STUDY ON THE MECHATRONIC SYSTEM WITH SELF-BALANCE

POPESCU Daniel-Mircea
Faculty: IER, Major: MSMMS, Year: VI, e-mail: daniel_mircea_popescu@yahoo.de

Scientific leader: Assoc. Prof. Ph.D. Eng. **Liviu-Marian UNGUREANU**

ABSTRACT: This paper aims to automate processes in MATLAB using a script that measures motor speed, issuing PWM commands to characterize the response of a 100:1 DC gearmotor, using code sections to partition scripts into smaller parts, using for loops to repeat blocks of code, using text tags and comments to organize and document matlab scripts, and creating and calling a MATLAB function.

KEYWORDS: mechatronics, matlab, component parts, equations, motorcycle.

1. Introduction

The project presents the modeling of a motorcycle that will have a two-wheeled robot that can balance and move using a rotating disc, which we will call the flywheel from now on, to compensate for the motorcycle's loss of balance. The motorcycle is controlled by an Arduino Nano 33 IoT module, an Arduino Nano Motor Carrier module, a DC motor to drive the rear wheel, a DC motor with encoder to control the flywheel, and a standard servo motor to steer the motorcycle's handlebars.

The objectives of this project are to: show how to program the motorcycle with Simulink, the control of the balance algorithm when moving in a straight line, and the improvement of the quality of the control algorithms through rapid prototyping.

2. The current stage

Building and programming a motorcycle model that balances and steers itself using a flywheel, using the knowledge gained in the courses, of physics, control algorithms and simulating the general behavior of a vehicle.

Before starting modeling and working on the project, the motorcycle must be assembled from the mechanical and electronic components. The estimated time for this operation is about 45 minutes.

For this project, the following simulation environments are used:

– *Matlab* combines a desktop environment set for analysis and iterative design processes with a programming language that directly expresses the mathematics of matrices and arrays. Includes the Live Editor for creating scripts that combine code, output, and formatted text into an executable notepad [7];
– *Simulink* is an environment for multi-domain simulation and model-based design. It supports system-level design, simulation, automatic code generation, and continuous testing and verification of embedded systems [7];
– *Simscape multibody* provides a multibody simulation environment for 3D mechanical systems such as robots. One can model multibody systems using blocks representing bodies, joints, constraints, force elements, and sensors [7];
– *Matlab support package for arduino hardware* is the package through which Matlab can communicate interactively with an Arduino board [7];
– *Simulink support package for arduino hardware* is the set of dedicated Simulink libraries to develop and simulate algorithms that run autonomously on Arduino [7].

3. Modeling the motorcycle and designing a controller for balancing

Modeling the motorcycle and designing a balancing controller aims to: model a dynamic system using knowledge of the underlying physical principles, design a feedback controller and use Simulink for rapid prototyping and controller design.

The motorcycle balancing problem can be thought of as an inverted pendulum control problem. Indeed, looking at a standing motorcycle from behind, it can look like a pendulum rod with a flywheel attached to it.

In Figure 1, the following notable points are found: A is the axis of rotation of the rotational couple between the pendulum rod and the ground. In this context of rotation, the A-axis means the axis passing through A and perpendicular to the plane of the paper. lAD is the length of the pendulum rod, i.e. the size of the motorcycle frame; B is the center of mass of the pendulum rod. We assume that the pendulum rod has a uniform density, which means that point B will be midway between A and D; C is the axis of rotation of the rotational couple between the pendulum rod and the flywheel. The flywheel is driven by a DC motor [1].

