DigitalMotorSim.py

Manual

Version: 1.0
Date: 3 November 2024
Author: Dino Accoto (dino.accoto@kuleuven.be)

Introduction

This Python program simulates digital controllers applied to armature-controlled DC motors, providing an interactive environment to experiment with digital control techniques in motor systems.

Key Features

- **Visual Feedback:** Evaluate controller performance in position control tasks.
- **Controller Modes:** Supports custom control laws as finite difference equations (FDEs) and a basic numerical PID controller.
- **Motor Parameter Storage:** Store and load motor parameters from data sheets.
- **Encoder Quantization:** Incorporates encoder resolution (counts per turn).
- **Current and Voltage Saturation:** Set and simulate limits for voltage and current.
- **Real-Time Dashboard:** Monitor position error, power, current, and voltage alongside motor position.

Setup

- **Motor Data Files:** Motor specifications are stored as `.py` files in the `motors` folder.
- **Configuration File:** Edit `settings.txt` to adjust simulation parameters.
- **Execution File:** Run `DigitalMotorSim.py` to start the simulation.

Required Libraries

The program requires Python 3+ with the following libraries:

- **numpy** (for calculations)
- **sys** (for exit controls)
- **pygame** (for graphical interface)
- **importlib** (to load motor modules dynamically)
- **ast** (to safely parse configuration values)

How It Works

Loading Parameters

The function `load_settings` reads parameters from `settings.txt`, which includes motor models, control settings, and display options. Comments in the file are marked with `#`.

Motor Module Import

The motor specified in `settings.txt` (e.g., `MaxonEC60flat`) is imported as a Python module with motor parameters in SI units.

```
Example from Maxon EC 60 flat (100W) Datasheet:
NOMINAL_VOLTAGE = 24.0  # V
NO_LOAD_SPEED = 4300.0 * (2 * pi / 60)  # rad/sec
NO_LOAD_CURRENT = 0.493  # A
# and so on...
```

Parameter Initialization

Essential parameters like target angle, encoder resolution, and control gains are read from `settings.txt`. Default values apply if parameters are missing.

Simulation Setup

Physical properties (e.g., *rotor moment of inertia*, *resistance*, *inductance*, etc.) are read from motor data.

Frame rate (Frames per seconds, FPS) controls visual refresh rate, while control and physics loops set the simulation's timing precision.

- **Sampling Time (Ts):** Calculated as Ts = 1 / (FPS * N_control).
- **Physics Time Step (dt):** Calculated as dt = Ts / N_physics, dictating the time resolution for dynamics calculations.

Main Simulation Loop

The main loop manages event handling, control updates, physics computations, and display.

- **Events:** Handles mouse clicks/drags and exits (ESC key).
- **Controller Modes:**
  - **FDE Mode:** Custom finite difference equations with user-defined coefficients (ca, cb). E.g.:
    `ca = [0.3, 0.6]` and `cb = [33.1, -66.0, 21.2]` describe the following control law:
```
y[k] = 0.3*y[k-1] + 0.6*y[k-2] + 33.1*e[k] - 66.0*e[k-1] + 21.2*e[k-2]
```
    where `y` is the output and `e` the input.
  - **Not FDE Mode (PID Mode):** Proportional, derivative, and integral gains (`Kp, Kd, Ki`) are applied as:
```
y[k] = Kp*e[k] + Kd*(e[k] - e[k-1]) / Ts + Ki*sum(e[:k+1])*Ts
```
  - **Saturation:** Limits voltage and current outputs if enabled.

Configuration (settings.txt)

General Structure

The file defines simulation parameters like motor model, target positions, encoder resolution, and control settings. Comments are prefixed with #.

Only this file (and the file describing the motor, if needed) should be edited to run a simulation.

Key Parameters

- **Motor Model:** `MOTOR_MODEL` specifies the `.py` file describing the motor (e.g., `MaxonEC60flat`).
- **Target Settings:** `target_angle` sets the target motor position in degrees. `angle_diff` and `tolerance` define obstacle spacing and acceptance range.
- **Encoder:** `cpt` (counts per turn) sets the encoder resolution.
- **Controller Settings:**
  - **FDE_mode**: Enables FDE if `True`; otherwise, uses PID.
  - **FDE Parameters:** `ca` and `cb` are coefficient lists for the custom controller.
  - **PID Parameters:** `Kp`, `Kd`, `Ki` set PID gains, used if `FDE_mode` is `False`.
- **Saturation Limits:**
  - `CHECK_VOLTAGE_SATURATION` and `CHECK_CURRENT_SATURATION` enable voltage and current limits.
  - `max_voltage_overshoot` and `max_current_overshoot` define saturation limits.
- **Simulation Timing:** `FPS` sets frame rate; `N_control` specifies how many control loop steps should be executed per frame; and `N_physics` specified how many physics loop steps should be executed per each control loop.

  The time 1/`FPS` between two consecutive frames are thus divided into `N_control steps`, over which a control loop is executed. Therefore, the sampling time (Ts) is

  `Ts = 1/(FPS* N_control steps)`

  Every `Ts` seconds, `N_physics` simulation loops of the physics are executed. Therefore, the time infinitesimal adopted for the numerical solution of the dynamic equation is

  `dt = Ts/N_physics`

  In the physics loop, motor angle, speed, and current in each physics iteration are calculated by numerically solving the control voltage laws to the motor using Euler first order method.