# Model-based Design for a Self-balancing Robot using the Arduino Micro-controller Board


1st Vincent Y. Philippart
*Electrical Engineering Department*
*Fontys University of Applied Sciences*
Eindhoven, Netherlands
v.philippart@student.fontys.nl

2nd Kristian O. Snel
*Electrical Engineering Department*
*Fontys University of Applied Sciences*
Eindhoven, Netherlands
kristian.snel@student.fontys.nl

3rd Antoine M. de Waal
*Electrical Engineering Department*
*Fontys University of Applied Sciences*
Eindhoven, Netherlands
a.dewaal@fontys.nl

4th Jeedella S.Y. Jeedella
*Electrical Engineering Department*
*Fontys University of Applied Sciences*
Eindhoven, Netherlands
j.jeedella@fontys.nl

5th Esmaeil Najafi
*Mechatronics Department*
*Fontys University of Applied Sciences*
Eindhoven, Netherlands
e.najafi@fontys.nl



***Abstract*—Self-balancing robot control is a common way to challenge students in control-oriented courses to stabilize an unstable, non-linear dynamic system. This paper presents an application of model-based design on a self-balancing robot using the Arduino Due micro-controller board. The system consists of a set DC-motors with quadrature encoders, a nine degree of freedom (9-DOF), a Bluetooth module, a motor controller and a micro-controller. By utilizing cheap off-the-shelf hardware and applying model-based design techniques and simulations an accessible learning environment is created. Both the PID and LQR controllers will be discussed for their educational value.**


## I. Introduction

In resent years, Segway proved the first successful human-friendly two wheeled balancing robot as a personal transport device [1]. The elimination of auxiliary wheels allows for increased mobility and traction [2], however, transforms the system into an unstable inverted pendulum system [3], [4]. The inverted pendulum system is a perfect example of a real-world implementation that requires textbook control algorithms for stabilization [5], [6] and is therefore suitable as an educational tool for students.

The TETRIX PRIME programmable robotics set provides the self balancing robot frame, the motors and motor control board. A MPU9250 9-DOF inertial measurement unit (IMU) sensor provides the angle measurements. The National Instruments myRIO embedded device is replaced with an Arduino Due with a shield to interface with the motor controller board. This resulted in our self-balancing robot as shown in Fig. 1.

Rather than a new system design, this paper emphasizes on solving the instability with controller design, model-based engineering implementation and code generation [7]. This way more focuses on the control domain of the self balancing robot rather than the construction and programming part is possible. To achieve this goal a model-based design method is applied using Matlab/Simulink. With Matlab/Simulink a mathematical model can be implemented on a micro-controller [8]. The Arduino Due with its Cortex-M3 is then programmed through Simulink to execute the mathematical model on the real world system. One of the main goals of this paper is to identify the efficiency of model-based design on a cheap micro-controller as a educational tool for students. Two implementations of control algorithms for the inverted pendulum model are the proportional integral derivative (PID) controller and the linear quadratic regulator (LQR) controller [9]–[11].

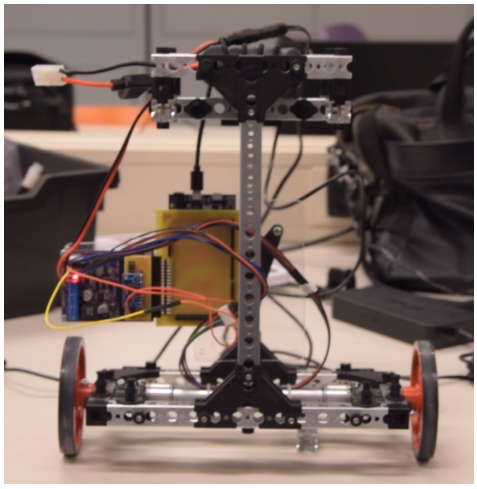


Fig. 1: TETRIX PRIME self-balancing robot, hardware replaced with the Arduino Due.

This paper is set-up in the following way. The use of Model-based design is described in Section II. Subsequently the dynamics of the self-balancing robot are derived and a control design are made in Section III. The simulations results are presented in Section IV and implementation on Arduino is given in Section V. Finally, Section VI provides a discussion and concludes the paper.


