Three Omni-wheel Ballbot Optimum Implementation

Iulian Calciu, Laura Mihaela Vasilescu, Adriana Drăghici, Daniel Rosner, Monica Pătrașcu
Faculty of Automatic Control and Computers
University Politehnica of Bucharest
Bucharest, Romania
iulian.calciu@aut.pub.ro, {laura.vasilescu, adriana.draghici, daniel.rosner}@cs.pub.ro, monica.patrascu@acse.pub.ro

***Abstract* — This paper presents novel three-wheel optimum implementation of a ballbot, a robotic platform mounted on a sphere with a single point of contact with the ground. The ballbot's purpose is to maintain meta-stable equilibrium and to offer possibility of movement based on user input. Thus, we offer a study on the decisions that support the hardware design of this non-linear unstable system, along with a basic control system, reproducible in laboratory settings, both from research and educational approaches.**

***Keywords* — ballbot; omni-wheel; control system; non-linear system; hardware implementation; education applications.**

I. INTRODUCTION

A *Ballbot* [1] is a mobile robot that maintains balance on a ball and has a single point of contact on the ground. This makes it an unstable system, even when motionless. Such a robot uses brushed motors as actuators, and makes contact through omnidirectional wheels (omni-wheels). Modern *ballbot* robots use fuzzy systems, genetic algorithms, and other artificial intelligence tools for control, but more widely used are the classic PID (Proportional-Integral-Derivative) control laws. Ballbot stabilizing controllers must generate commands for each motor that take into account issues of stability, as well as the robot's translation and rotation, data which is obtained from various sensors (for example, gyroscopes to indicate the tilted degree on three axes: OX, OY, and OZ). Additional supervisory algorithms may be applied in order to track the object in larger space and/or contexts, for instance, when talking about deliberative control architectures.

In this paper, the primary focus is stabilization of the robot. The required control laws depend on the number of the motors and, more importantly, on the way in which they are connected. For example, a classical implementation uses four motors mounted so that the generated force is perpendicular on the axis of each motor. Opposite motors will receive equal-value commands of opposite signs. Three motor models have a more complex computational aspect, because the command forces will be located at different angles from the gyroscope axis, thus making reactive methodologies inappropriate. Moreover, a ballot has more degrees of freedom than a simple inverted pendulum because the sphere allows travel in any given direction.

Various implementations have been successfully tested in recent years. For instance, *Rezero* [2] is a robot developed by a team of 10 students from ETH Zurich. In order to make it follow paths, the system used the sphere's movement in the desired direction by controlling it with the three omni-wheels available, at same time keeping the robot in the meta-stable position. *Rezero* measures the tilt 160 times per second. Even a very small error of runtime can make the robot to fall. *AIT Ballbot* [3] was designed in the Mechatronics Laboratory of Asia Institute of Technology. In contrast to *Rezero*, it uses four omnidirectional wheels, which is a significantly easier model with respect to the computational resources. The *NXT LEGO Ballbot* [4] uses only two common wheels and is the simplest ballbot model, with its motors mounted perpendicularly on the OX and OY axes.

This paper is organized as follows: section II presents our proposed *ballbot* design, section III showcases the control algorithms we used, sections IV and V contain the hardware and software implementation, respectively, while section VI presents conclusions and further work.

II. PROPOSED *BALLBOT* DESIGN

The proposed robot design makes use of an electronic board, 3-4 actuators (motors), omnidirectional wheels and some mechanical elements to put everything together. It also has a few sensors and a Bluetooth module to send data to the PC through a custom made application. The data received by the computer is processed and plotted by our C# application.

One of the most important design requirement that the *ballbot* must satisfy is to have the ability to remain in a fixed and stable position, characterized by meta-stable equilibrium oscillations around a stable limit cycle [5]. To ensure this behavior, the robot must be always active in terms of receiving and executing commands. In order to quantify its balance on the sphere, the robot must know its position relative to an axis perpendicular on the ground. Our implementation resolves this issue by using a gyroscope and a sensor to capture the robot's tilt. Another aspect is that the robot requires high resolution commands to maintain its balance. It requires the use of high-torque engines with low rotation speed. To satisfy those constrains, we chose 300:1 geared motors which reach a maximum speed of 100 rpm.

