AutoRob

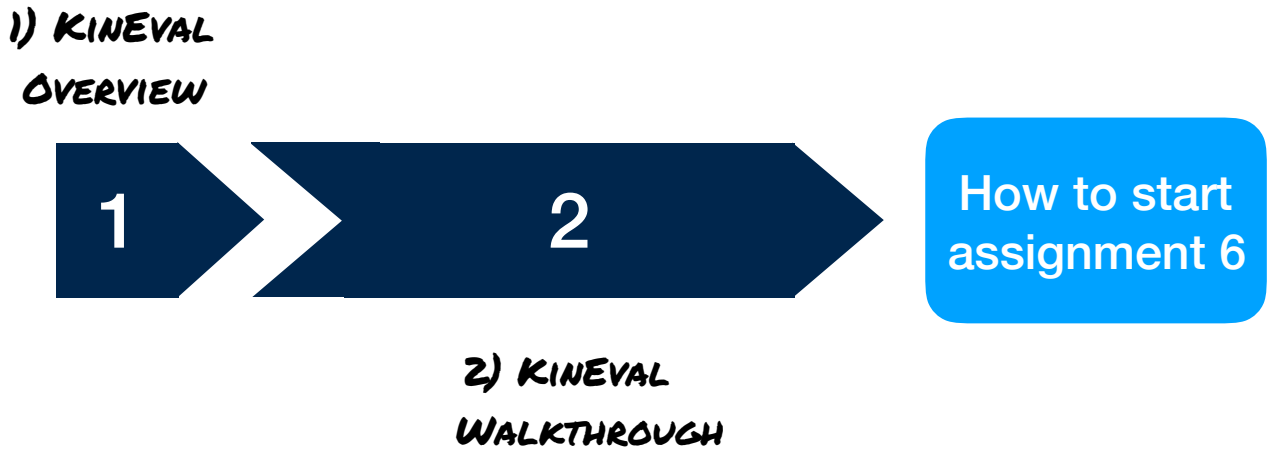
Introduction to Autonomous Robotics
Michigan EECS 367

Robot Kinematics and Dynamics
Michigan ME 567 EECS 567 ROB 510

Fall 2019

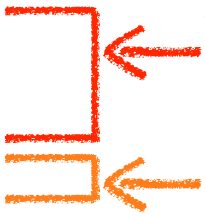
EECS 367 Lab: search_canvas.html Revisited

