AutoRob

Introduction to Autonomous Robotics
Michigan EECS 367

Robot Kinematics and Dynamics
Michigan ME 567 EECS 567 ROB 510

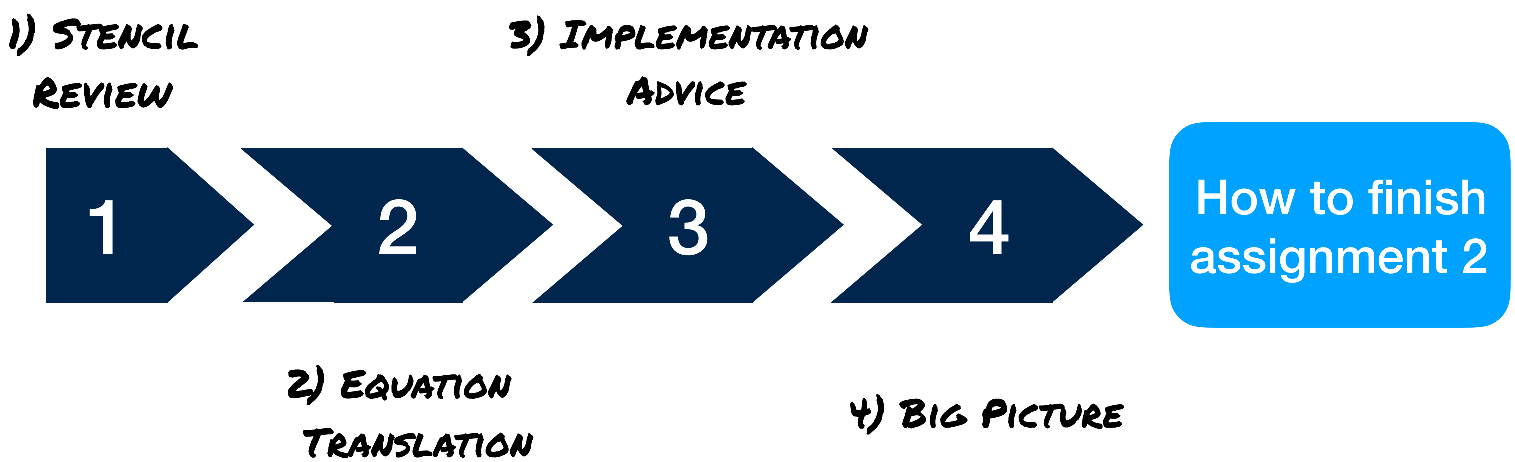
Fall 2019

EECS 367 Lab: pendularm1.html support

Administrative

- Assignment #2: Pendularm
  - Due 11:59pm, Wednesday, October 2
- Regrade policy described on course website

Lab Takeaways



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 GRADUATE SECTIONS

- FOR GRADUATE SECTIONS, ASSIGNMENT 2 STILL WORTH 18 POINTS
- 15 OF THESE POINTS CAN BE EARNED FROM LISTED FEATURES
- 3 OTHER POINTS LEFT FOR ADVANCED EXTENSIONS

Stencil Review

```
function animate() {
```

```
    // update servo desired state from user interaction
    if ( keyboard.pressed("e") )
        pendulum.desired += 0.05; // move the desired angle for the servo
    if ( keyboard.pressed("q") )
        pendulum.desired += -0.05; // move the desired angle for the servo


    // add user force from user interaction
    if ( keyboard.pressed("d") )
        pendulum.control += 50.0;  // add a motor force to the pendulum motor
    else if ( keyboard.pressed("a") )
        pendulum.control += -50.0; // add a motor force to the pendulum motor

    // STENCIL: implement servo controller
```

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

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

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

PID STENCIL

Stencil Review

```
function animate() {
```

```
    if (numerical_integrator === "euler") {
    // STENCIL: a correct Euler integrator is REQUIRED 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") {
    // STENCIL: Runge-Kutta 4 integrator
    }
   pendulum.geom.rotation.y = pendulum.angle;  // threejs cylinders have their
```

