EECS 367 & ROB 320 Interactive Session

Administrative

- Assignment #4: Robot FSM Dance Contest
  - Due Friday, March 11, 11:59pm
- Dance Contest-Demo
  - Next Wednesday, March 16th
  - During Interactive Session
  - Not additional feature points

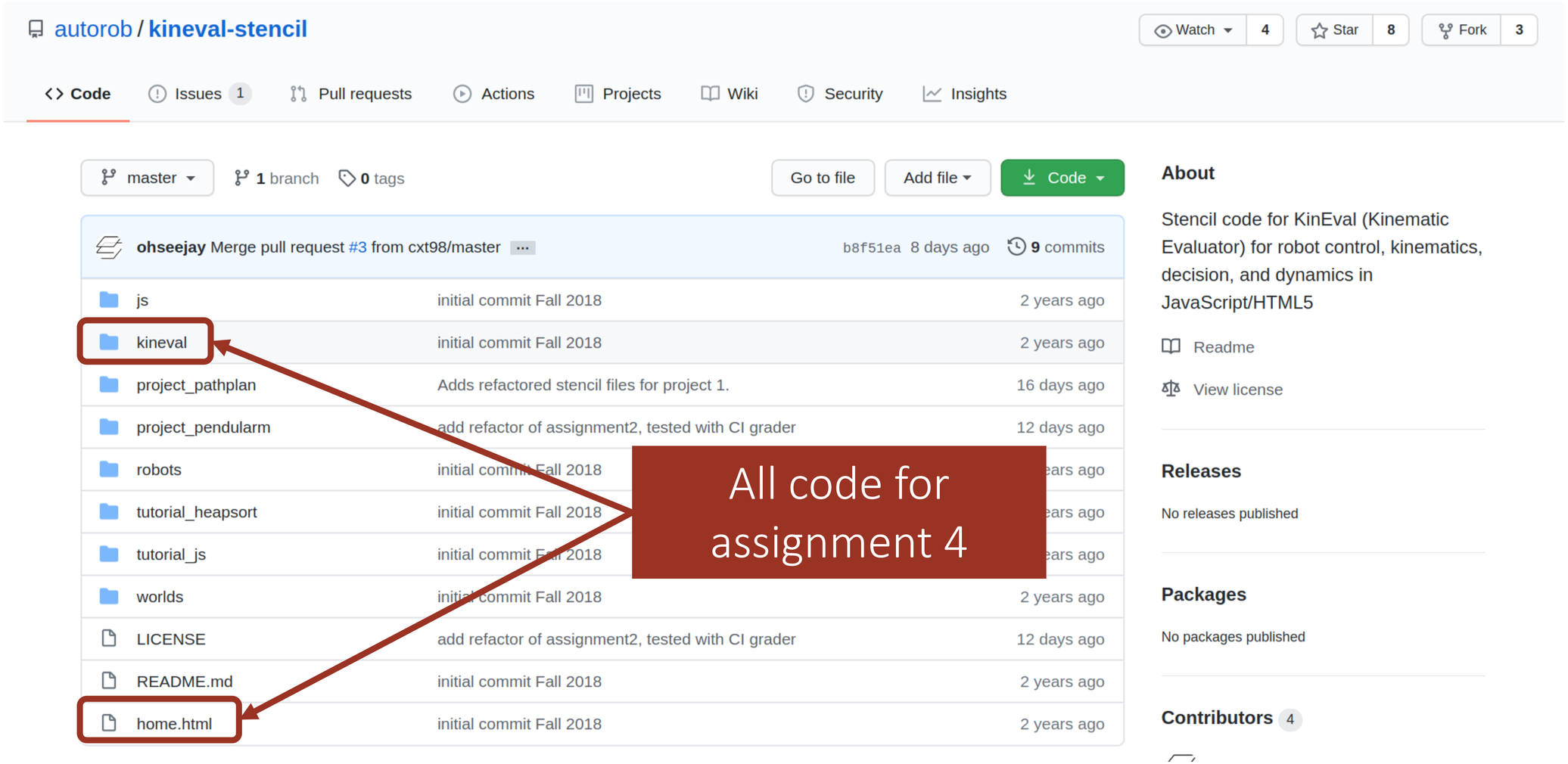
Today's Topics

- General Course Q&A
- Reviewing KinEval Assignment 4 Stencil
- FSM Walkthrough Q&A
- Extended Office hours

