EECS 367 Lab: search_canvas.html Revisited
Lab Takeaways

1) KinEval Overview

2) KinEval Walkthrough

How to start assignment 6
## Forward Kinematics Overview

<table>
<thead>
<tr>
<th>Assignment 6: Motion Planning</th>
<th>Features assigned to all sections</th>
<th>Features assigned to graduate sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 All</td>
<td>Collision detection</td>
<td>2D RRT-Connect</td>
</tr>
<tr>
<td>2 All</td>
<td>2D RRT-Connect</td>
<td>Configuration space RRT-Connect</td>
</tr>
<tr>
<td>6 All</td>
<td>Configuration space RRT-Connect</td>
<td>RRT-Star</td>
</tr>
<tr>
<td>6 Grad</td>
<td>RRT-Star</td>
<td></td>
</tr>
</tbody>
</table>
# KinEval Overview

- [kineval.js](#)  initial commit Fall 2018  last year
- [kineval_collision.js](#)  initial commit Fall 2018  last year
- [kineval_controls.js](#)  initial commit Fall 2018  last year
- [kineval_forward_kinematics.js](#)  initial commit Fall 2018  last year
- [kineval-inverse_kinematics.js](#)  initial commit Fall 2018  last year
- [kineval_matrix.js](#)  initial commit Fall 2018  last year
- [kineval_quaternion.js](#)  initial commit Fall 2018  last year
- [kineval_robot_init.js](#)  initial commit Fall 2018  last year
- [kineval_rosbridge.js](#)  initial commit Fall 2018  last year
- [kineval_rrt_connect.js](#)  initial commit Fall 2018  last year
- [kineval_servo_control.js](#)  initial commit Fall 2018  last year
- [kineval_startingpoint.js](#)  initial commit Fall 2018  last year
- [kineval_threejs.js](#)  initial commit Fall 2018  last year
- [kineval_userinput.js](#)  initial commit Fall 2018  last year
Rapidly-exploring Random Trees (RRT)
search_canvas.html

Similar to past search algorithms, implement each function as a single step within the iterative algorithm.

Start with 2D implementation in search_canvas.html
Tree structure implemented as a JavaScript object with array of vertices

```javascript
function initRRT(q) {
    // create tree object
    var tree = {};
    // initialize with vertex for given configuration
    tree.vertices = [];
    tree.vertices[0] = {};
    tree.vertices[0].vertex = q;
    tree.vertices[0].edges = [];
    // maintain index of newest vertex added to tree
    tree.newest = 0;
    return tree;
}
```

```
tree = {
    "vertices": [
        {
            "vertex": [x1,y1],
            "edges": [tree.vertices[1]],
        },
        {
            "vertex": [x2,y2],
            "edges": [tree.vertices[0]],
        },
    ]
}
```

![Tree structure diagram]

- [x1,y1]
- [x2,y2]
- [x1,y1]
search_canvas.html

Helper functions provided for you

```
function initRRT(q) {
  // create tree object
  var tree = {};
  // initialize with vertex for given configuration
  tree.vertices[q] = {}
  tree.vertices[q].vertex = q;
  tree.vertices[q].edges = [];
  // maintain index of newest vertex added to tree
  tree.newest = q;
  return tree;
}
```

```
function insertTreeVertex(tree, q) {
  // create new vertex object for tree with given configuration and no edges
  new_vertex = {};
  new_vertex.vertex = q;
  tree.vertices.push(new_vertex);
  // maintain index of newest vertex added to tree
  tree.newest = tree.vertices.length - 1;
  // draw location on canvas
  draw_2D_configuration(q);
}
```

```
function insertTreeEdge(tree, q1_idx, q2_idx) {
  // add edge to first vertex as pointer to second vertex
  tree.vertices[q1_idx].edges.push(tree.vertices[q2_idx].vertex);
  // add edge to second vertex as pointer to first vertex
  tree.vertices[q2_idx].edges.push(tree.vertices[q1_idx].vertex);
  // draw edge on canvas
  draw_2D_edge_configurations(tree.vertices[q1_idx].vertex, tree.vertices[q2_idx].vertex);
}
```
motion planner has been invoked in the background.
q_robot_config is an array representing current pose as point within configuration space.

Dimension of configuration space is a function of the robot.
Axis-aligned Bounding Boxes (AABB)

3D collision detection

Game development › Techniques for game development › 3D collision detection

Axis-aligned bounding boxes

As with 2D collision detection, axis-aligned bounding boxes (AABB) are the quickest algorithm to determine whether the two game entities are overlapping or not. This consists of wrapping game entities in a non-rotated (thus axis-aligned) box and checking the positions of these boxes in the 3D coordinate space to see if they are overlapping.

