2017 4th Experiment@ International Conference (exp.at'17)
June 6th – 8th, 2017, University of Algarve, Faro, Portugal

Control System for a Self-Balancing Robot

Ricardo Santos Martins
Dept. of Industrial Electronics
University of Minho
Guimarães, Portugal
ricardomsantos@gmail.com

Francisco Nunes
Dept. of Industrial Electronics
University of Minho
Guimarães, Portugal
Franciscoluis_9@hotmail.com

***Abstract*— The control of a self-balancing robot has been studied for many years and, despite several achievements, there are still open issues. The aim of this project is to study the efficiency of different control algorithms, as Proportional, Integral and Derivative (PID), pole placement, adaptive control, among others, in a home-made robot called Bimbo. It was also tested an algorithm, applied to the position. The robot was constructed with the modules for movement and position control. It was applied a Kalman filter to get the Roll angle from the Inertial Measurement Unit (IMU). Furthermore, the mechanism to read encoders and to control the two motors was implemented. It was built a mechanism to control system variables through Bluetooth communication, which allows to continuously monitor any robot variable, allowing to test the system and the control variables in real-time. The selected solution implements a PID, continuously changing the reference by a "position algorithm". A human control interface was created to command Bimbo navigation direction.**

Keywords—PID; Kalman Filter; Controler; "Position Control".

I. INTRODUCTION

The equilibrium control of a two-wheeled mobile robot is a topic of high interest for the scientific community, particularly in the field of engineering, creating impacts on society. Many authors have been studying these robots and their main goal is to find the perfect math model and structure characteristics. Some examples that follow the same principles are segways, automatic wheelchairs, among others [1]. Segway is the most well-known example of two-wheeled robots. They may have different usages like indoor/outdoor locomotion, industrial automation, defense application and, nowadays, the personal mean of transport.

The Segway Personal Transporter [2] is one of the most common approaches to the seagway robot in the market. Close approaches have been found and tested using different controllers. Silva *et al*, 2015 using a state model and a Linear Quadratic Regulator (LQR) combined with Proportional, Integral and Derivative (PID) controller succeeded to build a two-wheeled balancing robot [3]. This kind of self-balancing robots is very similar to the inverted pendulum, which is a recurrently studied topic in academic works. These robots have been widely studied in search of a "perfect" and "all-compatible" algorithm [4]. Some of these attempts use classic and linear multivariable algorithms [5], nonlinear backstepping controls [6], fuzzy-variable controls [7] and combinations of these techniques [8].

Following this trend, the work resented in this paper is focused on the development and control of a two-wheeled robot. The project was developed within a course project (Project I) at the Integrated Master of Industrial Electronics and Computers Engineering at the University of Minho. It aims two main goals: (i) to study physical and dynamical characteristics that a two-wheeled robot exhibits; and (ii) to experiment different control techniques to improve its performance. Furthermore, the solution under study should be as generic as possible.

This remainder of the paper is organized as follows: section II presents the main control algorithms employed, PID and the approach named "position control", an improvement studied to overcome the limitations detected in the performance of the PID controller; section III describes the solutions adopted in the construction of Bimbo, at hardware and software levels; section IV synthesizes the obtained results; and section V draw some conclusions and discuss future work.

II. FUNDAMENTALS

The general approach to address the equilibrium problem consists in implementing a PID controller, choosing holistically the parameters, or a Pole Placement technique, when the state model is known (which rarely happens). In either case, it is frequently difficult to find a satisfactory solution. In order to achieve a generic solution, a home-made "position controller" was developed. Giving the importance of these solutions, this section starts describing them in more detail.

A. PID

The Proportional-Integral-Derivative controller is one of the most common used controllers since it is not necessary to know the model equations. It is a controller that, given an error (distance to target), considers the in-moment error (proportional component), the previous errors (integral component) and the difference between the current and last error (derivative component). It is possible to weight each term through the tuning of the controller parameters proportional, integral and derivative gains, K_p, K_i and K_d, respectively.