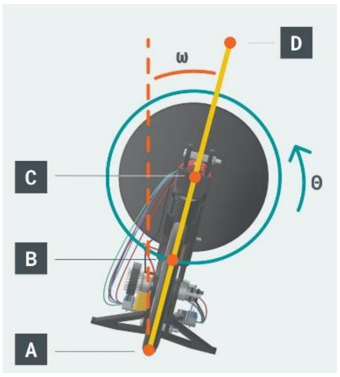


Fig. 1. The inverted pendulum

The considered system has two degrees of freedom and, therefore, the state of the system can be described by two coordinates: the angle of inclination of the pendulum rod θ, which is the angle between the rod and the vertical, so equal to 0° when the pendulum is perfectly vertical, positive when the pendulum swings clockwise and negative when the pendulum swings counterclockwise, the angular displacement of the flywheel. Since the flywheel is axially symmetric, we are not interested in the actual angular displacement but in the rotational speed (ω) and acceleration ($\dot{\omega}$) [1].

$$\text{Newton: } F = m \times a \Rightarrow \tau = I \times \ddot{\theta} \tag{1}$$

$$I = m \times r^2 \tag{2}$$

$$\text{Huygens-Steiner: } l_z = l_{cm} \times a^2$$

$$\tau = \tau_{gr} + \tau_{gw} + \tau_m$$

$$\tau_{gw} = m_w \times g \times l_{AC} \times \sin\theta$$

$$\tau_{gr} = m_r \times g \times l_{AB} \times \sin\theta$$

$$\tau_m = I_w^C \times (\dot{\omega} + \ddot{\theta}) \tag{3}$$

In addition to the torques created by gravity, the torque created by the flywheel motor must be considered. The torque applied to the flywheel and the torque applied to the pendulum rod have the same absolute value but are applied in different directions. We will define this torque as τm and assume that we have control over τ_m [2].

Now the equations for the moments of inertia of the key components are written.

The moment of inertia of the flywheel about point C. The above formula calculates the moment of inertia of a disk rotating about its center of mass [2]:

$$I_w^C = 0{,}5 \times m_w \times R^2 \tag{4}$$

The flywheel rotates about point C being driven by the DC motor, but the same flywheel also rotates about point A along with the pendulum rod. For this reason, the moment of inertia of the flywheel about point A must also be calculated, I_W^A. The previously described parallel axis theorem (Huygens-Steiner theorem) applies to the motorcycle [2]:

$$I_w^A = I_w^C + m_w \times l_{AC}^2 = 0{,}5 \times m_w \times R^2 + m_w \times l_{AC}^2 = m_w(0{,}5 \times R^2 + l_{AC}^2) \tag{5}$$

Continue by calculating the moment of inertia of the pendulum rod about its center of mass B: [2]

$$I_r^B = \frac{1}{12} \times m_r \times (3 \times r^2 + l_{AD}^2) \tag{6}$$

However, the pendulum rod rotates about point A, not its center of mass B; so the parallel axis theorem applies again [2]:

$$I_r^A = I_r^B + m_r \times l_{AB}^2 = \frac{1}{12} \times m_r \times (3 \times r^2 + l_{AD}^2) + m_r \times l_{AB}^2 \tag{7}$$

The equations of motion are written [2]:

$$\tau = I \times \ddot{\theta} \tag{8}$$

Knowing that the values of torques and moments of inertia are additive:

$$\tau_{gr} + \tau_{gw} + \tau_m = (I_w^A + I_r^A) \times \ddot{\theta} \tag{9}$$

Each term develops:

$$\begin{gathered} m_r \times \mathrm{g} \times l_{AB} \times \sin(\theta) + m_w \times \mathrm{g} \times l_{AC} \times \sin(\theta) - \tau_m = \\ m_w(0{,}5 \times R^2 + l_{AC}^2) + \frac{1}{12} \times m_r \times (3 \times r^2 + l_{AD}^2) + m_r \times l_{AB}^2) \times \ddot{\theta} \end{gathered} \tag{10}$$

The rotation of the flywheel is described by the following equations of motion:

$$\tau_m = I_w^C \times (\dot{\omega} + \ddot{\theta}) \tag{11}$$

$$\tau_m = 0{,}5 \times m_w \times R^2 \times (\dot{\omega} + \ddot{\theta}) \tag{12}$$

The next step is to bring the above equations of motion into state-space format. A state vector is first defined [2]:

$$x(t) = \begin{pmatrix} \theta \\ \dot{\theta} \\ \omega \end{pmatrix} \tag{13}$$

