The 10th Workshop on Humanoid Soccer Robots
15th IEEE-RAS International Conference on Humanoid Robots, Seoul, Korea, November 2015

Rhoban Hardware and Software Open Source Contributions for RoboCup Humanoids

Quentin Rouxel, Grégoire Passault, Ludovic Hofer, Steve N'Guyen, Olivier Ly

***Abstract*— Three newly released hardware and software open source contributions are presented. The first one is a new mechanical and electronics design for low cost stain gauge based foot pressure sensors. The second is a lightweight user interface library dedicated to robotics applications and allowing for an "on the fly" interaction with an on board program. The third one is an implementation of an open loop walk generator for humanoid robots based on splines and inverse kinematics.**

I. Introduction

The Robocup soccer Humanoid league is a highly competitive challenge with a regular increasing difficulty. The 2015 update of the rules included an artificial grass field, a white ball and white goals. These new problems make it quite difficult for new teams to enter the competition from scratch. Moreover, the willingness to create collaborative teams emphasizes the need for open platforms. In this context, this work presents three open source contributions from the Rhoban team (Humanoid kid size) to the Robocup community. These three projects are presented in the following sections and are entirely independent on each other.

II. Low Cost Foot Pressure

The locomotion is one of the biggest challenge of humanoid robotics, and especially with the RoboCup humanoid league constraints that only allow sensors and actuators that are comparable with humans.

Typically, for walking, shooting or balancing, one can ask:

- Where is the center of pressure of the robot?
- Which foot is actually touching the ground and supporting the robot?
- Is the robot standing on the ground, or falling?

All these questions can be estimated using IMU (accelerometer, gyroscope), the position of the motors and a complete model of the robot. However, sensing the force applied by the robot on the floor, and thus on the tips of the feet, brings new information.

Human-size recent robots, such as TORO [1] feature 6DOF industrial force sensors in each foot, which are very high quality but also very expansive measurement instruments. This is why we decided to produce a custom design that would be both affordable and suitable for small to mid-size humanoids [2].

The authors are with the *Rhoban team*, LaBRI, University of Bordeaux, France. Emails: {quentin.rouxel, gregoire.passault, ludovic.hofer, steve.nguyen, olivier.ly}@labri.fr

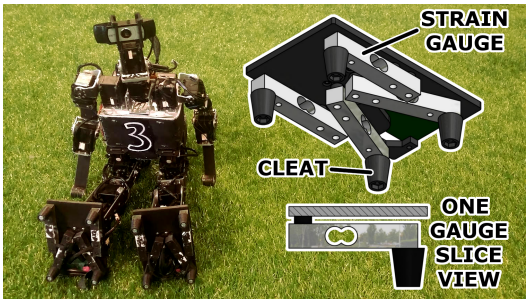

Fig. 1: On the left side, Sigmaban Kid-Size Humanoid Robot. On the right side, low cost foot pressure sensors made of 4 strain gauges and measuring the weight on each cleat. This mechanical assembly is robust and well adapted to soft artificial grass improving the stability of the robot

A. Overview

We propose a low-cost force sensing foot that is suitable for small humanoid robots that can be found here:

`https://github.com/Rhoban/ForceFoot`

This contribution is composed of mechanical, electronics and firmware designs.

B. Mechanical design

The well-known Nao robot [3] features FSR sensors that allows to measure the force below the feet. However, industrial standard 6DOF sensors used in human-sized humanoids are based on highly precise strain gauges.

A consideration for this new design was the transition from flat carpet to a soft ground in the humanoid league, which led to the integration of cleats below the feet, switching from a plane contact to contact points. The cleats were adopted by almost all teams, because it distributes ground contact forces on small surfaces, making easier to cross the turf and increasing the stiffness of the contact.

We used strain gauges based load cells, which are mechanical parts that are directly integrated to the structure of the foot and which infinitesimal deformations are measured using small resistors network depicted on Fig. 1. The resulting foot is about 85x130x40mm and weighs approximately 200g.

The deformations of the mechanical part are such that expansion and compression occur on each side when stressed as simulated in Fig. 2.