Fig. 1 presents the block diagram of a PID continuous time controller in a closed loop configuration, highlighting the three terms of the controller: proportional, integral and derivative actions. However, this model cannot be directly implemented by software since it is continuous-time based.

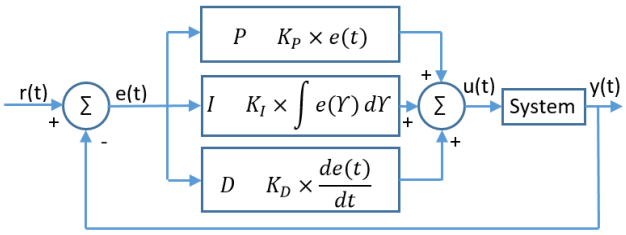

Fig. 1. Representation of a block diagram of PID continuous time controller.

To overcome this issue, it is required a discrete time model, which can be represented by equation (1).

$$u(k) = K_p \times e(k) + K_i \times T \times \sum e + K_d \times T^{-1} \times (e(k) - e(k-1)) \quad (1)$$

In equation (1), u(k) represents the control action (output variable of the controller), e(k) represents the error (set point - measured process variable), k represents the discrete time, T is sampling time and K_p, K_i and K_d are the controller gains [9].

To reach the stability of the robot these three gains must be well tuned, either by simulation or by trial and error. To address this issue, there are techniques like Ziegler–Nichols rules that tries to tune the parameters by observation, following a type of trial-error approach. For simple systems it may work, but not for all [10].

B. "Position Control"

Trying to overcome the state model complexity, and researching different alternatives, some promising solutions came up based on dynamic adjust of the controller reference, using the linear position and the velocity as inputs and producing an offset as output (this offset is then added to the controller reference). Following a trial-error approach, several experiments were performed: (i) first, using only the linear position, which produced a more stable behavior but not capable of responding to high speed stimulus; (ii) second, using the velocity to scale down the angular error; and (iii) mixing both approaches. It is important to note that in this approach there are no model or formal studies to follow since it is based solely on experimental work.

This method has just two requirements: (i) a controller, PID type (robot current angle as input, PWM value as output and desired angle as reference) for example; and (ii) a position sensor, like an encoder. Lets consider two real scenarios to better understand how it works:

- Suppose the robot is standing up, and it is pushed to the left side. The position algorithm will make the reference change to the opposite side (right side), proportionally to the distanced travelled. Only this component would make the robot reach the original position, with a target angle of 180 degrees (standing up), but probably with some speed, that would make it go to the right side. So, using only this component there are two main disadvantages: the original position would be constantly overpassed (oscillation over original position); and if the robot is too far from the original position, the target angle would be so low (or so high) that the robot would fall.
- Now, assume the robot is too far away from the initial position. As mentioned, the target angle would be very high (or low) and the robot would fall. In this situation, before the fall, the robot would get its maximum speed to keep the new target angle. So, the solution is to scale down this speed, by adding or removing an offset to the target point. After all, if the robot is moving too fast to the right side, the velocity component will set the target angle more to the left side. It is also important to mention that this component prevents the robot from falling down in two scenarios: pushed too hard (large perturbation); and huge position error.

With the combination of this two components, the robot: (i) will not present high oscillations from original position; (ii) in most cases it will not fall with perturbations; and (iii) it will always go to the original position after perturbations.

Fig. 2 illustrate the way the algorithm behaves in two typical scenarios, assuming an 180 degrees reference. The red arrow represents the offset in degrees obtained by position error, and the blue arrow represents the offset in degrees obtained by velocity (the difference between the last two positions in a sampling period). In both cases, the resulting offset (sum of blue and red offsets) will be added to the reference. In the left side, despite being far from the initial position, the robot will be forced to approach that point but avoiding too high speeds (that would make it fall). On the right side, after the induced perturbation the robot will be highly forced to react to it.

