EECS 367 Lab
Pendularm Support

Administrative

Make sure you have pulled from the upstream!

Next Wednesday, **September 30**:

- **Assignment 2** due at 11:59pm
- **Quiz 2** held on Gradescope between 12:00am-11:59pm

Regrade policy on course website

Lab Takeaways

1. Stencil review
2. Equation translation
3. Implementation advice
4. Big picture

Pendularm Overview

		Assignment 2: Pendularm
4	All	Euler integrator
4	All	Velocity Verlet integrator
4	All	PID control
1	Grad	Verlet integrator
2	Grad	RK4 integrator
3	Grad	Double pendulum

Features assigned to all sections

Features assigned to grad section only

Stencil Review

update_pendulum_state.js

```
1  function update_pendulum_state(numerical_integrator, pendulum, dt, gravity) {
2      // integrate pendulum state forward in time by dt
3      // please use names 'pendulum.angle', 'pendulum.angle_previous', etc. in else codeblock between line 28-30
4
5      if (typeof numerical_integrator === "undefined")
6          numerical_integrator = "none";
7
8      if (numerical_integrator === "euler") {
9
10     // STENCIL: a correct Euler integrator is REQUIRED for assignment
11
12     }
13     else if (numerical_integrator === "verlet") {
14
15     // STENCIL: basic Verlet integration
16
17     }
18     else if (numerical_integrator === "velocity verlet") {
19
20     // STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment
21
22     }
23     else if (numerical_integrator === "runge-kutta") {
24
25     // STENCIL: Runge-Kutta 4 integrator
26     }
27     else {
28         pendulum.angle_previous = pendulum.angle;
29         pendulum.angle = (pendulum.angle+Math.PI/180)%(2*Math.PI);
30         pendulum.angle_dot = (pendulum.angle-pendulum.angle_previous)/dt;
31         numerical_integrator = "none";
32     }
33
34     return pendulum;
35 }
```

Feature stencils

Default rotation

Acceleration Helper Function

update_pendulum_state.js

```
function pendulum_acceleration(pendulum, gravity) {
    // STENCIL: return acceleration(s) system equation(s) of motion

}

```

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta) + \frac{\tau}{ml^2}$$

Euler Equations

update_pendulum_state.js

```
    if (numerical_integrator === "euler") {

    // STENCIL: a                                                   for assignment

    }
    else if (numerical_integrator === "verlet") {

    // STENCIL: basic Verlet integration

    }
    else if (numerical_integrator === "velocity verlet") {

    // STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment

    }
    else if (numerical_integrator === "runge-kutta") {

```

$$\theta(t+dt) = \theta(t) + \dot{\theta}(t)dt$$
$$\dot{\theta}(t+dt) = \dot{\theta}(t) + \ddot{\theta}(t)dt$$

pendulum_acceleration

Verlet Equations

update_pendulum_state.js

```
    if (numerical_integrator === "euler") {

    // STENCIL: a                                    for assignment

    }
    else if (numerical_integrator === "verlet") {

    // STE

    }
    else if (numerical_integrator === "velocity verlet") {

    // STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment

    }
    else if (numerical_integrator === "runge-kutta") {
```

$$\theta(t+dt) = \theta(t) + \dot{\theta}(t)dt$$
$$\dot{\theta}(t+dt) = \dot{\theta}(t) + \ddot{\theta}(t)dt$$

Don't forget to initialize in `init_verlet_integrator`!

$$\theta(t+dt) = 2\theta(t) - \theta(t-dt) + \ddot{\theta}(t)dt^2$$

`pendulum_acceleration`

Velocity Verlet Equations

update_pendulum_state.js

```
if (numerical_integrator === "euler") {
// STENCIL: a
```

$$\theta(t + dt) = \theta(t) + \dot{\theta}(t)dt$$
$$\dot{\theta}(t + dt) = \dot{\theta}(t) + \ddot{\theta}(t)dt$$

```
for assignment
}
else if (numerical_integrator === "verlet") {
// STE
```

$$\theta(t + dt) = 2\theta(t) - \theta(t - dt) + \ddot{\theta}(t)dt^2$$

```
}
else if (numerical_integrator === "velocity verlet") {
// STEN
```

$$\theta(t + dt) = \theta(t) + \dot{\theta}(t)dt + \frac{1}{2}\ddot{\theta}(t)dt^2$$

```
or assignment
}
else if
```

$$\dot{\theta}(t + dt) = \dot{\theta}(t) + \frac{\ddot{\theta}(t) + \ddot{\theta}(t + dt)}{2}dt$$

PID Control

update_pendulum_state.js

```
function set_PID_parameters(pendulum) {
    // STENCIL: change pid parameters
    pendulum.servo = {kp:0, kd:0, ki:0}; // no control
    return pendulum;
}

function PID(pendulum, accumulated_error, dt) {
    //
                                                  _error

    re
}
```

Tune PID parameters here

pendulum.servo.error

$$e(t) = \theta_{desired}(t) - \theta(t)$$

pendulum.control

$$\tau(t) = K_p e(t) + K_i \int_0^t e(\alpha)d\alpha + K_d \frac{d}{dt} e(t)$$

accumulated_error

$$\int_0^t e(\alpha)d\alpha = \sum_{\alpha=0}^{t} e(\alpha) = e(t) + \sum_{\alpha=0}^{t-1} e(\alpha)$$

Implementation Advice

- Pay attention to time index within equations!
  - Previous time step: $t - dt$
  - Current time step: t
  - Future time step: $t + dt$
- Parameterized helper functions can help reduce code duplication
  - We've done this for you with `pendulum_acceleration(pendulum, gravity)`
- Unnecessary global variables can be difficult to debug!

Motivation of Assignment

Understand how *error* can be used as feedback in a closed-loop fashion to control a system

- **Practice:** Implement a PID servo controller for a pendulum functioning under well defined rules (classical/Newtonian mechanics)

Motivation of Assignment

Why are we working with Newton's equations of motion and numerical integrators?

The real world operates under Newtonian mechanics, why not implement the pendulum and PID controller there?

- Too expensive and too slow!

Instead, implement pendulum in simulation

- Use Newton's equations of motion to model the change of pendulum's position over time (model of the state dynamics)
- With numerical integration techniques and an initial position, we can approximate the pendulum's position at any specific time

Motivation of Assignment

Understand how *error* can be used as feedback in a closed-loop fashion to control a system

- **Practice:** Implement a PID servo controller for a pendulum functioning under well defined rules (classical/Newtonian mechanics)

Understand why simulations are relevant in robotics and how they can be implemented using differential equations

- **Practice:** Build a digital simulation of the pendulum operating under the laws of classical mechanics