The Journal of Open Source Education

pendsim: Developing, Simulating, and Visualizing Feedback Controlled Inverted Pendulum Dynamics

Mike Sutherland[1] **and David A. Copp**[1]

1 Department of Mechanical and Aerospace Engineering, University of California, Irvine

DOI: 10.21105/jose.00168

Software

- Review
- Repository
- Archive

Submitted: 11 January 2022
Published: 04 February 2023

Summary

This package is a companion tool for exploring dynamics, control, and machine learning for the canonical cart-and-pendulum system. It includes a software simulation of the cart-and-pendulum system, a visualizer tool to create animations of the simulated system, and sample implementations for controllers and state estimators. The user can use any platform or the browser to run the `pendsim` Python package. It gives the user a plug-and-play sandbox to design and analyze controllers for the inverted pendulum, and is compatible with Python's rich landscape of third-party scientific programming and machine learning libraries.

The package is useful for a wide range of curricula, from introductory mechanics to graduate-level control theory. The inverted pendulum is a canonical example in control theory (See, e.g. (Aström & Murray, 2008)). A set of example notebooks provide a starting point for introductory and graduate-level topics.

Statement of need

Curricula in the study of dynamical systems and control can be quite abstract. When a student studies the abstract mathematical model of a system they have difficulty seeing the effects of control and modeling parameters. Because of this, direct experimentation is a natural way to better understand how these systems evolve over time given different controllers and parameters.

Physical laboratory setups are expensive, time-consuming, and limited to four or five students at a time. Virtual experiments are cheap, easy-to-setup, and accomodate hundreds of students at a time. Virtual experiments can augment course content, even for remote-only instruction. The virtual platform allows students to share their work, run experiments collaboratively or individually, and develop controllers or investigate system dynamics in a fast design-test loop.

Instructors can design experiments in `pendsim`, and subsequently measure any system parameter or variable, including the animation of the system. The package includes visualizations and pre-built controllers. The package is a great companion to any control or dynamical systems course material, in either a virtual, hybrid, or in-person context.

Example Usage

The software is a virtual laboratory. Users create an experiment by specifying a set of parameters: the pendulum (mass, length, friction, and so on), and the simulation

parameters (such as external forces). The user can then design and apply a control policy in the simulation. Finally, the user can view the results of the simulation. The ability to rapidly create and run experiments allows for fast design-test loops.

This design-test-visualize sequence also allows instructors to introduce students to new topics and create interactive assignments that can augment theoretical class discussions and hardware experiments. Thus, `pendsim` may be naturally used in engineering courses related to dynamics and control that are taught in any format, including in-person, hybrid, and online settings. An example PID Controller Design Assignment notebook is available to show how students may use `pendsim` in a course assignment.

The following simple example (Example 1) shows the ease of creating, modeling, and visualizing a proportional-integral-derivative controller:

```
### Example 1
# define simulation parameters
dt, t_final = 0.01, 5.0
def forcing_func(t):
    return 10 * np.exp(-(((t - 2.0) / 0.2) ** 2))

# define pendulum parameters
pend = sim.Pendulum(
    2.0,  # Large mass, 2.0 kg
    1.0,  # Small mass, 1.0 kg
    2.0,  # length of arm, 2.0 meter
    # state inputs are stored as numpy arrays:
    # [x, xdot, theta, thetadot]
    initial_state=np.array([0.0, 0.0, 0.1, 0.0]),
)

# PID gains
kp, ki, kd = 50.0, 0.0, 5.0
# controllers are stored in the controller module.
cont = controller.PID((kp, ki, kd))

# create simulation object
simu = sim.Simulation(
    # timestep, simulation time, and forcing function
    dt, t_final, forcing_func,
    # simulate gaussian noise added to state measurements
    noise_scale=np.array([0.05, 0.05, 0.05, 0.05])
)

# run simulation with controller and pendulum objects
results = simu.simulate(pend, cont)

# create an animation of the results
visu = viz.Visualizer(results, pend)
ani = visu.animate()
```