Lab Takeaways



Forward Kinematics Overview

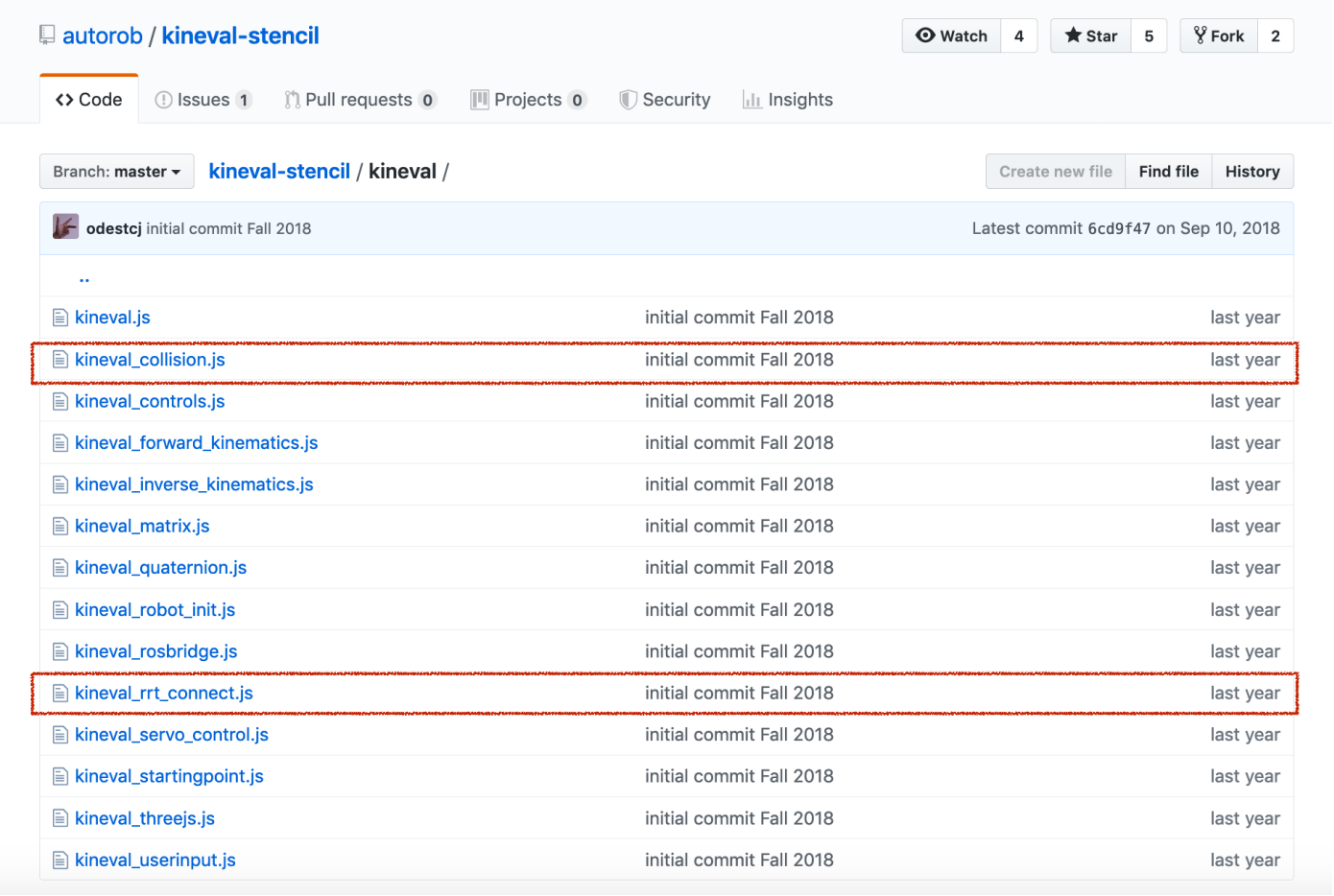
		Assignment 6: Motion Planning
4	All	Collision detection
2	All	2D RRT-Connect
6	All	Configuration space RRT-Connect
6	Grad	RRT-Star