The derivative of the state vector with respect to time has the expression:

$$\dot{x}(t) = \begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \dot{\theta} \\ \dfrac{g \times \sin(\theta) \times (m_r \times l_{AB} + m_w \times l_{AC}) - \tau_m}{m_w \times (0{,}5 \times R^2 + l_{AC}^2) + \dfrac{1}{12} \times m_r \times (3 \times r^2 + l_{AD}^2) + m_r \times l_{AB}^2} \\ {}^{\tau_m}/_{I_w^C} - \ddot{\theta} \end{pmatrix} \tag{14}$$

The next step is to implement the above equation in software, to allow the simulation of the behavior of this dynamic system and the design of a feedback controller.

These data are entered into *Matlab*, then the *Simulink* work environment is used which uses numerical approximation methods to evaluate them with finite precision. *Simulink* can use several different numerical integration methods to calculate the output of the block, each with advantages in certain applications. Using the Solver pane in the Configuration Parameters dialog box (Solver Pane) one can select the most suitable technique for a particular application.

Other block diagram elements: Multiplexing and demultiplexing blocks for efficient vector manipulation. In this case, the state vector consists of three elements, see the equation above; Purpose block for viewing signals; Constant signal sources, which represent motor torque; The labels associated with the signals represent the names given to them by the user.

Note that variables from the Matlab workspace are used to define the parameters of the Simulink model. The resulting model is presented in figure 2.

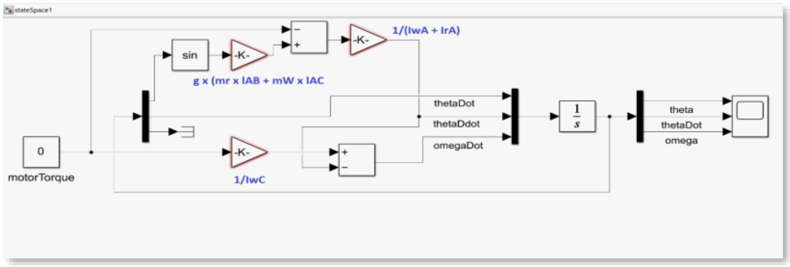
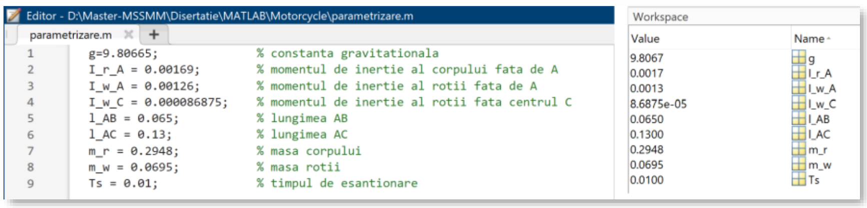


Fig. 2. Model in Simulink simulation environment



Editor - D:\Master-MSSMM\Disertatie\MATLAB\Motorcycle\parametrizare.m

parametrizare.m

```
g=9.80665;                % constanta gravitationala
I_r_A = 0.00169;          % momentul de inertie al corpului fata de A
I_w_A = 0.00126;          % momentul de inertie al rotii fata de A
I_w_C = 0.000086875;      % momentul de inertie al rotii fata centrul C
l_AB = 0.065;             % lungimea AB
l_AC = 0.13;              % lungimea AC
m_r = 0.2948;             % masa corpului
m_w = 0.0695;             % masa rotii
Ts = 0.01;                % timpul de esantionare
```

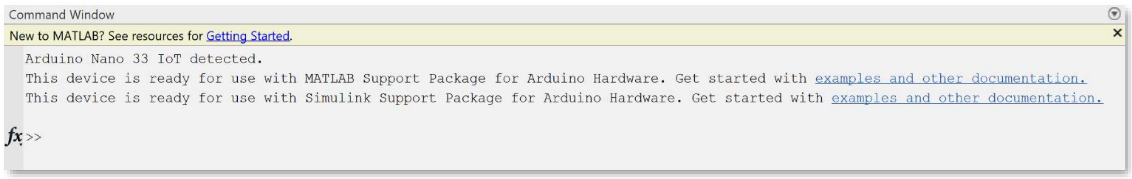
Workspace

Value	Name
9.8067	g
0.0017	I_r_A
0.0013	I_w_A
8.6875e-05	I_w_C
0.0650	l_AB
0.1300	l_AC
0.2948	m_r
0.0695	m_w
0.0100	Ts