978-1-7281-3998-2/19/$31.00 ©2019 IEEE

## II. Model-based Design

Model-based design is a method with which complex physical and cyber-physical systems are mathematically modeled and/or empirically identified in order to create a virtual model to be controlled [12]. The method allows for accurate control model creation and synthesizing using real-time prototyping [8], simulation and data collection. Iterative verification tackles modeling and system identification errors such that they can be reduced to a minimum. Using the model-based design approach as an integral part of the engineering curriculum allows for future improvements in many fields which are otherwise too complex.

Model-based code generation, as commonly implemented by Matlab and Simulink is a useful way to deploy a primarily graphical model to a prototype containing a micro-controller. The use of code generation tools and their respective compilers have increased their footprint on the engineering industry due to requirement traceability in safety critical applications [7], [13]. Due to technology more commonly being integrated into the daily life of human beings, safety requirement and its traceability feature a more important role in the design-cycle and should thus be taken into account early in the design process to avoid errors triggering fundamental system redesigns [14].

Model-based code generation additionally reduces the need for manual computation and complex software algorithms, creating a more focused practical control-oriented curriculum for control theory courses [15]. Code generation tooling, especially with safety-critical coverage such as Simulink are somewhat limited in choice [16] but special licensed packages are readily available for students by increasing the use of university packaged software licenses. By eliminating the need for manual hardware programming, more emphasis can be put on the control elements and thus making it useful in an educational context.

## III. Dynamics and control

This section describes the system dynamical behavior model creation and the creation of the cascaded control loop.

### A. Architecture

The system architecture consists of sensors, power distribution, controlling and interfacing. The system block diagram of the self-balancing robot is depicted in Fig. 2.

### B. Dynamic Model

Accurate controller design requires an well defined dynamical model. Ideally, with model verification, the virtual model is compared to actual real-world dynamics. The graphical system model representation is depicted in Fig. 3. The parameter symbols and values are listed in Table I. The listed values were determined experimentally.

The non-linear model is linearized and derived as found in literature [17], [18]. The horizontal movement from the origin is $\mathbf{x}$, the applied torque in $\tau_0$, wheel angle in $\varphi$ and the rotational angle of the body is expressed in $\theta$. The linearized dynamic equations of the system are as follows:

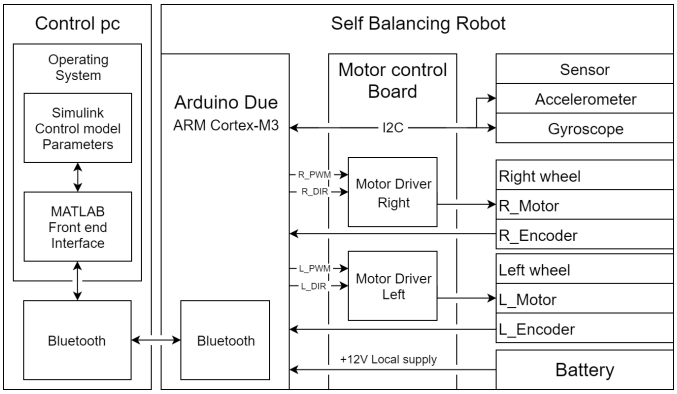



Fig. 2: Overview of the self-balancing robot system featuring a virtual simulation and parametrization environment and a physical self-balancing robot with a Arduino micro-controller.

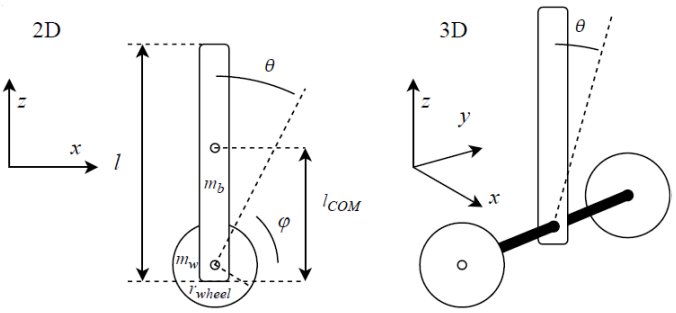



Fig. 3: Schematic representation of the self-balancing robot dynamics and corresponding physical parameters.