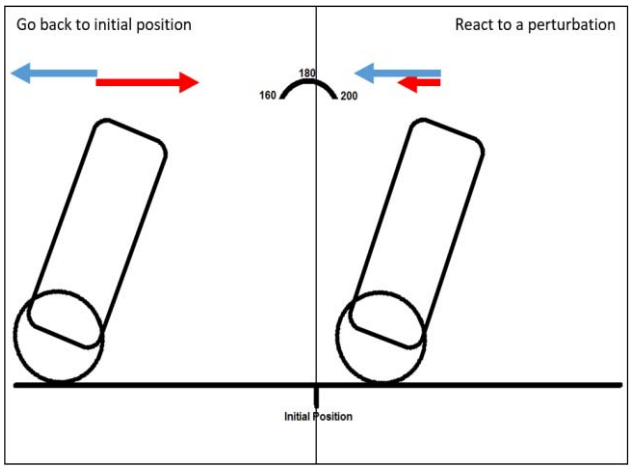

Fig. 2. Illustration of the algorithm behavior in the two main scenarios.

Besides, this algorithm has some parameters that can help the stability of the robot, recurring to Programmed Gains (tuned by

trial and error), to define different gains for different situations. For example, assuming K_1, K_2 and K_3 are programmable gains, the algorithm can be programed with three states: (i) 0 to 50 cm away from initial position; the offset will be obtained multiplying the error (current position – initial position) by K_1; (ii) 50 to 100 cm away from initial position; the offset will be obtained multiplying the error by K_2; (iii) more than 100 cm away from initial position; the offset will be obtained multiplying the error by K_3. The same technique can be applied with the velocity, programming it for several scenarios. These programmed gains are meant to make the robot recover faster from perturbations.

It is important to mention that this technique is similar to an "observer" because it does not interact directly with the variables of the balancing system. It is a block that stays away from the controller part, watch the system and then alter the reference of the controller.

III. Work Developed

This section describes both the mechanical and software procedures followed to build Bimbo.

A. Physical Characteristics

Fig. 3 presents the physical structure of Bimbo, which is a Lego based robot, with precise geometry and robustness. The structure has four main modules: (i) the bottom part that carries the motors; (ii) the second module to hold the electronic material (Arduino, sensors, H-bridge, among others); (iii) the third to carry the battery; and (iv) the last one, at the top, where the user can place objects to test the equilibrium, or use it to install more electronic.

Bimbo includes a 4800 mah 12V battery, an Arduino Pro Mini 5V/16MHz, a Pololu L298 Dual H-Bridge, two Pololu 30:1 Metal Gearmotor 12V 500rpm with 64 CPR Encoder motor, a Pololu 6 DOF IMU-MPU6050 sensor, an HC-06 Bluetooth module and a 3 digit display voltmeter to show battery voltage level. The complete assembled system can be seen in Fig. 3. Concerning physical dimensions, Bimbo is 15.8x16.1x7.1 cm size, 1062 g weigh and has two 9 cm diameter wheels.

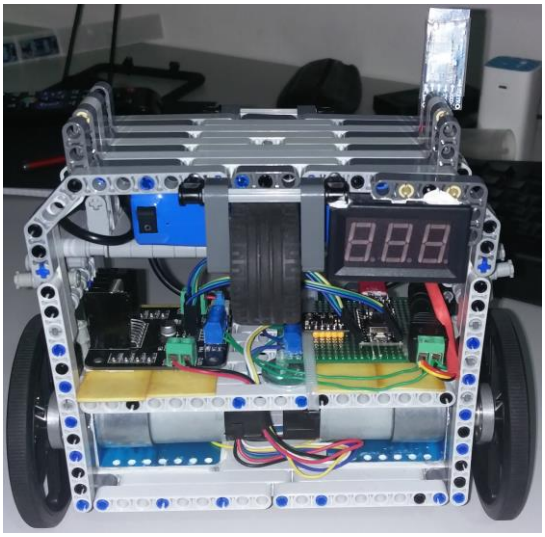

Fig. 3. Fully Assembled Bimbo Robot.