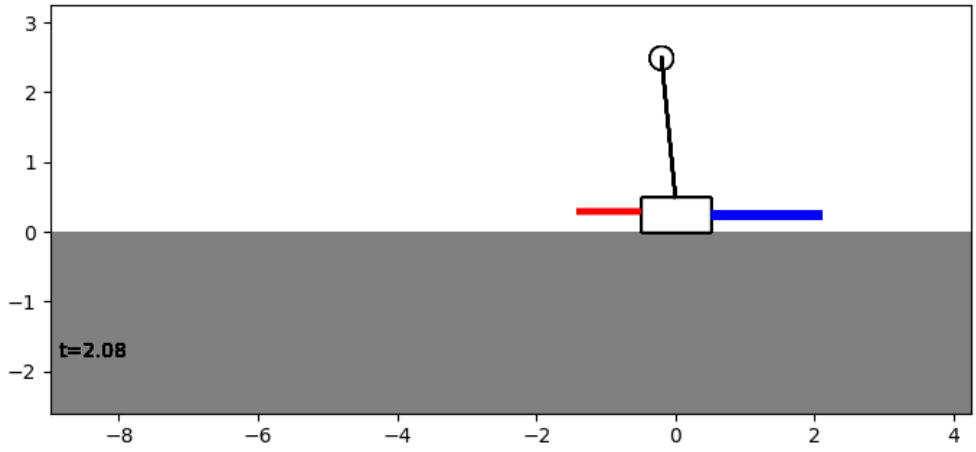

Figure 1: A still from the animation module. Here, a force pushes to the right (shown in red) while the controller pushes to the left to stabilize the pendulum (shown in blue).

Rich plots of any system attribute are easy to generate. This example shows the plot in Figure 2:

```
### Example 2
import matplotlib.pyplot as plt
fig, ax = plt.subplots()

ax.plot(results[("state", "t")])
ax.scatter(
  results.index, results[("measured state", "t")].values,
  color="black", marker=".",
  )
ax.set_ylabel("Angle (Radians)")
ax.set_xlabel("Time (s)")
ax.set_title("Pendulum Angle")
plt.show()
```

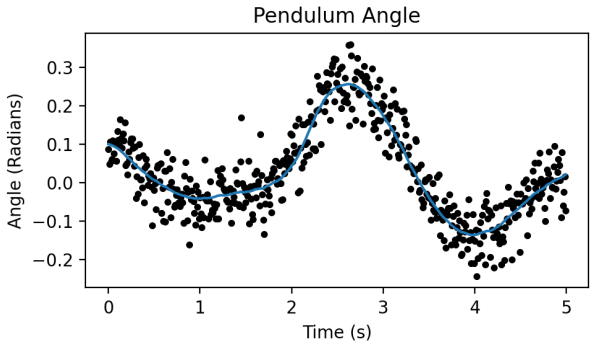

Figure 2: This plot (generated by the code in Example 2) shows the angle of the pendulum over time as it evolves in simulation. The black dots show the measured angle, while the line shows an Unscented Kalman Filter estimation of the angle. Such plots are easy to generate from outputs of the simulation.

Package Features

A core pendulum/simulation module. (`sim.py`)

This simulates the system dynamics and allows for external forces on the pendulum. Users can specify:

- pendulum parameters (e.g., masses, length, friction, etc.)
- a time period over which to simulate
- an external forcing function (e.g., push)
- noise characteristics
- a timestep for the simulation
- a controller to use, if any

Controllers, Estimators, etc. (`controller.py`)

Several controller implementations are pre-built. These include:

- Bang Bang (On-Off) controller
- Proportional-Integral-Derivative (PID) controller
- Linear Quadratic Regulator (LQR) controller
- State estimation using an Unscented Kalman Filter (UKF) (implemented with package `filterpy` (Labbe, 2021))
- Energy Swing-Up Controller

The user can implement any control policy by creating a new custom controller class. Users can test new open-ended controller designs. Controllers can dump data into the simulation results so that intermediate control inputs are accessible to the final results of the simulation.

Visualization (`viz.py`):

The simulation results are visualized in `viz.py`. The 'matplotlib' (Hunter, 2007) backend draws animations of the inverted pendulum and plots from the simulation. The visualization uses the results of the simulation to draw the inverted pendulum, including the external and control forces applied to it. The animation module allows for the system to plot real-time simulation data (e.g., data used by the controller) side by side with the animation.

The results of the simulation are easy to query and plot. This makes investigating the simulation easy and intuitive.

Example Notebooks:

The repository includes several notebooks which show the capabilities of the package. Example notebooks are hosted on Google Colab:

- Animated Plots
- System Linearization
- PID Tuning
- PID Controller Design Assignment
- Applying a state estimator for better control
- Swing-up by Energy Control

References

Aström, K. J., & Murray, R. M. (2008). *Feedback systems*. Princeton University Press. https://doi.org/10.1515/9781400828739

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Labbe, R. (2021). *Filterpy*. https://github.com/rlabbe/filterpy