazebo plugin for the contact sensor has to be implemented as a C++ class. Furthermore, we need a Yarp device driver class which is also a C++ class. Both classes are quite close coupled. First the Gazebo sensor has to be loaded and registered. Then a connection to the Yarp framework is created according to the configuration file which is defined in the Gazebo model. The second important part is the behaviour of the gazebo-yarp-plugin on update steps. The frequency of the sensor is set in the configuration file. If the function 'onUpdate' is called, we have to calculate the skin value for every taxel of our current skin patch. These values between 0 and 255 are written to a vector and then posted to the Yarp port (name of the port is also defined in the configuration file).

The Gazebo collision boxes can measure the forces during a contact. This force value is used to compute the strength of the contact. In the next step a mapping between these force values and the desired output values from 0 up to 255 is needed. For simulation tests we use an incomplete beta function (1) with $x = 0.8$

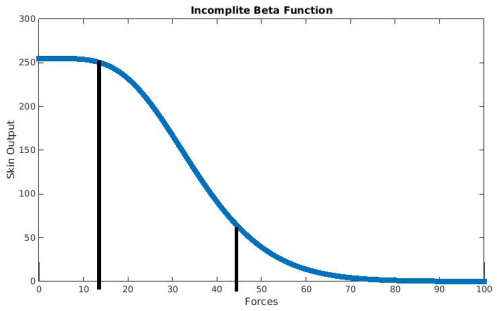


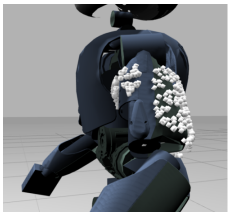
Figure 7: Incomplete beta function

and $p = 0 - 100$ and $q = 9$.

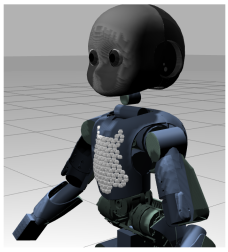
$$B_x(p,q) = \frac{1}{beta(p,q)} * \int_0^x t^{p-1}(1-t)^{q-1}\,dt \qquad (1)$$

The beta function returns values between 0 and 1 so we can adapt this function to get integer values between 0 and 255. The function is part of the C++ boost library.

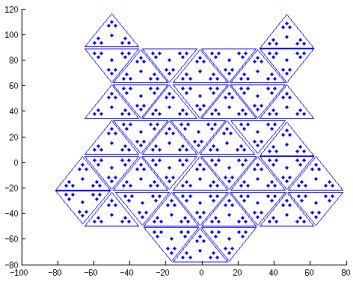
According to the documentation for the real iCub the default value is 235 (no pressure on the skin), therefore the output vector is initialized with 235. Then a list with all current contacts can be used to determine which parts of the skin are involved in a collision. Because every taxel has its unique identifier you can conclude its position in the vector.



(a) Collision boxes from provided data



(b) Manually generated skin sensor data



(c) Torso data generated with matlab

Figure 8: Skin sensors as collision boxes

5 Implement the skin in Gazebo

As there is no provided framework for simulating the skin of the iCub in Gazebo, we had to implement our own. The first idea was to use 'SkinSim' but this was unrealistic to achieve in the short time. Moreover SkinSim needs much computation time because it not only emulates the force and torques on the skin but also the surface with some kind of spring-damper system[16]. So our method is to simulate the skin sensors with tiny collision boxes in Gazebo. To achieve this we have written a Java program which gets the positions of the sensors from our computed and from provided data (the provided data has some noise due to a calibration process), and generates collision boxes for the iCub sdf Gazebo file. The data was not always correct so we had to change it. To get the collision data from the boxes every box is associated with a sensor which sends its data to Yarp. For the torso and legs we had no sensor data provided so a script must be written in Matlab which gets the data from the 2D skin gui as positions and orientation of the texal triangles and with that data we compute the sensors of the torso and the sensors for the legs. As the skin gui is in 2D the data of the manually computed sensors are lying in a layer. So we must compute the bulge. For reaching this, our idea was to use two cosine functions, each one for vertical and horizontal bending. To compute the sdf files needed for the Gazebo simulator we had to transfer the generated data from Matlab to the robot definition file. To achieve this we have written a Java program which uses the xml open source library 'JDOM' for creating this files, because the sdf file is in a xml format. In the program we defined the collision boxes, their associated sensors and included our Gazebo plugin for sending the collision data to Yarp.

$$bending = a * \cos(\frac{\pi x}{width}) \qquad (2)$$

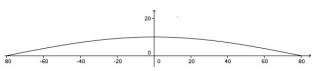


Figure 9: Cosine function for bending

6 Future work

The implemented skin model provides first possibilities to use the iCub skin in simulations. But compared with the real iCub some rather strong simplifications have been made. In the next steps the skin model can be improved. For example more physical properties of the skin could be included into the model. Depending on the application, a better estimate of the position of the tactile sensors on the robot is necessary. One way to do this is to use the calibration methode like Prete el suggested [6] or use CAD information. Additional tests for calibrating the mapping function of the force in simulation have to be done. Nevertheless these improvements can be done on top of the current gazebo-yarp-plugins. Then we can start with reinforcement learning in the simulator and learn some balancing controllers.

7 Conclusion

To reach the goal to improve the balancing of the iCub with the help of skin data we first have to implement a skin model in the simulator. In order to do so we created gazebo-yarp-plugins which insures the communication between Gazebo and Yarp. We have implemented a skin model in Gazebo, used a Yarp plugin for communications and put some logical into a Gazebo plugin. The artificial skin is an important feature of the iCub. With our model of the skin first applications like improving balancing are possible.