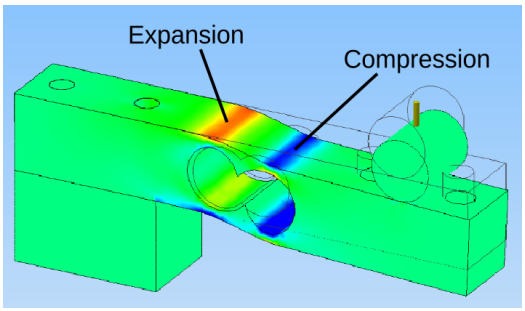

Fig. 2: Exaggerated deformations of a mechanical standard load cell with a force applied on it using finite elements method. We can see that both compression and expansion occur on the same side. This is where the strain gauge resistors are installed.

C. Electronics

On each of these parts (four per foot), four strain gauge resistors are installed, forming a full Wheatstone bridge shown in Fig. 3.

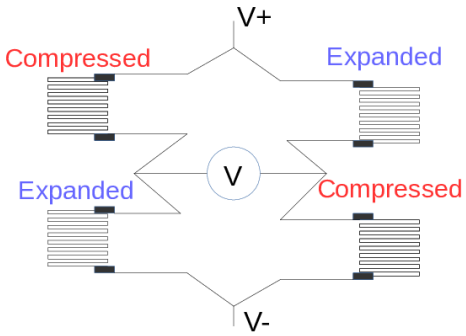

Fig. 3: Four strain gauges are mounted as a full Wheatstone bridge, two per side of the mechanical load cell.

The signal measured is the differential voltage between the two points of the Fig. 3, since this is a low voltage, it has to be amplified.

We used the HX711[1] chip, featuring an amplification of 128 and a 24 bit $\Sigma\Delta$ ADC and providing a serial interface. We designed a custom board including this chip and a MCU that is integrated on the foot.

D. Using the measures

Lots of humanoid robots of the RoboCup use Dynamixel servos, and all of them use daisy chain serial bus. This is why we designed our board with a Dynamixel compatibility.

Before using this measures, one should calibrate it. A good method to do this is by putting several known masses on the gauge and processing a linear regression. This allows to compute both the zero offset and the linear coefficient.

[1]The HX711 is a mainstream low-cost integrated circuit designed for weigh scales applications

Here are some interesting measures one can then get:

- **The total weight**: this will inform the system if the robot is holding on its feet or has fallen. Combined with the IMU, one can know if the robot is handled by an human. This can also be used to detect if everything is correct when performing system check or if any external force is applied.
- **The foot ratio**: this is the distribution of the weight between the two feet. This can be useful when walking because it depicts what is the current support phase.

Total weight and lateral center of pressure measures over a walk cycle are illustrated on Fig. 5.

E. Estimating the center of pressure

Note here that the actual center of pressure is partially estimated since only the force projected on vertical Z axis is measured. All horizontal ground reaction forces are not considered.

We can model each load cell with a spring of rigidity K_i (newton/meter) and a "plate", which correspond to the foot.

The pressure of the robot is then represented by a single force P on this plate (see Fig. 4).

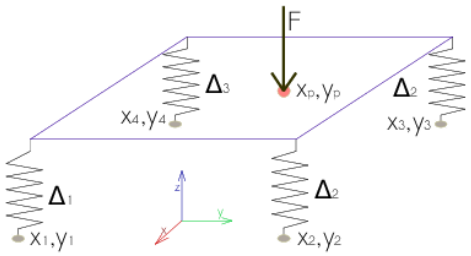

Fig. 4: Model of one foot using four strain gauges.

The force that is applied on the ith cleat at the ground contact is:

$$F_i = K_i \Delta_i = K_i C_i R_i$$

Where Δ_i (meter) is the deformation of the gauge, C_i (meter/volt) is the linear relation between the length of the gauge and the voltage R_i measured from the Wheatstone bridge (the "raw" measure). Note that, in general, one will likely calibrate simultaneously K_i and C_i, getting an unique gain per gauge (which is $K_i C_i$, newton/volt).

Statically, the sum of the forces and moments is zero, i.e.:

$$\sum_{i=1}^{i=4} F_i + P = 0 \qquad \begin{cases} \sum_i x_i F_i + x_p P = 0 \\ \sum_i y_i F_i + y_p P = 0 \end{cases}$$

Thus:

$$\begin{cases} P = -\sum_i F_i \\ x_p = -\frac{\sum_i x_i F_i}{P} \\ y_p = -\frac{\sum_i y_i F_i}{P} \end{cases}$$

Note that we can use this equation to estimate either the center of pressure for one specific foot or for the whole robot. However a kinematic model of the robot is needed in order to know the relative positions of each foot.