B. System Software

The software architecture developed for this project includes a set of standard libraries essentially to control motors and read the different sensors. This was designed as a kind of "software lab", allowing to test the controllers with the desired detail. This "lab" is user-friendly and flexible enough to allow all types of experiments. The design includes five libraries:

- Motor: allows to declare multiple motors; each one may have different variables (e.g. minimum stimulus value; or maximum stimulus value – should be set to prevent battery powering off), and some functions to control the motors (namely spin on and direction);
- Encoder: the motors have a 30:1 gearbox, with a 64 CPR encoder, which means that one rotation of the motor shaft should rise 30 × 64 = 1920 events. Events are associated to Arduino interrupts, which, using the Arduino Library, are impossible to handle. However, in this project, the full counting is not required. This allowed to use standard Arduino functions, counting an average of 320 ticks per wheel rotation;
- MPU 6050: responsible for the communication with the sensor accelerometer and gyroscope. The values read from this sensor are extremely noisy, requiring a filter. Both Complementary and Kalman filters work fine with this type of applications, but since Kalman has a faster response, it was chosen;
- Kalman: as stated above, this library is used by the MPU 6050. The implementation of this filter follows Kristian Lauszus [11] solution, adequately adapted to Bimbo;
- PID: responsible for "in position" balancing. This library was built with a fully operational control in mind,

allowing to test the impact of different gains and set points.

After the implementation of these libraries, each one was tested individually. To do so, a mechanism to check in-time variables was created, using Matlab as a monitoring tool and a Bluetooth interface to read the robot variables. From the Matlab side it was developed an algorithm to continuously show a time/signal variation graphic. This algorithm can show as many variables as the user wants, limited by the same axis scale, but allowing an individual input scale factor.

Within this framework, all libraries were tested showing the desired response, but there were three limitations. The first, one motor runs faster than the other, in a non-linear way – it was corrected by reading the encoders values, forcing a motor to move slower. The second one, for the different PWM values applied the battery voltage varies. This is a problem because common controllers output is assumed to have a linear behavior, modeled by equation 2 (where V_{in} is the battery voltage and V_{out} it the voltage applied to the motors) and assuming V_{in} constant. We could observe through the display that different PWM values force V_{in} to change. Besides, the state of the battery alters that response, making it harder to create a linearization operation. This problem was overcome with the previously mentioned "position control" algorithm.

$$V_{out} = (PWM / 255) \times V_{in} \quad (2)$$

Finally, the third problem was the noise presented in the accelerometer and gyroscope sensor output. Given the sensor position on the board, the parameter that Bimbo needs to equilibrate itself is the Roll (longitudinal axis angle), calculated by equation 3:

$$Roll = (\arctan(accel_y / accel_z) + \pi) \times RAD\_TO\_DEG \quad (3)$$

where $accel_y$ and $accel_z$ are, respectively, the accelerometer values in x and y axis and RAD_TO_DEG is the constant that convert radians into degrees. Although the sensor was kept immobile, the Roll shows high oscillations (noise). As mentioned, a Kalman filter was implemented to solve this issue. Using the test framework it was possible to change Kalman filter parameters and, observing the response in real time, adjusting them for best performance as can be seen in Fig. 4 – orange line represents the angle calculated with equation 3 and the blue line represents the calibrated filter output.