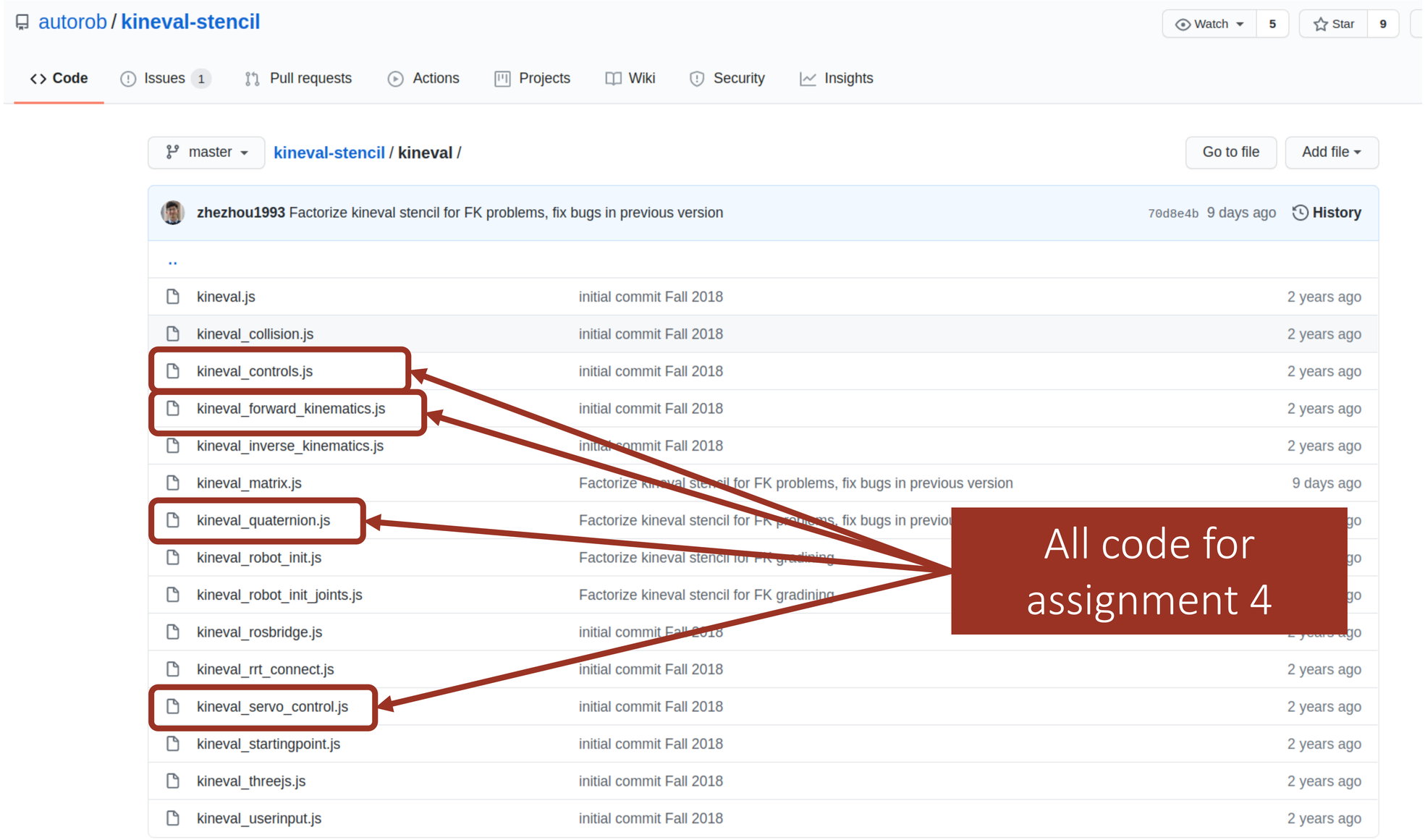
Dance Controller Overview

	Assignment 4: Dance Controller
6	Quaternion joint rotation
2	Interactive base control
2	Pose setpoint controller
2	Dance FSM

KinEval Overview



KinEval Overview



kineval_forward_kinematics.js Revisited

kineval_forward_kinematics.js

```
18
19  kineval.robotForwardKinematics = function robotForwardKinematics () {
20
21      if (typeof kineval.buildFKTransforms === 'undefined') {
22          textbar.innerHTML = "forward kinematics not implemented";
23          return;
24      }
25
26      // STENCIL: implement kineval.buildFKTransforms();
27
28  }
29
      // STENCIL: reference code alternates recursive traversal over
      nts starting from base, using following functions:
      ase
      ink
      oint
35      //
```

For each joint, incorporate `.axis` and `.angle` within forward kinematics. You will then be able to control joints!