Another important factor is the microcontroller: it has to perform real-time actions (< *10 ms*) and also have a battery

978-1-4673-8180-2/15/$31.00 ©2015 IEEE

with large enough capacity to ensure good running time of the robot without making it too heavy.

Besides the control functionality, the robot's behavior must be monitored. For this we used a Bluetooth module, which sends the robot's operating parameters to a computer that processes and displays it in a graphical interface, as shown in figure 1.

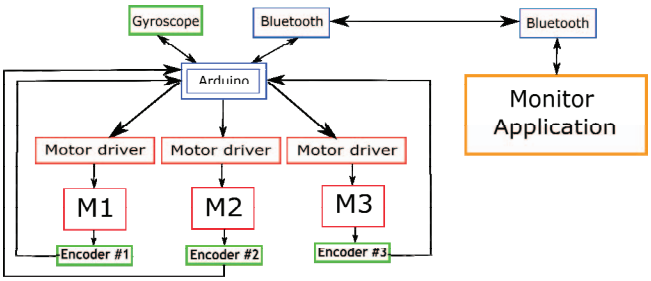

Fig 1. Ballbot design block diagram

The microcontroller reads data from the gyroscope, and applies the control algorithm in order to compute what commands must be executed by each of the motors in order to keep the robot stable. The motors execute the commands, and, after that, encoder data for motor monitoring is sent through the Bluetooth module to the computer application.

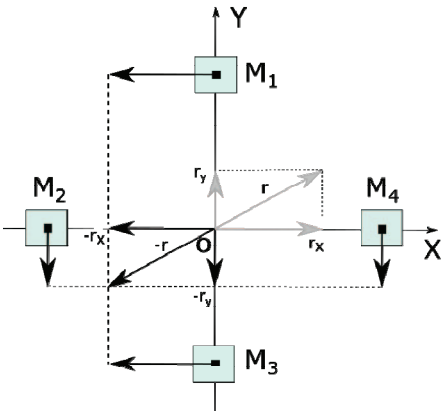

Fig 2. Four wheel ballbot model

To get to the final design version, the robot passed through several transformations. First, we developed a robot case, adaptable to any type of ball, with diameters between 5 cm and 15 cm. This helped us to find a sphere that is adherent to the surface, without concern about its size. Second, we changed the normal wheels with omni-wheels that allow any type of robot motion, eliminating the risk of blocking motors or wheels. In order to maintain the robot's oscillations inside a small enough vicinity of the limit cycle, we tried to lift and lower its gravity center until we concluded that making the gravity center as low as possible will make the robot much more stable.

For the first version of the robot, we chose to use four motors, thus making it is simpler to compute the forces acting upon the motors and subsequently their reaction forces (figure 2). The sensors collect inclination values on OX and OY axis (i.e. r_x and r_y). Even though the software implementation of this approach is facile, the hardware development is much more complicated as we were unable to align all four wheels to have exactly the same grip on the ball. This is important because the 4-wheel robot's model is a MIMO system with 2 inputs and 2 outputs, motors 1 & 3, and 2 & 4, respectively, receiving the same value commands. Control errors are represented (figure 2) by the axis tilting viewed from motor centers: error for M_1 and M_3 is r_x; and error for M_2 and M_4 is r_y.

Seeing that that mechanical issue is intractable, we chose to transform the robot into one with three wheels to solve the alignment problem, because any three points will be coplanar so it will be easier to place the robot on the sphere's surface, as shown in figure 3.

The modifications consist of removing a motor and shifting the three remaining motors in an equilateral triangle setup. Thus, the angles between the motors are 120 degrees. In this case, the robot behaves much better in terms of stability and the wheels have a greater grip.