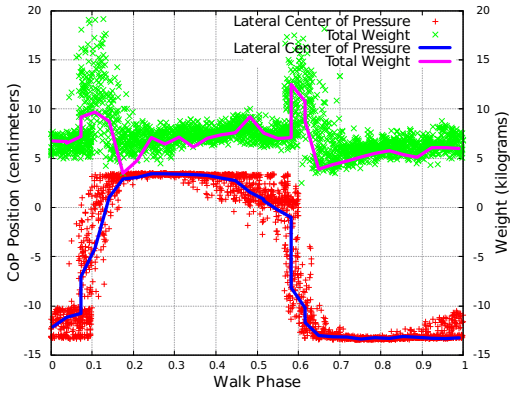

Fig. 5: The measured total weight of both feet and lateral (Y) estimated position of the center of pressure of the robot are plotted against the walk phase (between 0 and 1) while the robot is walking in place. Green and Red points are measured over several periodic walk cycles. Blue and purple pick out one typical trajectory. At $phase = 0.1$ and 0.6, the support foot exchange and the ground collision are clearly visible.

III. Rhoban Input Output Library

The robotics field involves both theoretical and experimental knowledge. During experiments, a complex piece of software is controlling an advanced piece of hardware prone to fast and unstable dynamics as in the example of humanoid robots. Since an established theoretical framework for perception and motor control of biped robots is far from being achieved, current implementation architecture are still highly prototypes.

Field experiments require a lot of software debugging, system monitoring and parameters tuning while dealing with the physical robot. In practice, an efficient robot user interface is often necessary, especially in the context of the RoboCup soccer competition where a fast parameter update can be a game changer.

A. Decentralized and Monolithic Architecture

The *Robot Operating System (ROS)* is one of the most famous robotics frameworks [4] providing a large set of tools handling software organization, library sharing, behavior monitoring, configuration and more. The core architecture is highly *decentralized* and modules are communicating using network packet messages.

On the other hand, the opposite architecture is *monolithic*. The whole system lies in one process and communication is achieved by direct memory access and function calls. This simple architecture is widely used in RoboCup Humanoid community with some notable exceptions ; the NimbRo team [5] for instance.

In the RoboCup context, we often need software modules to be able to get information from almost every other modules. In this case, a decentralized architecture would require a lot of communication channels (almost a complete graph of all the modules). Thus, for optimization purpose one could prefer a monolithic architecture.

This contribution aims to build a lightweight and easy to use library in order to achieve the same generic usability that ROS but on monolithic architectures. Moreover, this library also comes with a command-line tool for easy user interaction.

B. RhIO Overview

We propose the open source library *RhIO (Rhoban Input Output Library)* targeting existing monolithic projects available at:

`https://github.com/Rhoban/RhIO`

This project is addressing many software engineering issues encountered during our four entries to the RoboCup Kid-Size Humanoid. RhIO is a lightweight C++ library that is linked against the robot application in order to interact with the program on-the-fly, through its integrated server.

A quick summary of RhIO features [2] is listed below and developed in following subsections:

- Lightweight and low performance overhead
- Simple integration with existing project
- Very few code to write to monitor/control a new variable
- Real time variables monitoring and logging
- Real time parameters update
- Unix-like folder tree based organization
- Robot Configuration management
- Terminal shell and command line interface
- Client shell allows for real time plotting with Gnuplot
- Based on ZeroMQ network library which allows extensions to any clients (graphical or not) [3]

RhIO has been successfully tested and used during RoboCup 2015 in China and is already integrated within our agricultural autonomous robots projects [6].

Based on our RoboCup experience, such a system could be of great help when several people are collaborating together on different parts of the robot's architecture. RhIO is also an attempt to propose a generic user interface to robots. Without knowing how a module is internally structured, one can still easily debug and tweak parameters through the interface. Such standard user interfaces may become interesting when RoboCup soccer teams will increase in number and when heterogeneous robots will

[2] An example video can be found at: `https://youtu.be/MOizgXYENLc`

[3] See all supported language at `http://zeromq.org/bindings:_start`

play in the same team.

The RhIO library is divided into two major components as shows in Fig. 8. On the robot's side, the server is running the monolithic application. On user's side, one or many clients are connected to the server through TCP protocol (based on standard ZeroMQ[4] network library) for user interaction.