Define global `heading` and `lateral` vectors with respect to global space

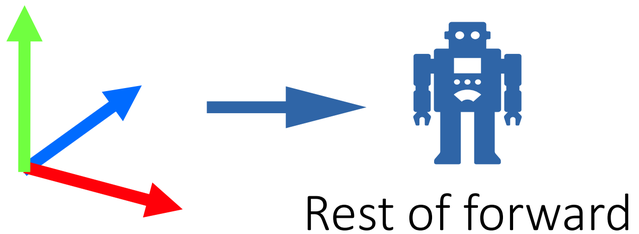
kineval_quaternion.js

kineval_quaternion.js

```
1  ////////////////////////////////////////////////
2  /////     QUATERNION TRANSFORM ROUTINES
3  ////////////////////////////////////////////////
4
5  // STENCIL: reference quaternion code has the following functions:
6  //   quaternion_from_axisangle
7  //   quaternion_normalize
8  //   quaternion_to_rotation_matrix
9  //   quaternion_multiply
10
11 // **** Function stencils are provided below, please uncomment and implement them ****//
12
13 // kineval.quaternionFromAxisAngle = function quaternion_from_axisangle(axis,angle) {
```

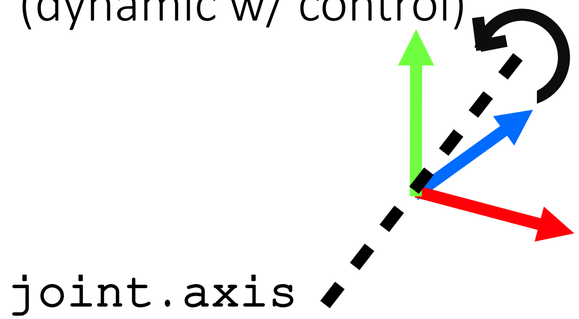
Define quaternion helper functions
→ Create a joint's rotation matrix from any axis-angle pair

Joint frame without control



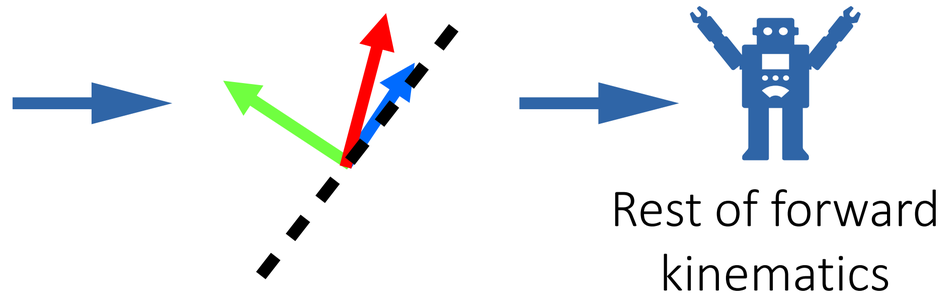
Rest of forward kinematics

`joint.angle`
(dynamic w/ control)



`joint.axis`
(constant)

Joint frame with control



Rest of forward kinematics

Lecture 8 - Axis-angle Rotation and Quaternions

1) form unit quaternion from axis and motor angle

$$q = [\cos(\Theta/2), u_x \sin(\Theta/2), u_y \sin(\Theta/2), u_z \sin(\Theta/2)]$$

2) convert quaternion to rotation matrix

- Inhomogeneous conversion to 3D rotation matrix of $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

$$\begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

or equivalently, homogeneous conversion

$$\begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

- Rotation matrix to quaternion can also be performed

kineval_controls.js

kineval_controls.js

```
19  kineval.applyControls = function robot_apply_controls(curRobot) {
20      // apply robot controls to robot kinematics transforms and joint angles, then zero controls
21      // includes update of camera position based on base movement
22
23      // update robot configuration from controls
24      for (x in curRobot.joints) {
25
26          // update joint angles
27          if ( (typeof curRobot.joints[x].type !== 'undefined')
28               || (typeof curRobot.joints[x].type !== 'fixed') ) {
29
30              if (isNaN(curRobot.joints[x].control))
31                  console.warn("kineval: control value for " + x +" is a nan");
32
33              curRobot.joints[x].angle += curRobot.joints[x].control;
34          }
35
36      // STENCIL: enforce joint limits for prismatic and revolute joints
37
38
39          // clear controls back to zero for next timestep
40          curRobot.joints[x].control = 0;
41      }
```