FEATURE STENCILS

DEFAULT ROTATION

UPDATE PENDULUM

Stencil Review

```
function animate() {
```

```
if (numerical_integrator === "euler") {
// STENCIL: a correct Euler integrator is REQUIRED for assignment
```

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

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

$$= \dot{\theta}(t) + \texttt{pendulum\_acceleration}(P, g, t)$$

```
}
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") {

// STENCIL: Runge-Kutta 4 integrator
}
```

FEATURE STENCILS

Stencil Review

```
function animate() {
```

```
if (numerical_integrator === "euler") {
// STENCIL: a correct Euler integrator is REQUIRED for assignment
```

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

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

$$= \dot{\theta}(t) + \texttt{pendulum\_acceler}$$

```
}
else if (numerical_integrator === "verlet") {
// STENCIL: basic Verlet integration
```

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

$$= 2\theta(t) - \theta(t-dt) + \texttt{pendulum\_acceleration}(P, g, t)dt^2$$

```
}
else if (numerical_integrator === "velocity verlet") {
// STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment






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

// STENCIL: Runge-Kutta 4 integrator
}
```

DON'T FORGET TO INITIALIZE FOR VERLET!

FEATURE STENCILS

Stencil Review

```
function animate() {
```

```
if (numerical_integrator === "euler") {
// STENCIL: a correct Euler integrator is REQUIRED for assignment
```

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

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

$$= \dot{\theta}(t) + \texttt{pendulum\_acceleration}(P, g, t)$$

```
}
else if (numerical_integrator === "verlet") {
// STENCIL: basic Verlet integration
```

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

$$= 2\theta(t) - \theta(t-dt) + \texttt{pendulum\_acceleration}(P, g, t)dt^2$$

```
}
else if (numerical_integrator === "velocity verlet") {
// STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment
```

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

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

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

// STENCIL: Runge-Kutta 4 integrator
}
```

FEATURE STENCILS

PAY ATTENTION TO ANGLE USED BY ACCELERATION CALCULATION

Stencil Review

```
function animate() {
```

```
            " a/d - apply user force " +
            "<br>" +
            " q/e - adjust desired angle " +
            "<br>" +
            " c|x - toggle servo " +
            "<br>" +
            " s - disable servo "

    ;

    // threejs rendering update
    renderer.render( scene, camera );

}
```

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

ACCELERATION STENCIL

```
function createScene() {

    // instantiate threejs scene graph
    scene = new THREE.Scene();

    // instantiate threejs camera and set its position in the world
    camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.inner
    camera.position.y = 1;
    camera.position.z = 4;
```

Implementation Advice

- Pay attention to time index within equations!
  - Ensure you're using the correct of:

previous	current	future
$t - dt$	t	$t + dt$

- Parameterized helper functions can reduce code duplication
  - Like, `pendulum_acceleration(P,g,…)`

- Unnecessary global variables can be difficult to debug

Motivation of Assignment

- Desired outcome of the pendularm assignment:
    - Understand how ‘error’* can be used as feedback in a closed-loop fashion to control a system

      ↳ Implement a PID servo controller for a pendulum functioning under well defined rules (classical/Newtonian mechanics)

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

*’error’ referring to some measure of difference between current and goal states of some system

Motivation of Assignment

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

- The real—physical—world operates under Newtonian mechanics, why not implement our pendulum and PID controller there?

↳ **Too expensive, too slow**

- How can we implement a pendulum in simulation?

Motivation of Assignment

- How can we implement a 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

INSIDE OF SIMULATION

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

OUTSIDE OF SIMULATION

$$\theta(t+dt) = \texttt{get\_measurement}(\theta, \texttt{realWorld})$$

Motivation of Assignment

- Desired outcome of the pendularm assignment:
  - Understand how ‘error’* can be used as feedback in a closed-loop fashion to control a system

    ↳ 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

    ↳ Build a digital simulation of the pendulum operating under the laws of classical mechanics

*’error’ referring to some measure of difference between current and goal states of some system

Final Thoughts

- If you think of an idea for an advanced extension, **which you are excited about,** let the course staff know!
  - We will consider the idea for possible extension points (following the working implementation)
- Have a great weekend!