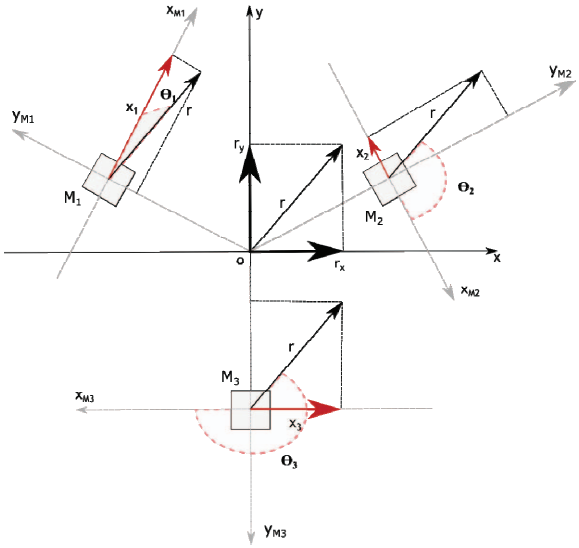

Fig. 3. Three wheel ballbot model

In this implementation, we chose to place on OY axis along motor M_3, whereas M_1 and M_2 are located at 120 degrees to the left or to the right side of it.

Starting from the slopes to the axes OX and OY, r_x, respectively r_y, we had to compute the forces acting on the centroid of each motor. We determined the resultant force r and the angle φ, made by r with the OX axis, obtaining the following:

$$r = \sqrt{\left(r_x^2 + r_y^2\right)} \tag{1}$$

$$\varphi = arctg\left(\frac{r_y}{r_x}\right) \tag{2}$$

The main goal is to compute x_i the projection axis of the tilt on each motor $i \in \{1,2,3\}$, so that we can control the robot's movement. We considered that the OY axis is the motor support and OX is the motor moving axis, writing the projection for each motor on their OX axis as a function of r and φ:

$$x_i = f(r, \varphi) \tag{3}$$

For the first motor we moved the tilt resultant force in its center and we have reached the angle:

$$\Theta_1 = \varphi + 150 \tag{4}$$

Resulting the projection value:

$$x_1 = r \cdot \cos(\Theta_1) = r \cdot \cos(\varphi + 150) \tag{5}$$

For the second motor M_2, we applied the same procedure as above:

$$\Theta_2 = -\varphi - 30 \tag{6}$$

$$x_2 = r \cdot \cos(\Theta_2) = r \cdot \cos(-\varphi - 30) \tag{7}$$

In the case of the third M_3 motor, because the motor's OX axis is parallel to the OX axis of the robot, the OX axis tilt projection of the motor M_3 is equal to the resultant tilt of projection r_x:

$$x_3 = r \cdot \cos(\varphi + 0) = r_x \tag{8}$$

Thus, the control errors of the three-wheel robot system are represented on the OX axis tilt of each motor, respectively x_1, x_2 and x_3.

III. *BALLBOT* CONTROL ALGORITHMS

An essential part of the robot's design is the control algorithm. First, we implemented a proportional [6] algorithm (figure 4) of parameter K:

$$u_k = K_P \cdot \varepsilon_k \tag{9}$$

where its output u_k is linear dependant on the controller inputs ε_k (with k the discrete time step), and added a hysteretic behavior in order to compensate the dead zone of the motors. In this case, as inputs for the algorithm we chose the OX and OY axes tilts of the gyroscope, which means rear-front, respectively left-right tilting.

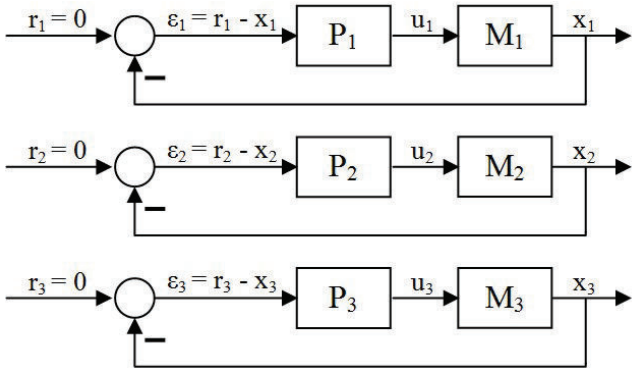

Fig. 4. P control system

In figure 5, parameter K_P is the slope of the command characteristic, varying from 1 for the maximum motor speed and decreasing to 0, taking into account that the command signal PWM values range between 0 and 255.