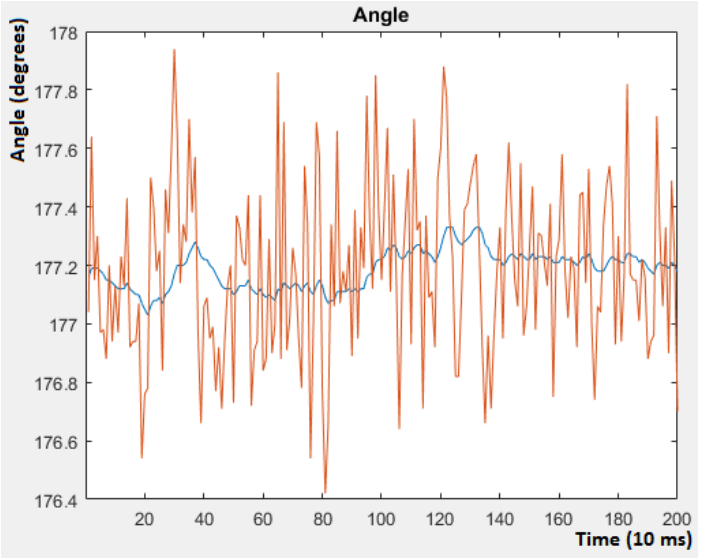

Fig. 4. Matlab display for Kalman filter analysis.

The response shown in Fig. 4 was obtained with the sensor perfectly immobile, where non-filtered roll showed a maximum oscillation of 1.6 degrees. If the sensor is held in a non-stable position (which is a more suitable scenario for the real Bimbo operation) larger variations can be measured. With the Kalman filter those variations have a reduced impact and the final response keeps very smooth.

C. Main Application

Initially, a PID controller was implemented, showing unsatisfactory results. The solution was implemented using the "position control" algorithm (described in section II). With a PID only with proportional gain and the "position control", Bimbo no longer fall down under induced perturbations, showing a satisfactory balancing response. However, some oscillation at the initial position was perceivable.

To improve this response, the following strategy was adopted: (i) the "position control" was removed and the PID gains were calibrated according to Ziegler–Nichols rules, ensuring smooth and less oscillatory response – yet, Bimbo always ended up falling; (ii) the "position control" was enabled again and some minimal adjusts to PID gains were performed. The final result was a stable balancing behavior and a non-falling robot was reached, fulfilling initial requirements.

In order to achieve Bimbo stability, navigation functions were implemented: (i) move forward; (ii) move backward; (iii) turn right; (iv) turn left; and (v) without any of the above commands, retain immobile. Functionally, operations (i) and (ii) are controlled by commanding the target angle. Considering that the PID output is an "absolute" value, adding symmetrical values to it before sending the final values to the motors will force rotation, without losing equilibrium (for example, a PID output of 100 applied to two motors, produce the same effect as applying 120 to one and 80 to the other, making it turn).

Finally, an interface application to control Bimbo was created. This application is a computer terminal to interact with Bimbo. It includes a window to show Bimbo internal data, some buttons to adjust PID parameters, two slides to control the linear and turning speed and an "aggressive" button, used to scale slides control outputs, adapting operation to different types of soils (see Fig. 5).

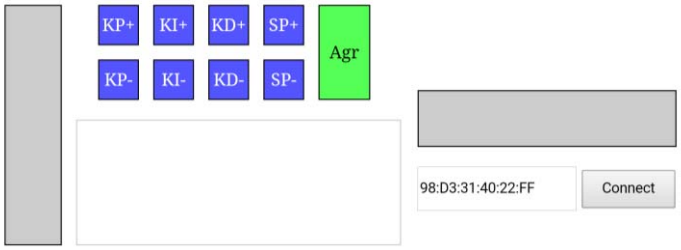

Fig. 5. Image of the Human Interface to control Bimbo

This application was developed in python and it creates a server in a host with Bluetooth interface, allowing, through a browser, to access the Bimbo remote terminal (Fig. 5). This kind of solution allows the user to access Bimbo without being nearby.

The flowchart describing the main application algorithm is depicted in Fig. 6. It is important to note that the loop time of Bimbo is 10 milliseconds. However, it could be reduced, since the loop functions takes about 5 milliseconds to execute, which would improve the stability. This change was not made because it would not allow Arduino to print some values on the terminal interface, which is considered a more important feature considering the cost-benefit relation.