Fig. 3. Parameterization of the model

We review the key hardware components of the motorcycle, sensors and actuators, and we will create Simulink models that can interface with the sensors and actuators in order to control the motorcycle.

A sensor is a device that detects changes in the environment. Information about the environment that comes from the sensors is used to change this environment in the way that is desired. The implementation of these changes is done by means of another class of devices, known as actuators [5].



Command Window

New to MATLAB? See resources for Getting Started.

```
Arduino Nano 33 IoT detected.
This device is ready for use with MATLAB Support Package for Arduino Hardware. Get started with examples and other documentation.
This device is ready for use with Simulink Support Package for Arduino Hardware. Get started with examples and other documentation.

fx >>
```

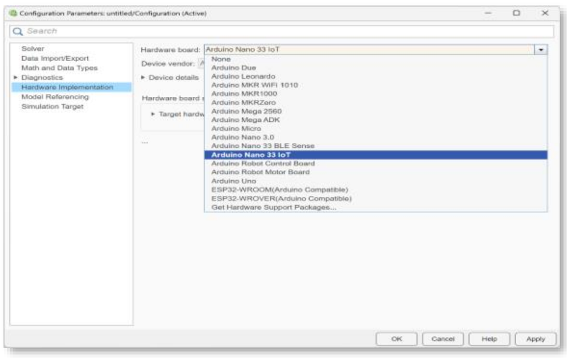
Fig. 4. Connect compatible drivers

My motorcycle has sensors and actuators. *Sensors:* accelerometer/gyroscope/magnetometer for measuring the angle of inclination of the motorcycle; rotary encoder, mounted on the flywheel motor, to measure the speed of the flywheel; rotary encoder, mounted on the rear wheel motor, for measuring the speed of the rear wheel; battery voltage sensor, which is not used for balancing, but could shut down the controller before it runs out of power.

Actuators: flywheel motor to drive the flywheel; rear wheel motor, to drive the rear wheel; servomotor for steering control [5].

Due to the inclusion of Arduino software libraries in the latest editions of Matlab, the program recognizes a wide range of Arduino boards, including shields, and connects to them via compatible drivers [6].

In Simulink we create a template that will be used to work with the ArduinoNano33IoT and test the connection:


Configuration Parameters: untitled/Configuration (Active)

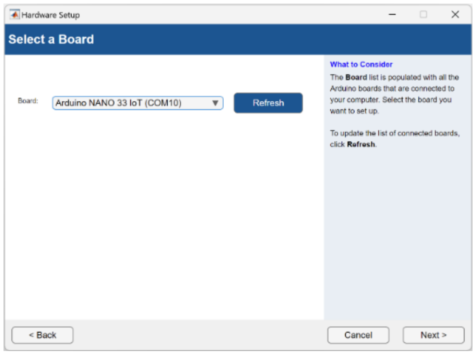
Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target

Hardware board: Arduino Nano 33 IoT
Device vendor:
Device details
Hardware board
Target hardw
...

None
Arduino Due
Arduino Leonardo
Arduino MKR WiFi 1010
Arduino MKR1000
Arduino MKRZero
Arduino Mega 2560
Arduino Mega ADK
Arduino Micro
Arduino Nano 3.0
Arduino Nano 33 BLE Sense
Arduino Nano 33 IoT
Arduino Robot Control Board
Arduino Robot Motor Board
Arduino Uno
ESP32-WROOM(Arduino Compatible)
ESP32-WROVER(Arduino Compatible)
Get Hardware Support Packages...