We established that the robot does not react at very small tilts and the maximum command will be sent at very large values of the inclination.

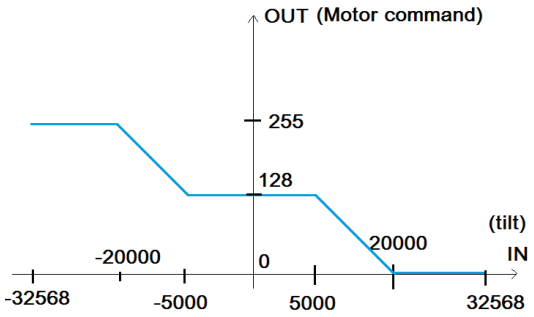

Fig. 5. Proportional algorithm command

This algorithm has proved ineffective, because more than one signal acts on each motor, thus, the system is strongly coupled. These inputs have been considered disturbances, but have not been rejected by the algorithm (proportional laws do not ensure disturbance rejection, nor zero steady state errors).

We improved the control system with a PD type algorithm (figure 6), by including a derivative [6] in the previous algorithm to compensate for sudden inclines or imbalances of the robot, which can not be handled by a purely proportional law.

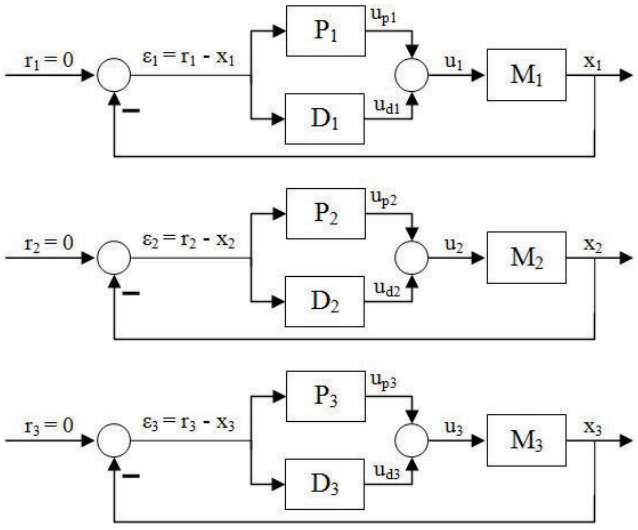

Fig. 6. PD control system

In order to implement derivative algorithms, the sampling time of the system is required (the most common discrete derivative of a signal is the ratio of two consecutive signal values over sampling time T_s). Systems of the ballbot's type have sampling times below 20 ms in closed

loop, so we chose a constant T_s = 7 ms, making sure the computational requirements would not exceed this value.

For this algorithm, the controller inputs are the control errors ε_k (with k the discrete time step), while their outputs u_k are the motor commands. The PD output signal is a linear combination of error and derivative of error, weighted by constants K_P or K_D:

$$u_k = K_P \cdot \varepsilon_k + K_D \frac{\varepsilon_k - \varepsilon_{k-1}}{T_s} \quad (10)$$

The tuning parameters of the PD control law were obtained experimentally, through a trial-and-error method based on the usual practices employed in automation and real world control systems design.

IV. HARDWARE IMPLEMENTATION

The main part of the Arduino Micro electronic board is the microcontroller, ATmega 32u4, from Atmel, a microcontroller with a very small QFN case. It contains all the necessary peripherals for the project implementation, operates at a 16 MHz suitable frequency and it is easy to program. The used peripherals are:

- UART serial interface: used for Bluetooth communication;
- I_2C interface: used for communication with the gyroscope and accelerometer;
- PWM Modules: We needed 4 pulse width modulation channels for the *ballbot* 4 motors;
- Analogue Inputs: 3 ADCs are used to read data from proximity sensors;
- digital inputs and outputs are used for reading data from encoders and for starting motor drivers.