Features assigned to all sections

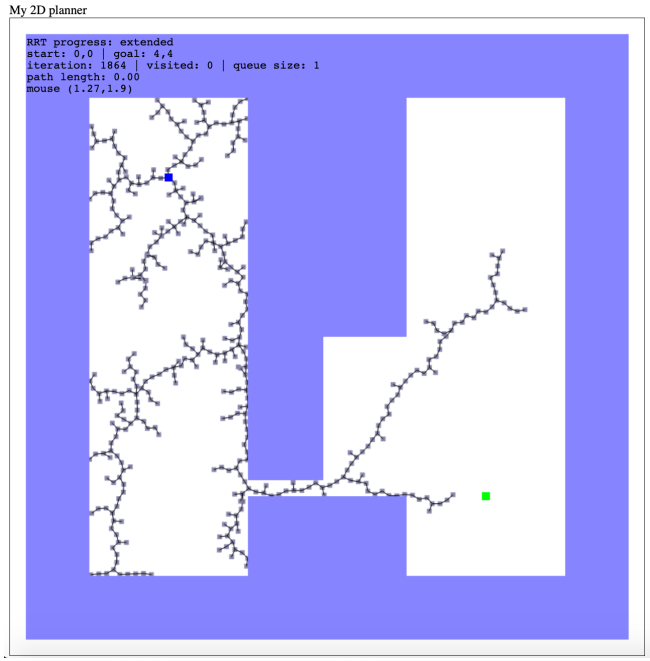
Features assigned to graduate sections

KinEval Overview

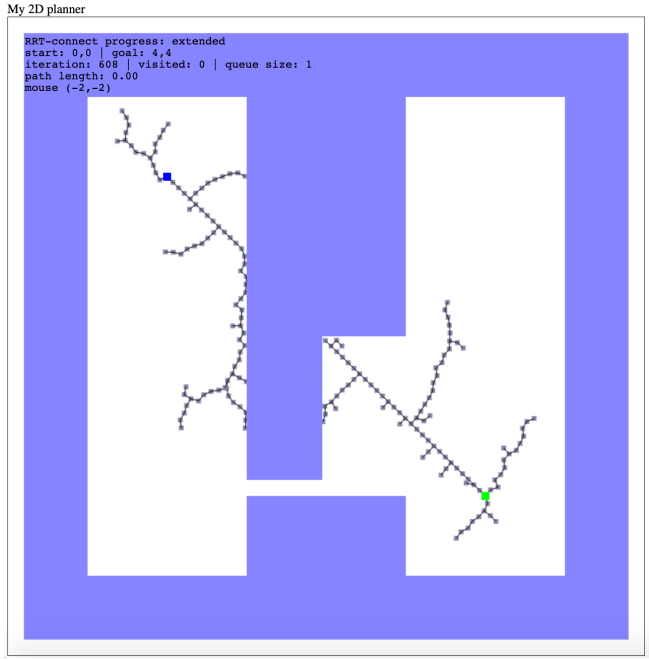


Rapidly-exploring Random Trees (RRT)

RRT



RRT Connect



search_canvas.html

Start with 2D implementation in search_canvas.html

search_canvas.html

```
296  function iterateRRT() {
297
298
299      // STENCIL: implement a single iteration of an RRT algorithm.
300      //   An asynch timing mechanism is used instead of a for loop to avoid
301      //   blocking and non-responsiveness in the browser.
302      //
                                    he search fails on this iteration.
                                    f the search succeeds on this iteration.
                                    herwise.
                                    nctions:
308      //
309      //   testCollision - returns whether a given configuration is in collision
310      //   tree_init - creates a tree of configurations
311      //   insertTreeVertex - adds and displays new configuration
312      //   insertTreeEdge - adds and displays new tree edge betwe
313      //   drawHighlightedPath - renders a highlighted path in a
314  }
315
316  function iterateRRTConnect() {
317
318
319      // STENCIL: implement a single iteration of an RRT-Connect
320      //   An asynch timing mechanism is used instead of a for lo
321      //   blocking and non-responsiveness in the browser.
322      //
323      //   Return "failed" if the search fails on this iteration.
324      //   Return "succeeded" if the search succeeds on this iteration.
325      //   Return "extended" otherwise.
326      //
327      //   Provided support functions:
328      //
329      //   testCollision - returns whether a given configuration is in collision
330      //   tree_init - creates a tree of configurations
331      //   insertTreeVertex - adds and displays new configuration vertex for a tree
332      //   insertTreeEdge - adds and displays new tree edge between configurations
333      //   drawHighlightedPath - renders a highlighted path in a tree
334  }
```