Fig. 5. ArduinoNano33IoT template


Hardware Setup

Select a Board

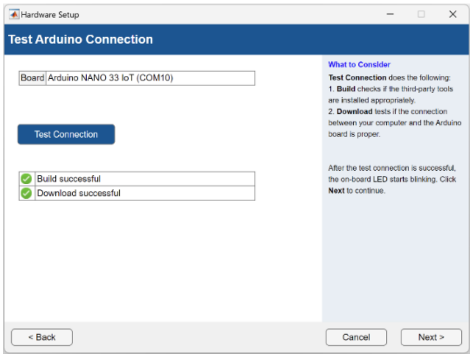
Board: Arduino NANO 33 IoT (COM10)

What to Consider

The **Board** list is populated with all the Arduino boards that are connected to your computer. Select the board you want to set up.

To update the list of connected boards, click **Refresh**.

Fig. 6. Testing the connection – step 1


Hardware Setup

Test Arduino Connection

Board Arduino NANO 33 IoT (COM10)

✓ Build successful
✓ Download successful

What to Consider

Test Connection does the following:
1. **Build** checks if the third-party tools are installed appropriately.
2. **Download** tests if the connection between your computer and the Arduino board is proper.

After the test connection is successful, the on-board LED starts blinking. Click **Next** to continue.

Fig. 7. Test the connection – step 2

This template will later be populated with project components.

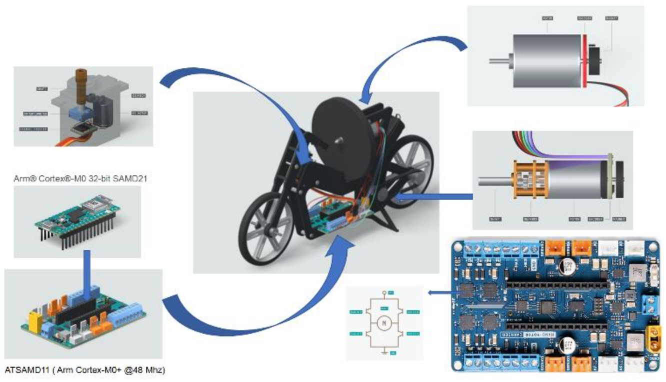

Arm® Cortex®-M0 32-bit SAMD21
ATSAMD11 (Arm Cortex-M0+ @48 Mhz)

Fig. 8. The component parts of the motorcycle

In closed-loop control (stationary balance) the designed controller is combined to implement a complete balance control application that will run on the Arduino Nano 33 IoT board.

This allows to test the algorithm on the motorcycle and to adjust it, taking into account the imprecision of the model and the specific characteristics of the hardware. By the end of this exercise, you will have a motorcycle that balances on the spot.

It is followed as follows: the interfacing of hardware sensors and motors with the balance control algorithm; the implementation of safety features to ensure the operation of the motorcycle with minimal stress on the hardware components; monitoring the physical response of the motorcycle to the control algorithm; adjusting the PD algorithm and gains so that the bike balances on the spot.

To create a high-level architecture, the model will be structured using subsystems from the beginning. It starts from a template model that was created previously: Arduino33IotNanoTemplate.slx. Open the model and add two Subsystem blocks, which will represent the motorcycle and a digital controller, respectively. Subsystem inputs and outputs are renamed to Import and Outport blocks inside each subsystem [6].

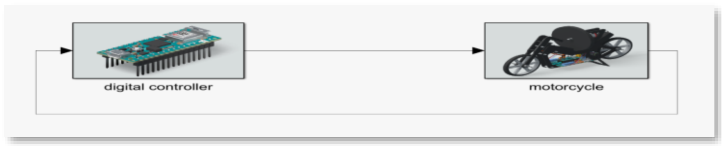


Fig. 9. Charts of both subsystems

To add hardware components (sensors and actuators) go into the motorcycle subsystem, add relevant components and test all these components separately.

To build the safety logic, following the related steps (sensor description, comparisons, reprocessing of sensor measurements, etc.), we have all the elements of the state vector: the angle of the motorcycle, the first derivative of the angle (i.e. speed) and the speed of the flywheel.