$$\begin{bmatrix} I_w + (m_w + m)r_{\text{wheel}}^2 & mr_{\text{wheel}}l_{\text{COM}} \\ mr_{\text{wheel}}l_{\text{COM}} & I_b + ml_{\text{COM}}^2 \end{bmatrix} \begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{bmatrix}$$

$$+ \begin{bmatrix} \beta_y + \beta_m & -\beta_m \\ -\beta_m & \beta_m \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ -mgl_{\text{COM}} \end{bmatrix} \theta = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tau_0 \quad (1)$$

Or in compressed form:

$$E \begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{bmatrix} + F \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \end{bmatrix} + G\theta = H\tau_0 \quad (2)$$

Resulting in the following state-space representation:

$$\frac{d}{dt} \begin{bmatrix} \varphi \\ \theta \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -E^{-1}G & -E^{-1}F & \end{bmatrix} \begin{bmatrix} \varphi \\ \theta \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} +$$

$$\begin{bmatrix} 0 \\ 0 \\ -E^{-1}H \end{bmatrix} \tau_0 \quad (3)$$

TABLE I: Physical parameters of the self-balancing robot

| Physical parameter | Symbol | Value | Unit |
|---|---|---|---|
| Wheel mass | $m_w$ | 405 | $g$ |
| Body mass | $m_b$ | 640 | $g$ |
| Total mass | $m$ | 1045 | $g$ |
| Wheel inertia | $I_w$ | $390 \times 10^{-6}$ | $kg.m^2$ |
| Body inertia | $I_b$ | $267 \times 10^{-3}$ | $kg.m^2$ |
| Wheel Diameter | $r_{\text{wheel}}$ | 4.55 | $cm$ |
| Gravity | $g$ | 9.81 | $\frac{m}{s^2}$ |
| Distance COM and wheel | $l_{\text{COM}}$ | 17.73 | $cm$ |
| Rolling damping ratio | $\beta_y$ | 0.01 | $\frac{N.m}{\text{rad}/s}$ |
| Friction damping ratio | $\beta_m$ | 0.01 | $\frac{N.m}{\text{rad}/s}$ |

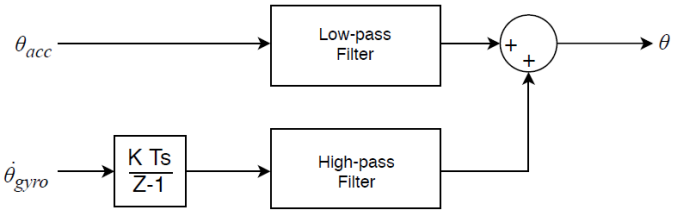



Fig. 4: Schematic representation of the self-balancing robot angle estimated diagram using a complementary filter.

And output matrix:

$$\mathbf{y} = \begin{bmatrix} r_{\text{wheel}} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \varphi \\ \theta \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} \tag{4}$$

given in the compact form:

$$\frac{d}{dt}\mathbf{x} = A\mathbf{x} + B\tau_0 \tag{5}$$

$$\mathbf{y} = C\mathbf{x} \tag{6}$$

### C. Angle Estimation

Estimation of the angle is performed with an accelero-meter and a gyroscope measurement. Angle estimation is done with a complementary filter [19], [20]. The filter consists of a high-pass and a low-pass filter. The complementary filter uses the accelero-meter data $\theta_{\text{acc}}$ and $\dot{\theta}_{\text{gyro}}$ for the final $\theta_{\text{est}}$. The accelero-meter data consists of multiple vector components, namely $\theta_{\text{acc},x}$, $\theta_{\text{acc},y}$ and $\theta_{\text{acc},z}$. The $\theta_{\text{acc}}$ is calculated as follows:

$$\theta_{acc} = \arctan \frac{\theta_{\text{acc},z}}{\sqrt{\theta^2_{\text{acc},y} + \theta^2_{\text{acc},x}}} \tag{7}$$

The complementary filter ratio is calculated as follows:

$$\theta_{\text{est}} = a \times (\theta_{\text{acc}} + \theta_{\text{gyro}} \times dt) + (a - 1) \times \theta_{\text{acc}} \tag{8}$$

With a filter factor $a$ of 0.97, as used in [19] and a cut-off frequency of $1Hz$, as used in [21]. The angle estimation diagram of the self-balancing robot is depicted in Fig. 4.