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

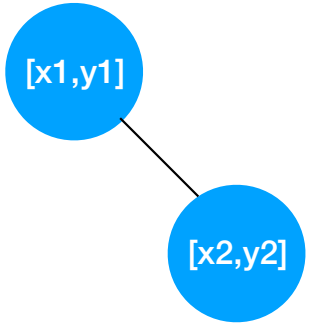
search_canvas.html

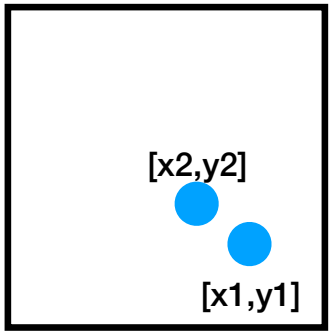
search_canvas.html

```
506  function initRRT(q) {
507
508      // create tree object
509      var tree = {};
510
511      // initialize with vertex for given configuration
512      tree.vertices = [];
513      tree.vertices[0] = {};
514      tree.vertices[0].vertex = q;
515      tree.vertices[0].edges = [];
516
517      // maintain index of newest vertex added to tree
518      tree.newest = 0;
519
520      return tree;
521  }
```

TREE STRUCTURE IMPLEMENTED AS A JAVASCRIPT OBJECT WITH ARRAY OF VERTICES

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





search_canvas.html

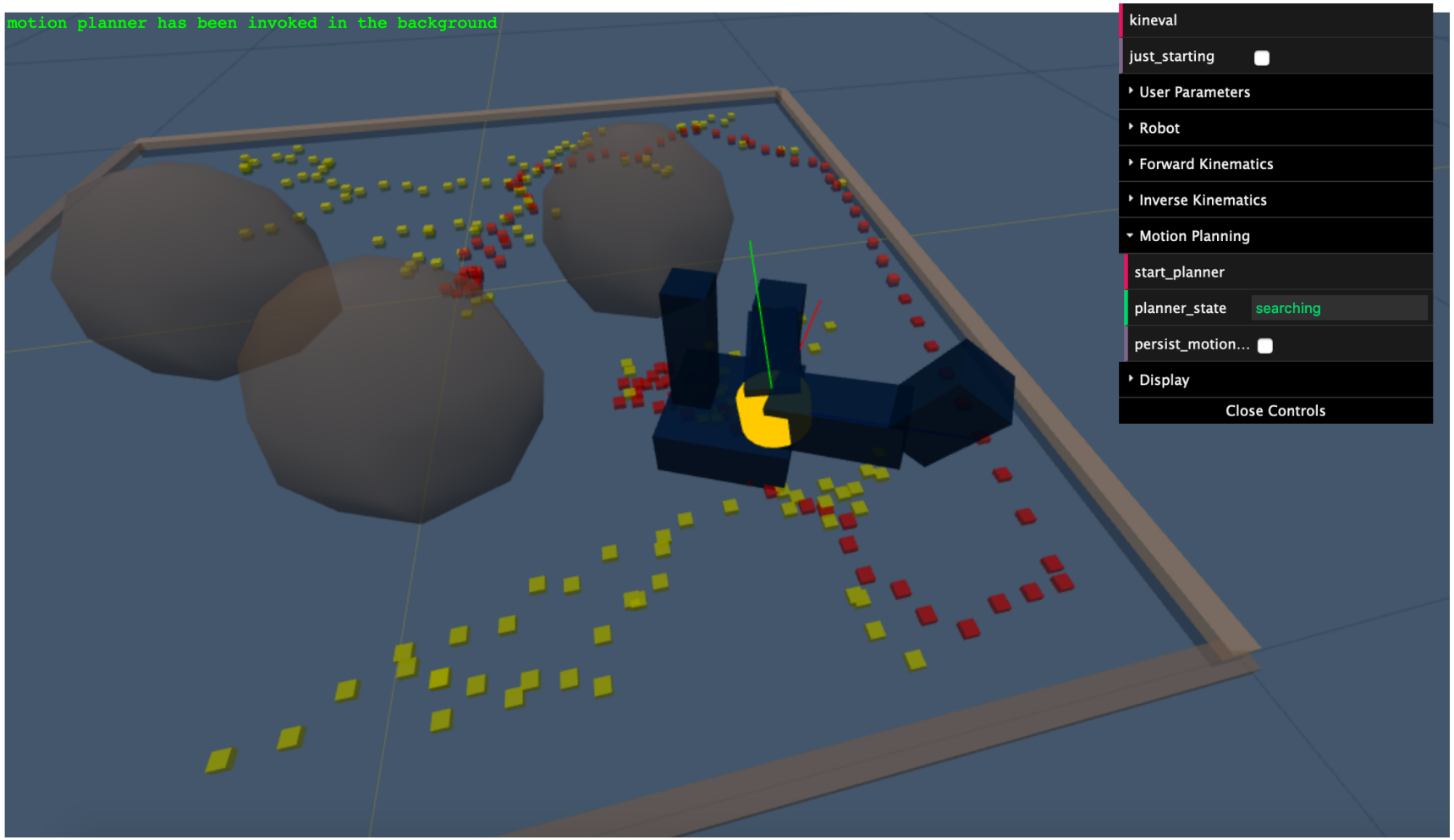
Helper functions provided for you

search_canvas.html

```
506
507  function initRRT(q) {
508
509      // create tree object
510      var tree = {};
511
512      // initialize with vertex for given configuration
513      tree.vertices = [];
514      tree.vertices[0] = {};
515      tree.vertices[0].vertex = q;
516      tree.vertices[0].edges = [];
517
518      // maintain index of newest vertex added to tree
519      tree.newest = 0;
520
521      return tree;
522  }
523
524  function insertTreeVertex(tree,q) {
525
526      // create new vertex object for tree with given configuration and no edges
527      new_vertex = {};
528      new_vertex.edges = [];
529      new_vertex.vertex = q;
530      tree.vertices.push(new_vertex);
531
532      // maintain index of newest vertex added to tree
533      tree.newest = tree.vertices.length - 1;
534
535      // draw location on canvas
536      draw_2D_configuration(q);
537  }
538
539  function insertTreeEdge(tree,q1_idx,q2_idx) {
540
541      // add edge to first vertex as pointer to second vertex
542      tree.vertices[q1_idx].edges.push(tree.vertices[q2_idx]);
543
544      // add edge to second vertex as pointer to first vertex