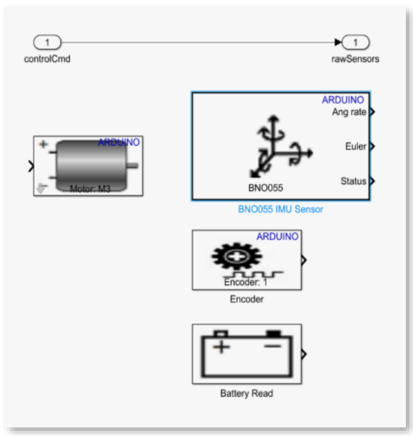


Fig. 10. Motorcycle subsystem - components needed to balance the motorcycle

In the ideal world, we'd simply reuse the controller we've already designed to balance the bike in the simulation. However, in real life, there are several considerations to take into account: IMU (Inertial Measurement Unit) calibration; battery level, loss of balance, activation of user control.

Logic control can be implemented with a dedicated Simulink > Stateflow library and adding a Chart block from the Stateflow library in the Library Browser.

The control logic implementation for handling loss of balance starts with the fact that if the motorcycle loses balance (if the lean angle is greater than a certain threshold), the controller must be disabled. One can formalize this requirement in relation to the Stateflow diagram as follows: the diagram accepts a single input, the tilt angle; chart returns a single output equal to 1 (true) when the tilt angle is within certain limits and equal to 0 (false) when the tilt angle is outside the given limits.

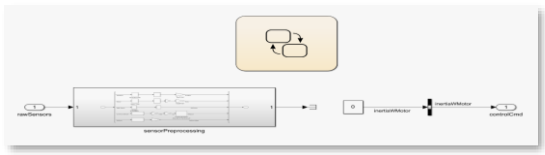


Fig. 11. Simulink dedicated library

Control logic can be implemented in various ways. We will consider the key components of the model below. The first thing you might notice are the user interface components, namely a Switch and a Lamp, both available in Simulink > Dashboard. The switch allows the user to manually disable the control when not needed. The lamp has no effect on the behavior of the model but visualizes the output of the Stateflow diagram. I also added input signals to the Stateflow diagram.

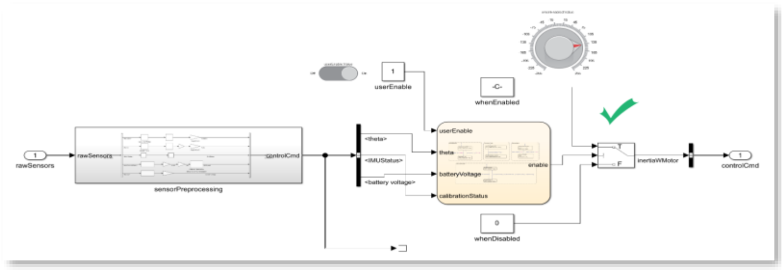


Fig. 12. Stateflow Simulink diagram

The chart itself has four parallel states, which can be identified by dashed frames:

- ✓ checkBalanced: is characterized by two states: balanced_notOK (default) and balanced_OK. The transition between the two states is made according to the theta angle: if the absolute value of the angle is less than 15°, the transition is made to the balanced_OK state. If the angle is outside this range of values, the transition is made to the balanced_notOK state.