TABLE II: Motor parameters of the self-balancing robot

| Physical parameter | Symbol | Value | Unit |
|---|---|---|---|
| Gear ratio | $N$ | $52.734 : 1$ | $\#$ |
| Poles Per Turn | $P$ | 6 | $\#$ |
| Torque Constant | $k_m$ | 0.22 | $\frac{N.m}{A}$ |
| Voltage Constant | $k_e$ | 0.122 | $\frac{V}{\text{rad}/s}$ |
| Total Inertia | $I_m$ | $26.7 \times 10^{-3}$ | $\frac{V}{\text{rad}/s}$ |
| Damping coefficient | $c$ | 0.25 | $\frac{N}{\text{rad}/s}$ |
| Armature Inductance | $L_a$ | 12 | $mH$ |
| Armature Resistance | $R_a$ | 3.3 | $\Omega$ |

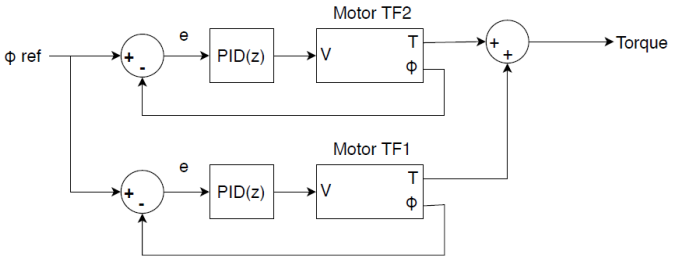



Fig. 5: Schematic representation of the self-balancing robot motor synchronization control between two DC-motors with hall effect quadrature encoder feedback.

### D. Motor Synchronization

Due to nonlinear relation between the left and right motors, forward and backward driving may cause a drift in the angular direction of the self-balancing robot. Additionally, to increase the response time of the motors, a closed loop configuration is preferred. The motor is a DC-motor with input voltage $V_a$ and rotational speed $\omega$ and wheel angle $\varphi_{\text{wheel}}$. The motor transfer function is defined as follows:

$$\frac{\omega(s)}{V_a(s)} = \frac{\frac{k_m}{L_a I_m}}{(s + \frac{R_a}{L_a})(s + \frac{c}{I_m}) + (\frac{k_b k_m}{L_a I_m})} \tag{9}$$

The motor parameters are listed in Table II. The values were determined experimentally and according to the data sheet. Calculation of the wheel angle $\varphi_{\text{wheel}}$, encoder input $x_{\text{enc}}$ and $N$ rotations is done as follows:

$$\varphi_{\text{wheel}} = x_{\text{enc}} \frac{2\pi}{NP} \tag{10}$$

The discrete PID controller, using a sample time $t_{\text{sample}}$ of $0.01s$ is created. Special attention is shown to the avoidance of motor voltage saturation. Also rapid switching of motor direction may cause vibrations causing further disturbance in the sensors. The motor synchronization control diagram of the self-balancing robot and its respective PID controller values depicted in Fig. 5 and Table III.

TABLE III: Motor synchronizing PD-controller parameters

| Physical parameter | Symbol | Value |
|---|---|---|
| Proportional gain | $k_p$ | 6.5 |
| Derivative gain | $k_d$ | 0.4 |

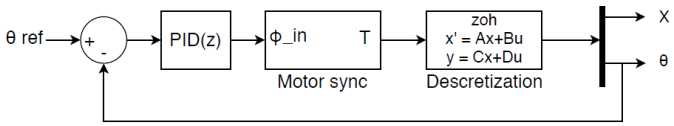



Fig. 6: Schematic representation of the self-balancing robot balancing using PID control.

TABLE IV: Balancing PID controller parameters

| Physical parameter | Symbol | Value |
|---|---|---|
| Proportional gain | $k_p$ | 4.76 |
| Integral gain | $k_i$ | 76.67 |
| Derivative gain | $k_d$ | 0.074 |

### *E. Balancing Control*

The two most common ways of balancing a self-balancing robot are PID control and LQR control. This subsection describes the design of both controllers.