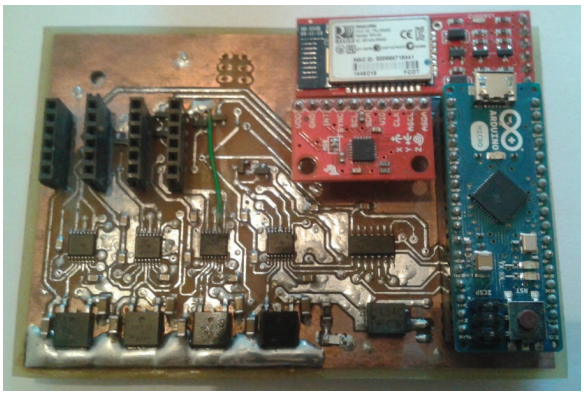

Fig. 7. Ballbot electronic board

The electronic schematic of the robot was developed in CadSoft Eagle. Because the components where not found in the common libraries, we also had to implement the program symbols and layouts.

The board layout suffered several transformations until we realized a very compact 10cm x 7cm form, containing all the necessary functional components for the robot. We put the Bluetooth module into one corner of the board to avoid being interfered with other components and gyro sensor was positioned towards the board middle to feel the best robot tilts. The remaining components were placed in rest of the board so as to minimize the space occupied by them. The routes have minimum dimensions that can be developed in laboratory, except the power routes that were made thicker, to allow bigger currents.

V. SOFTWARE IMPLEMENTATION

Our project contains two software applications:

- Arduino Application: reading, filtering and sending data; it also contains stabilization algorithms;
- C# developed PC application: robot parameters monitor.

Both programs communicate with each other, for optimum robot control.

The Arduino application contains the implementation of several libraries: for reading and filtering data from the gyroscope and accelerometer, for Bluetooth communication and for control algorithms.

The data read from the gyroscope, is raw data with integer 16-bit values between -32768 and 32767. Their structure is not altered during the program running, and the computation is done based on average values considered more accurate by the filtering algorithm. Gyroscope sends data via I_2C interface for each of the three axes X, Y and Z inclinations.

For Bluetooth communication we need to initialize UART peripheral device and to set a baud-rate of 115200 bps, standard transmission rate for our Bluetooth device. Bluetooth device has a fairly simple protocol and accepts character string commands. These are described in device programming manual.

In the control libraries we have implemented functions to translate the forces in the coordinate system of the motors, in order to compute its projections on the OX axis, the axis of rotation for each wheel.

We divided 16-bit read inclinations (values ranging between -32768 and 32767) and set them into the range of available commands for motors (-128 to 127). To compute the forces projections on the axes we applied formulas deducted in section III for each motor tilt.

Finally we compute the control error on each axis. From the proofs from section III, these are the following values of errors for each motor:

- M1: $error_1 = r \cdot x \cdot \cos(\varphi - 1.0471)$;
- M2: $error_2 = r \cdot x \cdot \cos(-\varphi + 2.0943)$;
- M3: $error_3 = x$.

Furthermore, for the PD algorithm, we computed one derivative for each of the 3 regulators as shown in eq. 10. Finally, each command is sent to each motor.

The obtained system reaches and maintains the desired stable limit cycle, with small enough oscillations around the equilibrium point that do not topple the robot from the sphere.