The axis-aligned constraint is there because of performance reasons. The overlapping area between two non-rotated boxes can be checked with logical comparisons alone, whereas rotated boxes require additional trigonometric operations, which are slower to calculate.

Source: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection

Sphere Obstacle Collision

If sphere separable from AABB in any dimension, return no collision

loc_y-radius>y_max?

loc_y+radius<y_min?

loc_x+radius<x_min? loc_x-radius>x_max?

If sphere collides on all tests, return collision

loc_y-radius>y_max?

loc_y+radius<y_min?

loc_x+radius<x_min? loc_x-radius>x_max?
function traverse_collision_forward(kineval_collision_world, link, stack, i) {
  /* test collision FK */
  console.log(link);
  if (typeof link.visual === 'undefined') {
    var local_link_xform = matrix_multiply(stack, generate_translation_matrix(link.visual.origin.xyz[0], link.visual.origin.xyz[1], link.visual.origin.xyz[2]));
  } else {
    var local_link_xform = matrix_multiply(stack, generate_identity());
  }
  var local_link_xform = matrix_multiply(stack, generate_identity());
  /* test collision by transforming obstacles in world to link space */
  var stack_inv = matrix_invert_affine(stack);
  var stack_inv = numeric.inv(stack);
  // test each obstacle against link box geometry by transforming obstacle into link frame and testing against axis aligned bounding box.
  for (i = 0; i < robot_obstacles.length; i++) { //for (i = 0; i < robot_obstacles.length; i++) {
    var obstacle_location = matrix_multiply(stack_inv, robot_obstacles[0].location);
    var in_collision = true;
    // assume link is in collision as default
    var in_collision = true;
    // if obstacle lines outside the link extents along any dimension, no collision is detected
    if (obstacle_location[0] > link.bbox.min.x - robot_obstacles[0].radius ||
        obstacle_location[0] < link.bbox.max.x + robot_obstacles[0].radius)
      in_collision = false;
    if (obstacle_location[1] > link.bbox.min.y - robot_obstacles[0].radius ||
        obstacle_location[1] < link.bbox.max.y + robot_obstacles[0].radius)
      in_collision = false;
    if (obstacle_location[2] > link.bbox.min.z - robot_obstacles[0].radius ||
        obstacle_location[2] < link.bbox.max.z + robot_obstacles[0].radius)
      in_collision = false;
  }
}

loc_y-radius>y_max?  no collision
loc_y+radius<y_min?
loc_x+radius<x_min?  loc_x-radius>x_max?

If sphere separable from AABB in any dimension, return no collision

loc_y-radius>y_max?
loc_y+radius<y_min?
loc_x+radius<x_min?  loc_x-radius>x_max?

If sphere collides on all tests, return collision

Michigan EECS 398/567 ROB 510 - aubrob.org
Collision detection pseudo code:
- For each link in robot
  - For each obstacle in world
    - If intersection(link, obstacle)
      - Return link is in collision
    - Return no collision

```
kineval.poseIsCollision = function robot_collision_test(q) {
  // perform collision test of robot geometry against planning world
  // test base origin (not extents) against world boundary extents
  if ((q[0]<robot_boundary[0][0]) || (q[0]>robot_boundary[1][0]) || (q[1]<robot_boundary[0][1]) || (q[1]>robot_boundary[1][1]) || (q[2]<robot_boundary[0][2]) || (q[2]>robot_boundary[1][2]))
    return robot.base;
  // traverse robot kinematics to test each body for collision
  // STENCIL: implement forward kinematics for collision detection
  // return robot_collision_forward_kinematics(q);
}
```
kineval_rrt_connect.js

```javascript
function robot_rrt_planner_iterate() {

  var i;
  rrt_alg = 1; // 0: basic rrt (OPTIONAL), 1: rrt_connect (REQUIRED)

  if (rrt_iterate && (Date.now()-cur_time > 10)) {
    cur_time = Date.now();

    // STENCIL: implement single rrt iteration here. an async
    // is used instead of a for loop to avoid blocking and
    // in the browser.
    //
    // once plan is found, highlight vertices of found path
    // tree.vertices[i].vertex[j].geom.material.color =
    // provided support functions:
    //
    // kineval.poseIsCollision - returns if a configuration is in collision
    // tree_init - creates a tree of configurations
    // tree_add_vertex - adds and displays new configuration vertex for a tree
    // tree_add_edge - adds and displays new tree edge between configurations
  }
}
```
Rapidly-exploring Random Trees (RRT)

**Arbitrary Initial Configuration**

**Collision Free Path to Home**

**Backward Step along Motion Plan**

**Forward Step along Motion Plan**