C. Server Side

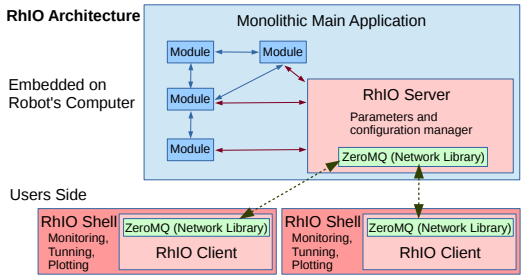

Fig. 6: RhIO Client-Server architecture

```
RhIO Root:
-vision/
-pressure/
-sensors/
-teamplay/
-moves/
    -approach/
    -robocup/
    -standup/
    -walk/
        footYOffset          float 0.025
        freq                 float 1.7
        supportPhaseRatio    float 0
        trunkRoll            float 0
        trunkYOffset         float 0
        trunkZOffset         float 0.02
        [...]
    [...]
-servos/
    -head_pitch/
        readingErrors    int   18
        angle            float 69.7852
        speed            float 0
        temperature      float 36
        torque           float 0.966764
        voltage          float 16.3
        [...]
    -head_yaw/
    -left_ankle_pitch/
    -left_ankle_roll/
    -left_elbow/
```

Fig. 7: Example of RhIO root node showing our internal software structure at RoboCup 2015 (Humanoid Kid-Size League). Our main modules are displayed at root level and some sub nodes show how values can be used to monitor motor state and configure the walk engine

RhIO allows for the creation of 3 kind of global objects on the server side. These objects can then be accessed everywhere within the robot's application:

- *Values* (of 4 basic types[5]) used for monitoring or as configuration parameters. Values have meta parameters such as their name, an optional comment, optional minimum and maximum range along with a flag indicating if the value is temporary or if it will to be persisted on the disk when the configuration is saved.
- *Functions* used by the client to call custom functions on the robot side, with optional arguments. They are used to trigger special actions or display monitoring report from the robot.
- Output stream allows for displaying textual information on a specific channel. Its primary use is for debugging and acts as a remote logging output.

Each of the previous object is attached to a Node belonging to a virtual hierarchy tree. In practice, each root node represent a system module (as motion, vision, sensors, ...). A tree example is partially displayed at Fig. 7.
It is then possible to easily interact with this representation with a command line tools mimicking a UNIX shell.

More complete documentation is available on the project webpage [6].

D. Client Side

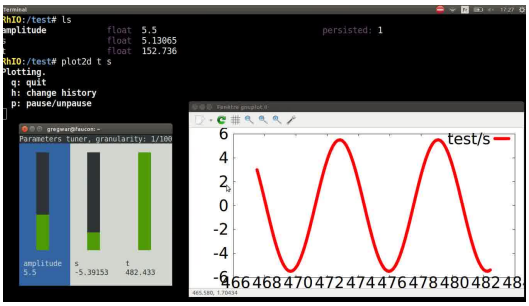

Fig. 8: RhIO client interface showing the command line shell, the command line sliders (parameters tuning) and the Gnuplot real time plotting.

For now, the only client implemented is the command line Shell shown in Fig. 8. The navigation inside node hierarchy is implemented in the same way as folders on an Unix system. Nodes are represented as folders and values (and streams) are represented as files. Some useful commands similar to the *bash* environment are implemented.

The Shell is providing the following features:

- Network connection through TCP protocol and ZeroMQ library
- Multiple clients can be connected at the same time
- Terminal-like environment
- Command line sliders (*NCurses*) used to tune multiple parameters
- A Gnuplot binding allows to monitor values in real time
- Joypads (on Linux) can be binded to some parameters. For example, in order to easily control the robot's walk directly through the RhIO walk parameters.

[4] http://zeromq.org/

[5] Types are either boolean, integer, floating number or string

[6] See API documentation at: https://github.com/Rhoban/RhIO/blob/master/Docs/api.md

- The client library is implemented in C++. Writing a new client in the same language would be quick and easy.

E. Future Work

At the current state, the library lacks a support for an image type. This particular kind of communication would be very useful since monitoring vision processing is of primary importance. Both server side and proper client side tools are needed to efficiently monitor the vision module ; debugging tagged images does not need to be computed when not monitored. In addition, a graphical client (could be web-based) might be convenient.