*1) PID Control:* The discrete PID controller, using a sample time $t_{\text{sample}}$ of $0.01s$ is tuned around the initial condition of $\theta = 0°$ using the Simulink PID Tuner App with a Phase margin of $60°$ and Bandwidth of $32$ rad$/s$. The balancing PID control diagram of the self-balancing robot and its respective PID controller values are represented in Fig. 6 and Table IV.

*2) LQR Control:* The LQR controller is created using Matlab with [17], the weighting matrices **Q** and **R** are as:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = 1 \tag{11}$$

Computation in Matlab for feedback gain matrix **K** results in:

$$\mathbf{K} = \begin{bmatrix} 1.00 & 67.5 & 0.957 & 13.8 \end{bmatrix} \tag{12}$$

The balancing LQR control diagram of the self-balancing robot is depicted in Fig. 7.

## IV. Simulation Results

Using the created dynamical model and the designed control models, two simulations, with each respective controller, PID and LQR are done. The simulations of self-balancing robot with a PID and LQR Controller are depicted in Fig. 8 and Fig. 9 respectively.

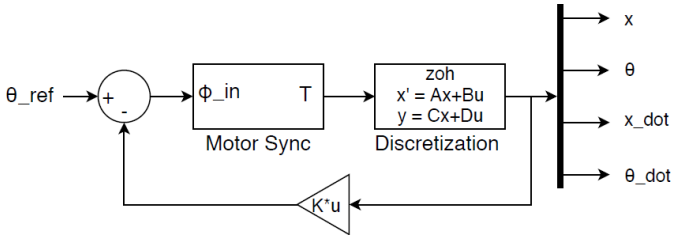



Fig. 7: Schematic representation of the self-balancing robot balancing using LQR control.

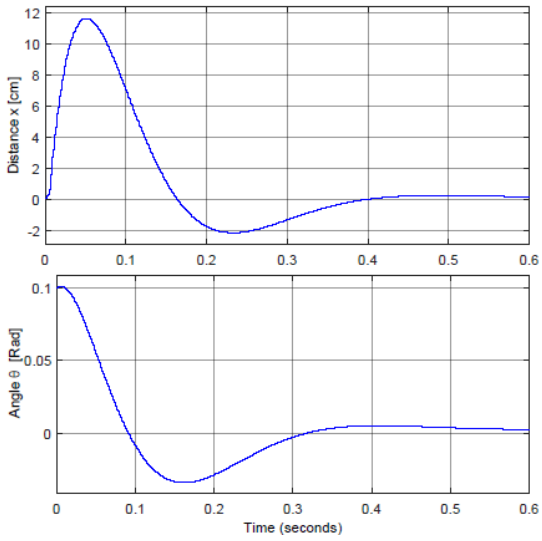



Fig. 8: Simulation results of the self-balancing robot balancing using PID Control from an initial state of $\theta = 0.1$ rad.

The bottom plots in Fig. 8 and Fig. 9 show that the self-balancing robot stabilizes from a $-0.1$ rad to a $0$ rad equilibrium. The top plots of Fig. 8 and Fig. 9 show the traveled distance of the self-balancing robot with each controller during the stabilizing movement. It's observed that with the LQR controller, the self-balancing robot exhibits a drift during the initial balancing correction movement, which slowly reduces towards $0m$ over time.

## V. Deployment on Arduino

This section describes the implementation of the control system on the Arduino Micro-controller. For its application, the self-balancing robot is created by using the primarily graphical-based Simulink environment. The models use discretized zero-order hold conversions of the continuous-time equivalents to allow the cheap hardware to follow a constant fundamental sampling time which is determined automatically.

### *A. Sensor, motor-controller, Bluetooth interfaces*

The default Simulink library combined with additional functionality blocks from externally provided add-ons cover a large amount of control-model implementations and hardware interfacing for the Arduino board platform. However not all low-level hardware or interfacing options are provided in the default Simulink add-on libraries and a block-based or Matlab-based function may cause unnecessary complexity. S-functions are used to carry out low-level hardware functions that are directly written in C or C++.