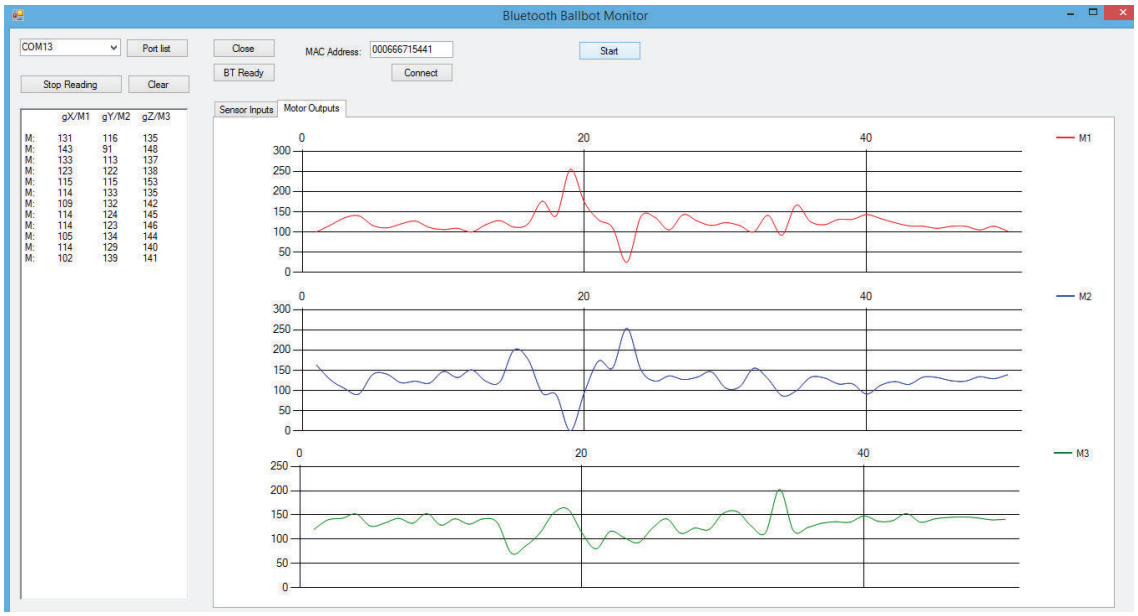

Fig. 8. Ballbot PC monitor application

The Arduino application also has the role to send data via bluetooth and generates commands for the motors. The PC application is an interface for monitoring parameters. The communication interface has some buttons for automatically finding the port where bluetooth module is connected.

One problem encountered in serial communication was data synchronizing. Thus, we developed a simple protocol, that sends two sync bytes of data and then send the useful data. This is implemented in the Arduino application and in the PC monitor too.

The main functionality of the application is to display data received from the serial interface. This gives the possibility of displaying entries from gyroscope on each of three axis, as well as the motors rotation speed. Thus, we can monitor in real time the evolution of all parameters of a wireless controlled robot.

Each graph displays the previous 50 measurements that are saved in a queue structure. This technique allows the user to have live preview of the graphics, by processing the acquired data in real time. The measurements for monitoring are done every 200ms, as we determined experimentally that this is the perfect sampling interval in order to generate smooth graphics without unnecessarily overtaxing the system.

VI. Conclusions

In this paper we have developed an application that uses a complex model of an omni-directional inverted pendulum for a mobile robot. We have implemented real-time monitoring of the robot parameters, as well as basic stabilizing control algorithms.

The educational value of this paper comes from several perspectives. First, the ballbot allows for extensive study of nonlinear unstable systems, and gives students a priceless visual understanding of limit cycles and meta-stable oscillations around equilibrium points.

Moreover, the ballbot offers examples of including real-time concepts into programming of monitoring and control applications, being useful in laboratories that deal with hardware-in-the-loop and even human-in-the-loop systems.

Last, but not least, the design procedure can be followed by students and researchers in similar applications, thus narrowing the gap between theory and implementation in both computer science and control engineering.

Acknowledgements

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/187/1.5/S/155420.

References

[1] Hollis, Ralph. "*Ballbots*" Scientific American 18 (2008): 58-63.

[2] Leutenegger, Stefan, and Péter Fankhauser. "*Modeling and Control of a Ballbot*" ETH Zurich bachelor thesis (2010).

[3] Sukvichai, Kanjanapan, and Manukid Parnichkun. "Double-level ball-riding robot balancing: From system design, modeling, controller synthesis, to performance evaluation." *Mechatronics* 24.5 (2014): 519-532.

[4] Yamamoto, Y. "*NXT ballbot model-based design-control of a self-balancing robot on a ball, built with LEGO Mindstorms NXT*." (2009).

[5] Khalil, Hassan K., and J. W. Grizzle. *Nonlinear systems*. Vol. 3. New Jersey: Prentice hall, 1996.

[6] Aström, Karl Johan, and Richard M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.