IV. Open Loop Walk Engine

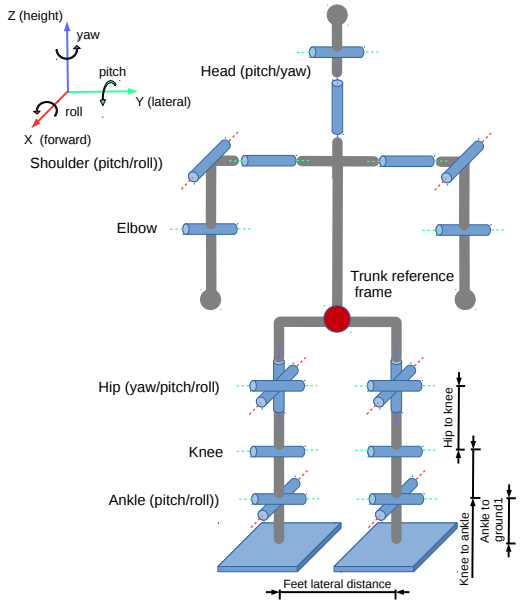

Fig. 9: Kinematic model and reference frame of the humanoid. 20 degrees of freedom, 6 per leg.

Biped walk is an extensively studied problem and various approaches have been tried to address it, from classic automatic control and the well known ZMP, bio-inspiration and CPG to machine learning.

The theoretical problem is quite difficult, involving complex kinematic structures, high dimensional control, dynamics motion and changing ground contact with collisions. However in practice, acceptable stable and omnidirectional walk engine can be achieved in a rather simple way on small humanoid robots along with good mechanical design and some expert manual parameters tuning. The walk engine that we propose here is sharing similar ideas that were first exposed by Behnke [7].

Periodic splines generate the main oscillatory pattern used to define feet and trunk cartesian parametrized trajectories. Motor target positions are then computed through "standard" inverse kinematics.

The library *IKWalk* that we propose is a C++ implementation of a typical fully open loop walk engine for humanoid robots. This implementation was used on artificial grass by the *Rhoban Football Club* during the soccer competition Robocup Kid-Size 2015 in China. The only software dependency needed is the *Eigen* linear algebra library [7].

`https://github.com/Rhoban/IKWalk`

A. Walk Engine Overview

The implemented engine is only considering the 12 degrees of freedom of the legs. Fig. 9 shows the "standard" small humanoid model design. Conveniently aligned 6 degrees of freedom per leg allow for a quite simple analytic inverse kinematics. The center of the knee axis, the intersection of the three rotation axes of the hip and the intersection of the two rotation axes of the ankle are all aligned on a vertical line.

This model is defined by 4 geometrical parameters: the distances between

- the center of hip axes and knee axis
- the knee axis and the center of ankle axes
- the center of ankle axes and the ground
- the two feet

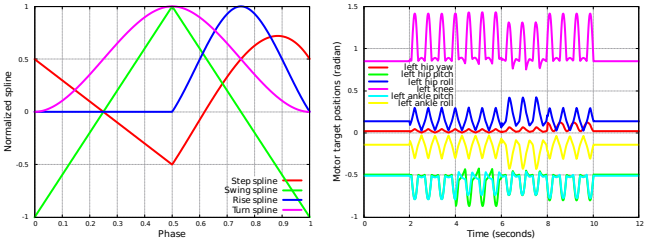

Fig. 10: (Left) The four periodic spline patterns using example parameters. The walk is only in single support phase.

Fig. 11: (Right) Computed target positions for the left leg with example parameters. Between $t = 0...2$ and $t = 10...12$, the walk is stopped. Between $t = 2...4$, the robot walks in place. Between $t = 4...6$, the walk is going forward. Between $t = 6...8$, the robot is walking on the left with lateral steps. And between $t = 8...10$, the robot is turning on its right.

The four splines patterns depicted in Fig. 10 are used to generate the whole motion in Cartesian space. The walk cycle begins at $phase = 0$ when the left foot just lands and starts going backward. At $phase = 0.5$, the right foot lands and the left foot takes off and goes forward when the double support phase length is set to zero.

- Step spline: Forward and lateral footstep displacements (X and Y axes) with respect to the trunk. Note that at $phase = 1$, the left foot is landing with non zero backward velocity.
- Rise spline: Foot motion in Z axis

[7]C++ header library available at: `http://eigen.tuxfamily.org/index.php?title=Main_Page`

- Turn spline: Foot yaw rotation used for robot's turning
- Swing spline: Lateral (Y) oscillation of the trunk with respect to the feet used to move the center of mass toward the supporting foot.

These normalized splines are then scaled and phase shifted to build the actual foot target positions (X,Y,Z) and orientations and fed to the inverse kinematics.

