DigitalMotorSim

Manual

V1.0 3 November 2024

This Python program simulates a DC motor controller with visual feedback.

Key Functionalities

Controller Modes: Allows choosing between FDE-based control or simpler PID control.

Collision Indicators: Visual feedback when Gargamel or the Smurf is "hit."

Saturation Checks: Simulates realistic limits for voltage and current.

Real-Time Graphics: Displays motor behavior with updated voltage, current, and error indicators.

This setup provides a realistic and interactive simulation for experimenting with motor control dynamics.

Here’s a brief overview of its workflow and key components:

Overview

Library Imports:

numpy: For mathematical functions.

sys: For program control (exiting on ESC).

pygame: Manages the graphical interface.

importlib: Dynamically loads motor modules based on the settings file.

ast: Safely evaluates strings as Python literals to parse settings values.

Settings Loader:

The function load_settings reads settings.txt, which contains the configuration for motor models, control parameters, and graphical settings. Non-comment, non-blank lines are parsed, handling integers, floats, and lists.

Motor Module Import:

The motor model specified in settings.txt (e.g., "MaxonEC60flat") is imported as a Python module. This module contains motor parameters, available through the motor's datasheet.

IMPORTANT: the values should be in SI units.

Example (from the datasheet of Maxon EC 60 flat, brushless, 100W, part no. 645604):

NOMINAL_VOLTAGE = 24.0 # V

NO_LOAD_SPEED = 4300.0 # rpm

NO_LOAD_SPEED *= 2.0*pi/60 # convert rpm into rad/sec

NO_LOAD_CURRENT = 0.493 # A

NOMINAL_SPEED = 3730 # rpm

NOMINAL_SPEED *= 2*pi/60 # rad/sec

NOMINAL_TORQUE = 0.272 # Nm (max continuous torque)

NOMINAL_CURRENT = 5.18 # A (max continuous current)

STALL_TORQUE = 2.51 # Nm

STALL_CURRENT = 83.2 # A

MAX_EFFICIENCY = 0.832

TERMINAL_RESISTANCE = 0.288 # Ohm

TERMINAL_INDUCTANCE = 0.279e-3 # Henry

TORQUE_CONSTANT = 52.5e-3 # Nm/A

SPEED_CONSTANT = 182 # rpm/V

SPEED_CONSTANT *= 2*pi/60 # convert rpm/V into rad/sec/V

ROTOR_INERTIA = 0.835e-4 # kg * m^2

NOMINAL_POWER = 100

now all units are in SI

Parameter Initialization:

Essential motor and simulation parameters (like target angle, encoder resolution, and control gains) are read from the settings file. Default values are provided if parameters are missing.

Simulation Setup:

Physical parameters, including moment of inertia (J), resistance (R), and inductance (L), are calculated from motor data. Sampling time (Ts) and time steps (dt) are also initialized based on FPS, N_control, and N_physics.

Main Simulation Loop:

The main loop handles events, control computations, physics updates, and graphics.

Event Handling: Captures mouse clicks and ESC key presses. Allows dragging the motor's angle with the mouse.

Controller: If not in mouse-drag mode, the controller updates based on the target angle. Two modes are available:

Finite Difference Equation (FDE) Mode: Applies a custom controller with coefficients ca and cb.

Naive PID Mode: Uses proportional, derivative, and integral gains (Kp, Kd, Ki) to compute control output.

Saturation: Voltage and current outputs are limited if saturation is enabled.

Physics Loop: Calculates motor angle, speed, and current in each physics iteration, applying the control voltage to the motor.

Collision Detection:

Checks if the motor's arm (club) hits the target (Gargamel) or obstacle (Smurf) within a tolerance range.

Graphics Rendering:

Pygame is used to render the simulation elements:

Static Elements: Background, target (Gargamel), and obstacle (Smurf).

Dynamic Elements: Motor angle is displayed as a rotating arm.

Status Displays: Shows the time, voltage, current, and error levels on the screen.

Exit and Cleanup:

The simulation exits cleanly when ESC is pressed or the window is closed.

Settings.txt

General Structure

The settings file defines parameters for simulating digital controllers on DC motors, including motor selection, target and obstacle positions, encoder resolution, and control settings. Comments begin with the # symbol.

Parameters

Motor Model Selection

MOTOR_MODEL: Specifies which motor model to use. Motor datasheets should be placed in the "motors" folder.

Example: MOTOR_MODEL = "MaxonEC60flat"

Target and Obstacle Settings

target_angle: Defines the target angle for the motor in degrees. Positive rotation is counterclockwise (CCW) starting from 0° on the x-axis.

angle_diff: Sets the angular distance in degrees between the target and any obstacle.

tolerance: The tolerance in degrees to consider the target or obstacle as "hit."

Encoder Settings

cpt: Encoder resolution specified as Counts Per Turn (cpt), indicating the number of increments per full rotation.

Digital Controller Settings

Controller Mode:

FDE_mode: If True, the controller uses a Finite Difference Equation (FDE). If False, a naive controller form is used (proportional-integral-derivative).

FDE Mode Parameters:

ca, cb: Lists of coefficients for the FDE-based controller.

Example: ca = [0.3333, 0.6667] and cb = [334.2, -666.7, 332.5]

Naive Mode Parameters:

Kp, Kd, Ki: Proportional, derivative, and integral gains for the naive controller. These are ignored if FDE_mode is True.

Example: Kp = 1.0, Kd = 0.0, Ki = 0.0

Voltage and Current Saturation Checks

CHECK_VOLTAGE_SATURATION: If True, simulates voltage saturation.

CHECK_CURRENT_SATURATION: If True, simulates current saturation.

max_voltage_overshoot: Maximum allowable voltage overshoot.

max_current_overshoot: Maximum allowable current overshoot.

Motor and Simulation Parameters

REDUCTION_RATIO: Gear reduction ratio for the motor.

LOAD_INERTIA: Moment of inertia of the load ($kg \cdot m^2$).

AMPLIFIER: Factor to increase the microcontroller's voltage output.

Timing and Control Loop Settings

FPS: Frames per second for the simulation.

N_control: Number of control loops executed between screen refreshes. The sampling time Ts is calculated as Ts = 1 / (FPS * N_control).

N_physics: Number of physics loop executions per control period. Each physics loop has a time step dt = Ts / N_physics.