Control is already applied to all `joint.angles` and `robot.origin` for you

Grad section will need to enforce joint limits

kineval_servo_control.js

kineval_servo_control.js

```
19  kineval.setpointDanceSequence = function execute_setpoints() {
20
21      // if update not requested, exit routine
22      if (!kineval.params.update_pd_dance) return;
23
24      // STENCIL: implement FSM to cycle through dance pose setpoints
25  }
26
27  kineval.setpointClockMovement = function execute_clock() {
28
29      // if update not requested, exit routine
30      if (!kineval.params.update_pd_clock) return;
31
32      var curdate = new Date();
33      for (x in robot.joints) {
34          kineval.params.setpoint_target[x] = curdate.getSeconds()/60*2*Math.PI;
35      }
36  }
37
38
39  kineval.robotArmControllerSetpoint = function robot_pd_control () {
40
41      // if update not requested, exit routine
42      if ((!kineval.params.update_pd)&&(!kineval.params.persist_pd)) return;
43
44      kineval.params.update_pd = false; // if update requested, clear request and process setpoint control
45
46      // STENCIL: implement P servo controller over joints
47  }
```

Implement a Finite State Machine for setpoint dance routine

Implement P controller for joint control to setpoints

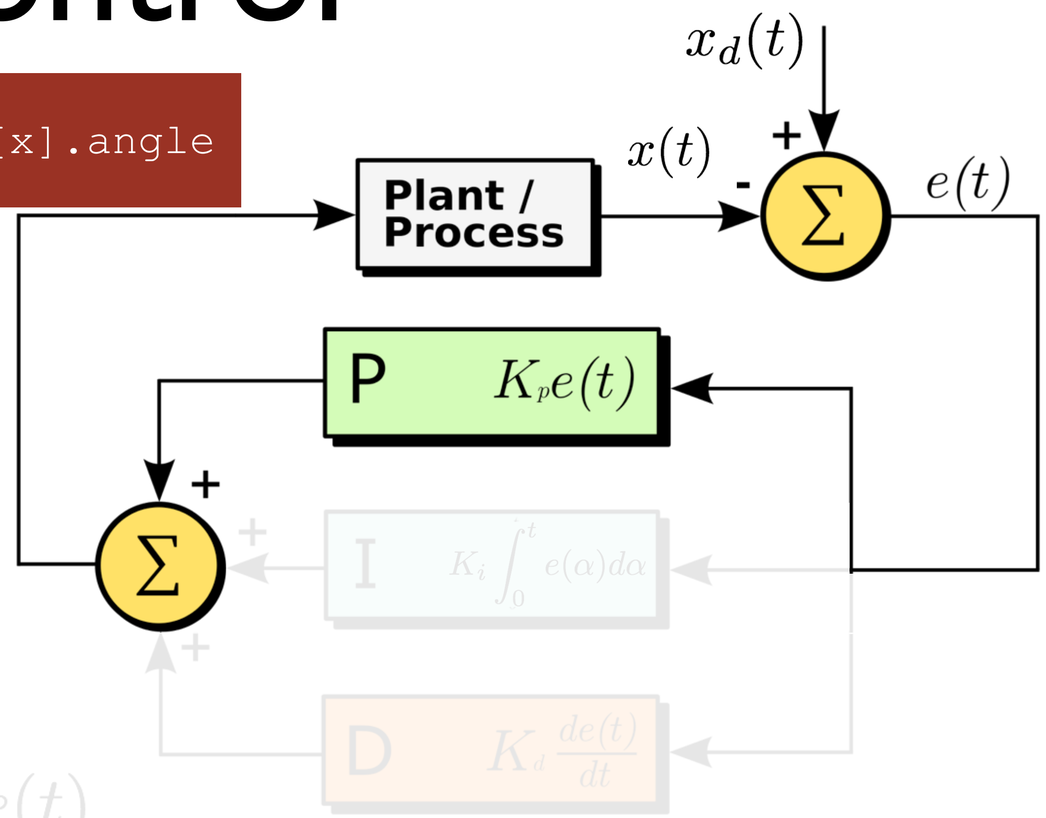
PID Control

`kineval.params.setpoint_target[x]`
or
`robot.joints[x].servo.p_desired`

`robot.joints[x].angle`

Error signal:

$$e(t) = x_{desired}(t) - x(t)$$



Control signal:

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

P $K_p e(t)$ — Current

I $K_i \int_0^t e(\alpha)d\alpha$ — Past

D $K_d \frac{de(t)}{dt}$ — Future

kineval_servo_control.js

kineval_servo_control.js

```
19  kineval.setpointDanceSequence = function execute_setpoints() {
20
21      // if update not requested, exit routine
22      if (!kineval.params.update_pd_dance) return;
23
24      // STENCIL: implement FSM to cycle through dance pose setpoints
25  }
26
27  kineval.setpointClockMovement = function execute_clock() {
28
29      // if update not requested, exit routine
30      if (!kineval.params.update_pd_clock) return;
31
32      var curdate = new Date();
33      for (x in robot.joints) {
34          kineval.params.setpoint_target[x] = curdate.getSeconds()/60*2*Math.PI;
35      }
36  }
37
38
39  kineval.robotArmControllerSetpoint = function robot_pd_control () {
40
41      // if update not requested, exit routine
42      if ((!kineval.params.update_pd)&&(!kineval.params.persist_pd)) return;
43
44      kineval.params.update_pd = false; // if update requested, clear request and process setpoint control
45
46      // STENCIL: implement P servo controller over joints
47  }
```

Implement a Finite State Machine for setpoint dance routine

Implement P controller for joint control to setpoints

FSM Pseudocode

- Identify current setpoint
- Compare current pose to setpoint pose
    - If current pose is similar, update setpoint

FSM Pseudocode

Logic-based Approach

- Identify current setpoint
- Iterate over each joint
  - If joint's angle is 'close' to setpoint[joint]'s angle
    - Continue
  - Else
    - Return
- Update setpoint index

Cumulative Sum-based Approach

- Identify current setpoint
- Iterate over each joint
  - total_error += difference(joint, setpoint[joint])
- If total_error<threshold
  - Update setpoint index

kineval_servo_control.js

kineval_servo_control.js

```
19  kineval.setpointDanceSequence = function execute_setpoints() {
20
21      // if update not requested, exit routine
22      if (!kineval.params.update_pd_dance) return;
23
24      // STENCIL: implement FSM to cycle through dance pose setpoints
25  }
26
27  kineval.setpointClockMovement = function execute_clock() {
28
29      // if update not requested, exit routine
30      if (!kineval.params.update_pd_clock) return;
31
32      var curdate = new Date();
33      for (x in robot.joints) {
34          kineval.params.setpoint_target[x] = curdate.getSeconds()/60*2*Math.PI;
35      }
36  }
37
38
39  kineval.robotArmControllerSetpoint = function robot_pd_control () {
40
41      // if update not requested, exit routine
42      if ((!kineval.params.update_pd)&&(!kineval.params.persist_pd)) return;
43
44      kineval.params.update_pd = false; // if update requested, clear request and process setpoint control
45
46      // STENCIL: implement P servo controller over joints
47  }
```

Implement a Finite State Machine for setpoint dance routine

Implement P controller for joint control to setpoints

home.html

home.html

```
131  ////////////////////////////////////////////////
132  /////     MAIN FUNCTION CALLS
133  ////////////////////////////////////////////////
134
135  // start KinEval execution once the page and its resources are loaded
136  //window.onload = kineval.start;
137  document.body.onload = kineval.start;
138
139  // STUDENT: my_animate is where your robot's controls and movement are updated over time
140  function my_init() {
141
142      kineval.startingPlaceholderInit(); // a quick and dirty JavaScript tutorial
143
144  }
```

Initialize `kineval.setpoints` and `kineval.params.dance_sequence_index` here

Create a cool dance routine by defining a sequence of joint angle setpoints to be used by the FSM implementation

Poses for servo can be set and stored **interactively** in KinEval using [0-9] keys and Shift+[0-9]

`JSON.stringify(kineval.setpoints)` will output the currently available servo setpoints to the console as a string