B. Engine Parameters

The walk engine uses the following main parameters [8]:

- Frequency of the complete walk cycle (two steps)
- Ratio between single support only to full double support
- Foot rise height (Z) during flying phase
- Trunk height (Z) from the ground
- Lateral (Y) distance offset between the two feet
- Amplitude of lateral (Y) swing oscillations
- Lateral swing phase shift with respect to the foot rise timing
- Forward (X) trunk offset with respect to the feet
- Trunk pitch (forward) orientation
- Two parameters allow to tune the trunk and center of mass trajectory as displayed in Fig. 12 [9]

In addition, these next "dynamic" parameters are updated during the walk in order to control the walking direction and achieve omnidirectional motion:

- Enabled ratio allows to stop and restart the oscillatory pattern smoothly
- Length of forward (X) footstep
- Length of lateral (Y) footstep
- Yaw angle rotation of footstep for turning

The parameters should not be instantaneously updated in the middle of the walk cycle since the discontinuity may destabilize the robot. Parameters updates can either be slowly smoothed or be flushed only at foot support swap ($phase = 0$ or $phase = 0.5$).

The resulting motor target positions are displayed in Fig. 11.

C. Closing the Loop

In practice, the performance of this open loop walk engine is decent if used on a small humanoid robot with proper mechanical design. For example, it is essential to reduce the hip yaw backlash by using a needle roller bearing.

However, the system is still highly affected by small external perturbations, especially during the lateral footsteps. The major cause of falling occurs when the robot lateral oscillation dynamics get desynchronized with the open loop cycle.

A simple way to add a feedback reaction is to use the foot pressure sensors presented in section I in order to detect the actual support foot. The walk cycle phase can thus be adjusted by postponing the support foot exchange when not enough weight is detected on the next support foot.

Despite its first practical efficiency, further work is required to investigate the full use of the pressure sensors for the stabilization of the walk and to proceed to quantitative benchmark.

[8] Used values during Robocup 2015 can be found at `https://github.com/Rhoban/IKWalk/blob/master/Example/example.cpp`

[9] The center of mass trajectory is simulated from a complete kinematic model with masses of the robot

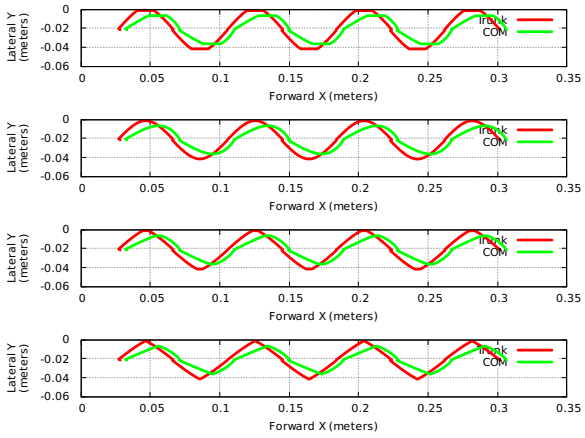

Fig. 12: The walk engine allows to choose several possible trajectories for the trunk and the center of mass during a forward walk. Top view of 4 possible trunk and CoM trajectories are shown.

V. Conclusion

We have presented three independent hardware and software open source contributions to the Robocup community. This effort will be pursued and we hope to be able to collaborate with other teams in the future in order to improve the sharing of knowledge in order to accelerate robots development.

References

[1] J. Englsberger, A. Werner, C. Ott, B. Henze, M. A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid, *et al.*, "Overview of the torque-controlled humanoid robot toro," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 916–923.

[2] G. Passault, Q. Rouxel, L. Hofer, S. N'Guyen, and O. Ly, "Low-cost force sensors for small size humanoid robot," in *2015 IEEE-RAS International Conference on Humanoid Robots (Video contribution)*, accepted.

[3] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *CoRR abs/0807.3223*, 2008.

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[5] P. Allgeuer, M. Schwarz, J. Pastrana, S. Schueller, M. Missura, and S. Behnke, "A ros-based software framework for the nimbro-op humanoid open platform," in *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE Int. Conf. on Humanoid Robots, Atlanta, USA*, 2013.

[6] O. Ly, H. Gimbert, G. Passault, and G. Baron, "A fully autonomous robot for putting posts for trellising vineyard with centimetric accuracy," in *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 44–49.

[7] S. Behnke, "Online trajectory generation for omnidirectional biped walking," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1597–1603.