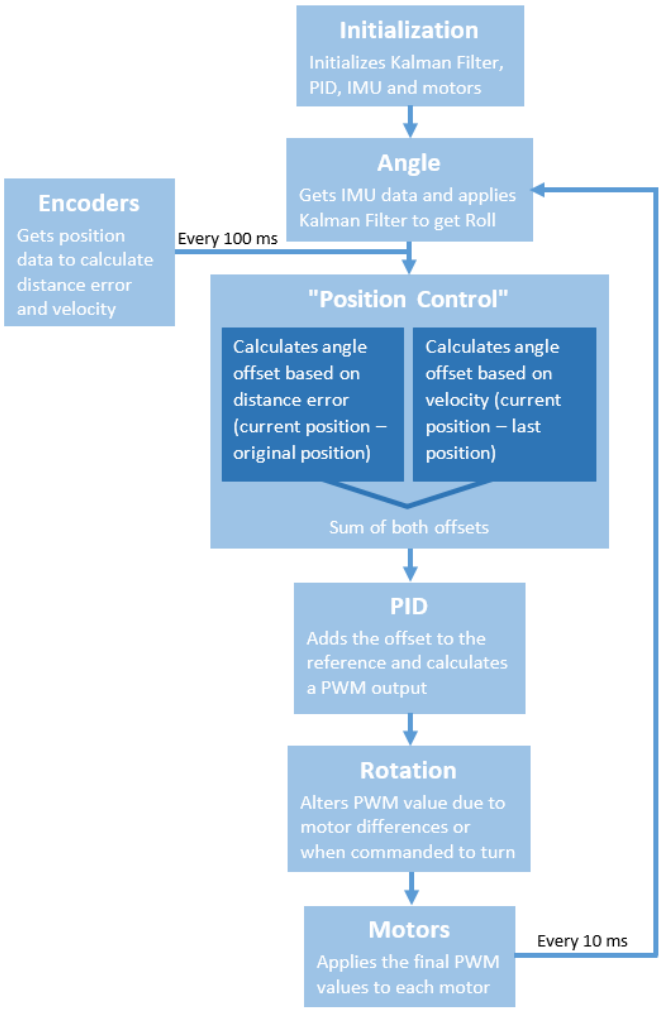

Fig. 6. Schematic of Bimbo main loop.

IV. Results

As a result of this project, it is considered that the main two objectives were reached, balancing a two-wheeled robot and finding an easy and multi-infrastructure controller. It is believed that this new approach can be used to control various types of robots (including inverted pendulum).

The stability of Bimbo was tested inducing perturbations in the form of external forces (pushes) and changing the robot mass (carrying cellphones, student cases, and tool boxes). Its performance can be seen at https://youtu.be/56pK1ym2mJs.

In Fig 7 is shown the Bimbo behavior reacting to two perturbations (at instants 5 and 13 seconds). As it can be seen, the original position is always reached in few seconds. For small perturbations (instant 5) it reaches the original position without oscillations, but for huge perturbations (instant 13) it passes initial position showing some oscillations around it (instant 18 to 23). The time Bimbo takes to reach initial position is related to the

"position control" parameters. However, gains that induce a faster recover time will cause more oscillation. At the bottom, an illustration of real Bimbo position is shown, where: (i) blue representations means that Bimbo is stable and at initial position; (ii) red representation means that a perturbation was induced and it is away from initial position; and (iii) green representations means that Bimbo is recovering from the perturbation and it is trying to achieve the initial position.

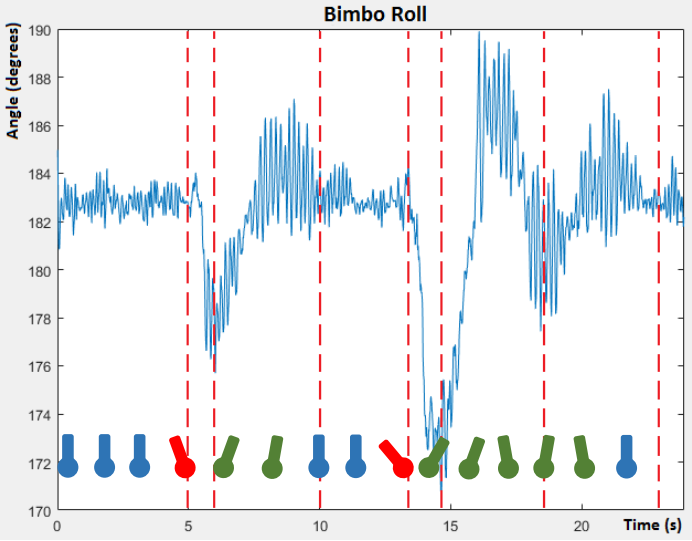

Fig 7. Graphic showing the behavior of Bimbo reacting to two perturbations.