8 Appendix

8.1 How we installed our environment

For our development environment we used Ubuntu 14.04. Yarp is the basis software to interact with the robot, so we downloaded and installed it from GitHub[4]. After that we installed the latest version of Gazebo (Gazebo 5) [17] with the Gazebo Yarp plugins, so the Gazebo simulator gets data from Yarp to control the robot. For this plugin there must be defined some environment variables. The next step was to get the iCub model with its configuration files. With this files you can see the iCub model on the simulator by opening the icub.world file with gazebo in the related folder. Besides this its important too to install the iCub software itself. To do this the related dependencies must be installed with the package 'icub-common'. Note: Yarp and the iCub software must be installed from sources. If this is done we had to install the Codyco superbuild package which contains all necessary dependencies to install and the WBI Toolbox itself. In here there must be again defined some environment variables and dependencies. The last step was to setup Matlab/Simulink for the toolbox (again environment entries are necessary). To do this we had to include some paths in Matlab. Now we were able to work with the toolbox if we set the path from the controller which we want to use in Matlab and define the parameters before starting the controller:

```
robotName='icubGazeboSim'
localName='simulink'
Ts=0.01
ROBOT_DOF=25
```

8.2 Detailed description of our controller problems

As mentioned in 'Introduction' we had problems to control the robot with gains which are depending on the system hardware. To solve this problem we first tried to use different gains. We divided all gains by different factors like five or ten, varied the weights of the several gains, set them to zero and much more. As this didnt work we tested to set the output torques of the controller output by hand. So we made it to localize a division factor for the torques so the robot stood on the ground and made for estimated five seconds no strange movements (like flying around the simulator), but only if we let the feet be fixed. With additional adaptions the robot doesn't fly around but rests in a strange position. This result was very unsatisfying, so we tried to limit the joints, which had no valuable impact on the results. As new controllers were be uploaded we immediately tested them, but they had many bugs which the developers admitted in the GitHub issue that we opened. So none of the controllers are working currently.

References

[1] IIT. iCub, 2015. http://www.icub.org/, last visit: 09.02.2015 16:02 Uhr.

[2] IIT. iCub Bazaar, 2015. http://www.icub.org/bazaar.php/, last visit: 09.02.2015 16:04 Uhr.

[3] IIT. Welcome to Yarp, 2015. http://wiki.icub.org/yarpdoc/index.html/, last visit: 09.02.2015 16:07 Uhr.

[4] IIT. Whole Body Interface Toolbox (WBI-Toolbox v0.2) - A Simulink Wrapper for Whole Body Control, 2015. https://github.com/robotology-playground/WBI-Toolbox/, last visit: 09.02.2015 16:10 Uhr.

[5] Francesco Nori and Daniele Pucci. iCub balancing on one foot while interacting with humans, 2015. https://www.youtube.com/watch?v=VrPBSSQEr3A, last visit: 08.03.2015 16:29 Uhr.

[6] Andrea Del Prete and Simone Denei. Skin spatial calibration using force/torque measurements. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3694–3700. IEEE, 2011.

[7] IIT. Tactile sensors, 2015. http://wiki.icub.org/wiki/Tactile_sensors_(aka_Skin), last visit: 07.02.2015 16:34 Uhr.

[8] Andrea Del Prete. iCub Skin Tutorial, 2015. http://eris.liralab.it/images/6/6e/Skin_tutorial.pdf, last visit: 07.02.2015 16:34 Uhr.

[9] IIT. iCubSkinGui, 2015. http://wiki.icub.org/brain/group__icub__iCubSkinGui.html, last visit: 07.02.2015 16:34 Uhr.

[10] Jan Hodicky. *Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May 5-6, 2014, Revised Selected Papers*, volume 8906. Springer, 2014.

[11] Daniele E. Domenichelli. gazebo-yarp-plugins structure, 2015. https://github.com/robotology/gazebo-yarp-plugins/wiki/Design, last visit: 07.02.2015 16:34 Uhr.

[12] Silvio Traversaro. Embed gazebo yarp plugins in an SDF model, 2015. https://github.com/robotology/gazebo-yarp-plugins/wiki/Embed-gazebo-yarp-plugins-in-an-SDF-model, last visit: 07.02.2015 16:34 Uhr.

[13] IIT. Getting Started with YARP Devices, 2015. http://wiki.icub.org/yarpdoc/note_devices.html, last visit: 07.02.2015 16:34 Uhr.

[14] Alessio Rocchi. YARP Plugins for Gazebo Simulator:development and application on the iCub and COMAN robots, 2015. http://www.icub.org/other/files/rocchi-gazebo.pdf, last visit: 07.02.2015 16:34 Uhr.

[15] OSRF. Contact sensor, 2015. http://osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1sensors_1_1ContactSensor.html, last visit: 07.02.2015 16:34 Uhr.

[16] Kyle Shook Ahsan Habib, Isura Ranatunga and Dan O. Popa. SkinSim: A Simulation Environment for Multimodal Robot Skin, 2015. http://www.mae.cuhk.edu.hk/~cmdl/activity/201406_ICRA%20Soft%20robot%20workshop/SkinSim_uta.pdf/, last visit: 09.02.2015 16:12 Uhr.

[17] OSRF. Gazebo Simulator, 2015. http://gazebosim.org//, last visit: 09.02.2015 16:29 Uhr.