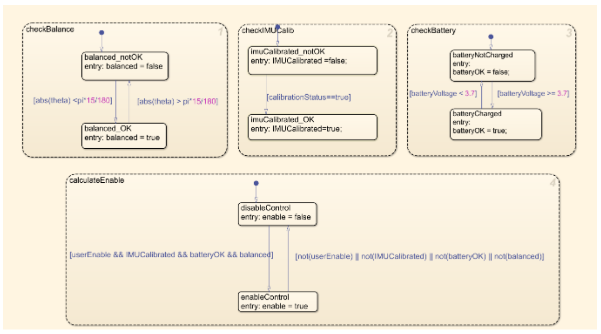


Fig. 13 Diagram with broken frames

- ✓ checkIMUCalib: The two states are: imuNotCalibrated (default) and imuCalibrated. transition to the imuCalibrated state is made when the calibrationStatus signal becomes true or 1.
- ✓ checkBaterry: Statuses indicate whether the battery is usable or discharged. The transition is made in both directions depending on the battery voltage. The switching threshold is 3.7 volts.
- ✓ calculateEnable: The activation of the control depends on the simultaneous fulfillment of the previous conditions. That is why the AND logical operator is used: userEnable && batteryOK && IMUCalibrated && balanced.

For the control to transition to the disabled state, at least one of the conditions must not be met and the logical OR operator is used: Not(userEnable) || not(batteryOK) || not (IMUCalibrated) || not (balanced).

When calculateEnable is true, the signal to the flywheel motor is enabled. The control lamp sign turns green, indicating that all the conditions in the diagram are met.

Although the parallel states run concurrently, it must be determined when to activate each during the simulation.

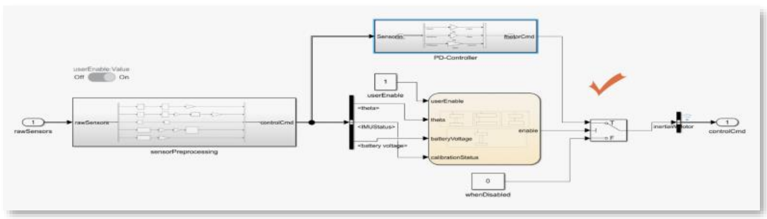


Fig. 14. PDP controller subsystem

The last step in the design workflow is to implement a feedback controller, which will calculate the control action based on the sensor measurements. The feedback control law I designed for my Simscape model consists of three components: feedback from the angle of inclination of the pendulum (motorcycle); feedback from the derivative of the pendulum tilt angle (angular velocity); feedback from the flywheel speed. Finally, the PDP controller subsystem looks like in figure 14.

Since the research work is very complex, the stages completed can only be presented in detail in the form of the final results.

4. Conclusions

In this paper, I wanted to present some of my research on self-balancing in motorcycles by which the mechatronic system balances itself, and the connection to Simulink (multi-domain simulation environment and model-based design) can be changed in real time, the parameters of SET.

"Study on the mechatronic system with self-balance" is a small part of a complex research study done by me, and this paper represents a first part of the analysis and implementation of a feedback controller and other components related to center of gravity balancing of a motorcycle and its demonstration through simulation programs such as *Matlab*, *Simulink*, *Simscape Multibody*, *Simulink Support Package for Arduino Hardware*, etc.

5. References

[1]. Comănescu, A., Comănescu, D., Ungureanu, L., Boblea, D. and Boblea, L. (2021), *Mecanisme. Analiza și sinteza, modelarea, simularea și optimizarea sistemelor mecanice*, Editura Politehnica Press, București, ISBN 978-606-515-948-8;
[2]. Moise, V., Maican, Ed. and Moise, Ș.I. (2016), *Metode numerice. Aplicații în Matlab*, Editura Printech, București, ISBN 978-606-23-0625-0;
[3]. Moise, V., Simionescu, I., Ene, M. and Rotaru, A. (2015), *Analiza mecanismelor plane cu bare articulate. Aplicații în Matlab*, Editura Printech, București;
[4]. Moise, V., Simionescu, I. and Ene, M. (2018), *Sinteza optimală a mecanismelor cu came plane - Aplicații în Matlab*, Editura Printech, București, ISBN 978-606-23-0895-7;
[5]. *** *Scopul și principiul de funcționare a principalilor senzori*, https://ro.avtotachki.com/naznachenie-i-princip-raboty-osnovnyh-datchikov-akpp/, 2023;
[6]. *** Arduino, https://www.arduino.cc/, 2023;
[7]. *** MathWorks Inc., https://www.mathworks.com/, 2023;
[8]. *** Bosch Sensortec GmbH, fișă post, 2023.