A final obtained result, not directly linked to the main objectives of this project, is the Matlab based workbench developed, which can be considered a general purpose real-time monitoring tool useful for testing and debugging any type of signal processing application.

V. Conclusions

The main goal of this project was the development of a simple and generic self-balancing robot solution. Bimbo was the prototype developed that allowed to test different controllers (Pole Placement, PID and "position control").

Considering the different kind of controllers tested, Pole Placement technique showed a poor performance (a result of the lack of precision of the state model) while the PID, tuned with Ziegler–Nichols rules performs better, but still not fulfilling the initial requirements. A final solution was obtained by a PID with a dynamic reference adjustment method ("position control"). Besides, controlling dynamically the PID reference allows also to control robot movements.

The described solution seems to be a general purpose one and easy to apply to different types of robots. But this hypothesis needs further tests, which require some experimental work with different robots.

As mentioned previously, the "position control" techniques requires tuning some gains. In this project, these were tuned by trial and error. However, in the near future, it would be interesting to develop a method to tune them taking into account the sampling period (referred to velocity calculation) and the encoder ticks per rotation (referred to linear position calculation).

The next challenging step would be allowing Bimbo to navigate autonomously.

Acknowledgment

Recognition for the support on this project is dedicated to Professor Filomena Maria Rocha Menezes Oliveira Soares and Professor Estela Guerreiro Silva Bicho Erlhagen.

References

[1] R. Chan, K. Stol, and C. Halkyard, "Review of modelling and control of two-wheeled robots," *Annu. Rev. Control*, 2013.

[2] K. Gordon, "Segway Personal Transporter," *LAW ORDER-WILMETTE THEN DEERFIELD-*, 2006.

[3] V. Silva, P. Leite, F. Soares, G. Lopes, and J. Esteves, "Controlling an Equilibrist Lego Robot", *Proceedings of The World Congress on Engineering 2015, WCE 2015*, Editors: S. I. Ao, Len Gelman, David WL Hukins, Andrew Hunter, and A. M. Korsunsky, London, U.K., 1-3 July, 2015.

[4] H. Juang and K. Lurrr, "Design and control of a two-wheel self-balancing robot using the arduino microcontroller board," *2013 10th IEEE Int. Conf.*, 2013.

[5] F. Grasser, A. D'arrigo, and S. Colombi, "JOE: a mobile, inverted pendulum," *Ind. Electron.*, 2002.

[6] T. Nomura, Y. Kitsuka, and H. Suemitsu, "Adaptive backstepping control for a two-wheeled autonomous robot," *ICCAS-SICE, 2009*, 2009.

[7] K. Su and Y. Chen, "Balance control for two-wheeled robot via neural-fuzzy technique," *SICE Annu. Conf. 2010, Proc.*, 2010.

[8] X. Ruan and J. Cai, "Fuzzy backstepping controllers for two-wheeled self-balancing robot," *Int. Asia Conf. Informatics ...*, 2009.

[9] K. Kwok, M. Ping, and P. Li, "A model-based augmented PID algorithm," *J. Process Control*, 2000.

[10] D. Valério and J. da Costa, "Tuning of fractional PID controllers with Ziegler–Nichols-type rules," *Signal Processing*, 2006.

[11] K. Lauszus, "TKJ Electronics 2012," *Github*. [Online]. Available: https://github.com/TKJElectronics/KalmanFilter, accessed on 12 November 2016.