Subsequently for the self-balancing robot, three S-functions are used. Firstly, to access low-level hardware features such as the hardware-based interrupts for the two two-channel magnetic quadrature encoders, an S-function was written which allows for discrete, sampling-time independent sample collection. Secondly, an S-function is used to interface the self-balancing robot to the RN-42 Bluetooth module. Finally,

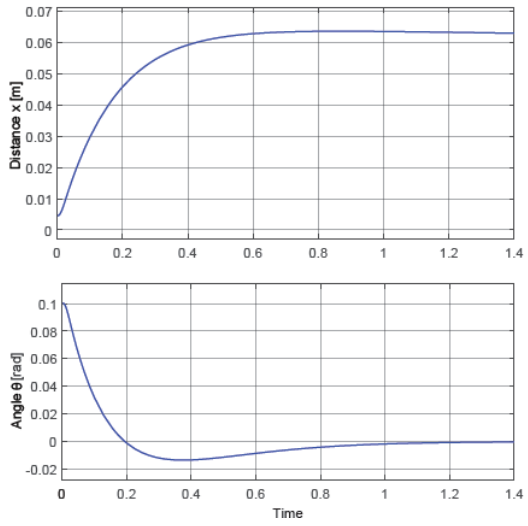



Fig. 9: Simulation results of the self-balancing robot balancing using LQR Control from an initial state of $\theta = 0.1$ rad.

an S-function was written to allow interfacing with the 9-DOF IMU MPU-9250 by utilizing a set of libraries for connection. The sample rate of the S-functions is set to $100Hz$. The implemented sensors and their respective s-functions of the self-balancing robot are depicted in Fig. 10.

### B. Controller

The implemented PID controller and LQR Controller of the self-balancing robot are exhibited in Fig. 11a and Fig. 11b respectively. The controllers are clamped to a $\pm 6V$ voltage, converted to a pulse-width modulation (PWM) range between 0 and 255 with the sign as direction and sent the Arduino DUE PWM and digital output pins.

### C. Code generation

The S-function code generation is done by using the MinGW-w64 C/C++ Compiler as implemented in the Matlab environment. The target language is set to C using the Arduino ARM tool chain. A restriction of using the Arduino ARM tool chain is that it does not support direct discrete sample rate transitions between different controller sample rates, increasing the difficulty of a cascaded control loop with different controller bandwidths. However, the speed of the ARM Cortex-M3 in the Arduino DUE allows fundamental sample rates higher than $200Hz$ to be possible with the Arduino DUE and should not affect the implementation.

## VI. Discussion and Conclusion

The simulations show that the self-balancing robot is able to virtually balance, practical implemented behavior shows increased drift and high sensitivity to sensor noise causing instability after several seconds. It's thus required that an iterative verification step is made to confirm all used constants for reliability of the simulations and design procedure. The model-based design method requires verification of the model when results are non-optimal. Future improvements, next to verification steps, can be improving the sample rate chain inside the cascaded control loop, improved sensing and a faster motor control.

The Arduino Due proves to be capable of running cascaded control loops and interfacing generated by Simulink and therefore opens up possibilities for cheap cyber-physical systems. Real-time control parameter tuning, as implemented in the Bluetooth interface block may, however not the goal of this paper, allow for extra tuning of the PID controller. The model-based approach and code generation allows for a low barrier in controller design inside micro-controllers.

In summary, model-based design is a method with which using a cheap micro-controller board and sensors, a self-balancing robot can be created through modeling, simulation and implementation. Testing a control system on a real-world application is often complicated because of the additional hardware and programming knowledge required. By using an existing platform and Simulink to program the hardware, this process is far more simplified and shortens the time from idea to a working prototype. It is concluded that using the model-based design approach for model creation, simulation and implementation on cheap hardware such as the Arduino allows for a more educational experience in a control oriented curriculum. Further use of model-based design on Arduino or their alternatives can provide a better hands-on experience.

## Acknowledgments

This work was supported by DEMCON, Fontys University of Applied Sciences and Fontys Center of Expertise HTSM.

## References

[1] J. Morantes, D. Espitia, O. Morales, R. Jiménez, and O. Aviles, "Control system for a segway," *International Journal of Applied Engineering Research*, vol. 13, no. 18, pp. 13 767–13 771, 2018.

[2] E. Najafi, G. A. Lopes, and R. Babuska, "Balancing a legged robot using state-dependent Riccati equation control," in *Proceedings of the 19th IFAC World Congress*, 2014, pp. 2177–2182.

[3] H. Yu, Y. Liu, and T. Yang, "Closed-loop tracking control of a pendulum-driven cart-pole underactuated system," *Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 222, no. 2, pp. 109–125, 2008.

[4] E. Najafi, R. Babuska, and G. A. Lopes, "An application of sequential composition control to cooperative systems," in *Proceedings of the 10th International Workshop on Robot Motion and Control*, 2015, pp. 15–20.

[5] G. A. Lopes, E. Najafi, S. P. Nageshrao, and R. Babuska, "Learning complex behaviors via sequential composition and passivity-based control," in *Handling Uncertainty and Networked Structure in Robot Control*, L. Busoniu and L. Tamas Eds. Springer, 2015, pp. 53–74.

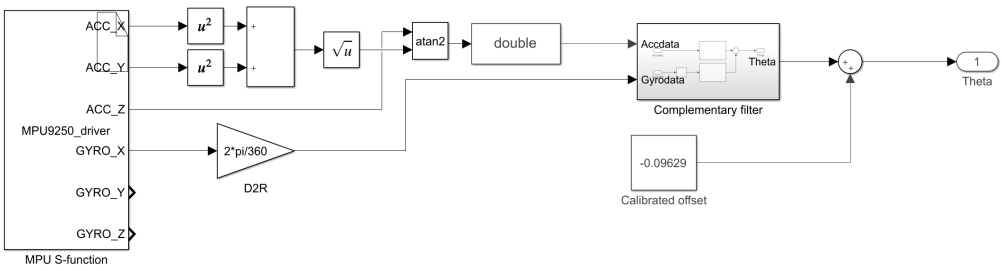
[6] F. Dai, X. Gao, S. Jiang, W. Guo, and Y. Liu, "A two-wheeled inverted pendulum robot with friction compensation," *Mechatronics*, vol. 30, pp. 116–125, 2015.

[7] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, and A. Andreev, "Automatic code generation from Matlab/Simulink for critical applications," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 2014, pp. 1–6.

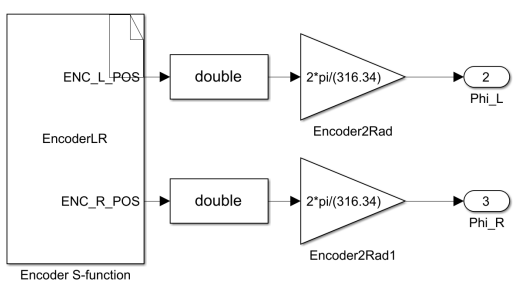
[8] R. Grepl, "Real-time control prototyping in Matlab/Simulink: Review of tools for research and education in mechatronics," in *Proceedings of the IEEE International Conference on Mechatronics*, 2011, pp. 881–886.

[9] E. Najafi, R. Babuska, and G. A. Lopes, "Learning sequential composition control," *IEEE transactions on cybernetics*, vol. 46, no. 11, pp. 2559–2569, 2016.

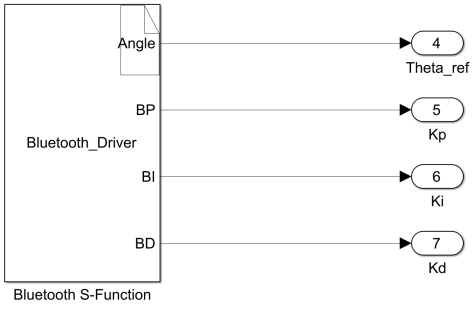
[10] P. Reguera, D. García, M. Domínguez, M. Prada, and S. Alonso, "A low-cost open source hardware in control education. case study: Arduino-Feedback MS-150," *IFAC-PapersOnLine*, vol. 48, no. 29, pp. 117–122, 2015.




(a)

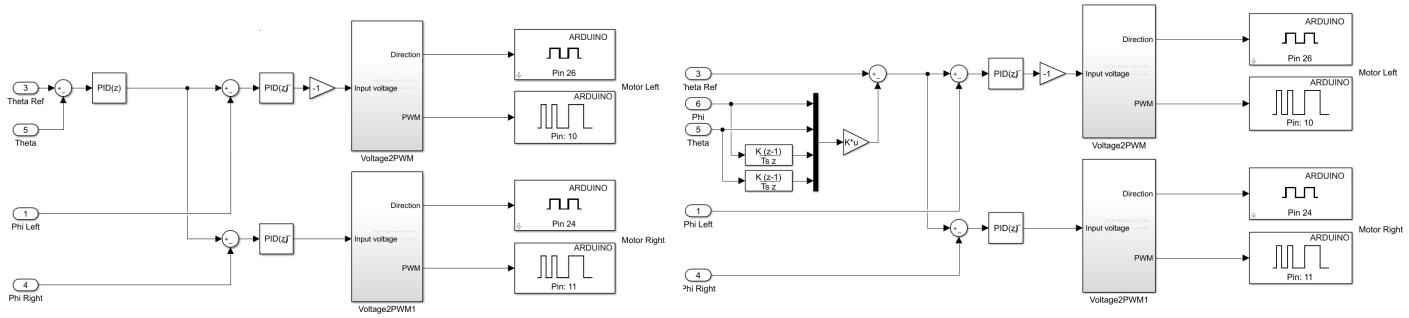



(b)




(c)

Fig. 10: (a) Implemented angle S-function and complementary filter for $\theta$. (b) Implemented hall effect encoder S-function for $\phi_{\text{wheel}}$. (c) Implemented Bluetooth S-function for $\theta_{\text{ref}}$ and PID tuning parameters.




(a) (b)

Fig. 11: (a) Implemented self-balancing robot PID Controller with Arduino Hardware outputs and voltage to PWM conversion. (b) Implemented self-balancing robot PID Controller with Arduino Hardware outputs and voltage to PWM conversion.


[11] E. V. Kumar and J. Jerome, "Robust LQR controller design for stabilizing and trajectory tracking of inverted pendulum," *Procedia Engineering*, vol. 64, pp. 169–178, 2013.

[12] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *Proceedings of the International Wireless Communications and Mobile Computing Conference*, 2011, pp. 1666–1671.

[13] E. Najafi, A. Shah, and G. A. Lopes, "Robot contact language for manipulation planning," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 3, pp. 1171–1181, 2018.

[14] E. Najafi, T. Laugs, P. N. Rubio, and S. Alers, "An implementation of AutomationML for an Industry 4.0 case study: RoboCup@Work," in *Proceedings of the International Conference on Research and Education in Mechatronics*, 2019, pp. 1–6.

[15] J. L. Guzman, R. Costa-Castello, S. Dormido, and M. Berenguel, "An interactivity-based methodology to support control education: How to teach and learn using simple interactive tools," *IEEE Control Systems*, vol. 36, no. 1, pp. 63–76, 2016.

[16] A. Martyanov, E. Solomin, and D. Korobatov, "Development of control algorithms in matlab/simulink," *Procedia engineering*, vol. 129, pp. 922–926, 2015.

[17] Y. Ding, J. Gafford, and M. Kunio, "Modeling, simulation and fabrication of a balancing robot," *Harvard University, Massachusettes Institute of Technology*, vol. 151, 2012.

[18] E. Najafi and G. A. Lopes, "Towards cooperative sequential composition control," in *Proceedings of the 55th IEEE International Conference on Decision and Control*, 2016.

[19] N. H. Ariffin, N. Arsad, and B. Bais, "Low cost mems gyroscope and accelerometer implementation without Kalman filter for angle estimation," in *Proceedings of the International Conference on Advances in Electrical, Electronic and Systems Engineering*, 2016, pp. 77–82.

[20] E. Najafi, G. A. Lopes, S. P. Nageshrao, and R. Babuska, "Rapid learning in sequential composition control," in *Proceedings of the 53rd IEEE International Conference on Decision and Control*, 2014, pp. 5171–5176.

[21] H. G. Min and E. T. Jeung, "Complementary filter design for angle estimation using mems accelerometer and gyroscope," *Department of Control and Instrumentation, Changwon National University, Changwon, Korea*, pp. 641–773, 2015.