545      tree.vertices[q2_idx].edges.push(tree.vertices[q1_idx]);
546
547      // draw edge on canvas
548      draw_2D_edge_configurations(tree.vertices[q1_idx].vertex,tree.vertices[q2_idx].vertex);
549  }
```

Additional helper functions that will be useful for RRT

```
560  ////////////////////////////////////////////////
561  /////     RRT IMPLEMENTATION FUNCTIONS
562  ////////////////////////////////////////////////
563
564      // STENCIL: implement RRT-Connect functions here, such as:
565      //   extendRRT
566      //   connectRRT
567      //   randomConfig
568      //   newConfig
569      //   findNearestNeighbor
570      //   dfsPath
571
```


motion planner has been invoked in the background
kineval
just_starting
User Parameters
Robot
Forward Kinematics
Inverse Kinematics
Motion Planning
start_planner
planner_state
searching
persist_motion...
Display
Close Controls

kineval_collision.js

kineval_collision.js

```
22  kineval.robotIsCollision = function robot_iscollision() {
23      // test whether geometry of current configuration of robot is in collision with planning world
24
25      // form configuration from base location and joint angles
26      var q_robot_config = [
27          robot.origin.xyz[0],
28          robot.origin.xyz[1],
29          robot.origin.xyz[2],
30          robot.origin.rpy[0],
31          robot.origin.rpy[1],
32          robot.origin.rpy[2]
33      ];
34
35      q_names = {};  // store mapping between joint names and q DOFs
36
37      for (x in robot.joints) {
38          q_names[x] = q_robot_config.length;
39          q_robot_config = q_robot_config.concat(robot.joints[x].angle);
40      }
41
42      // test for collision and change base color based on the result
43      collision_result = kineval.poseIsCollision(q_robot_config);
44
45      robot.collision = collision_result;
46  }
```

`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)

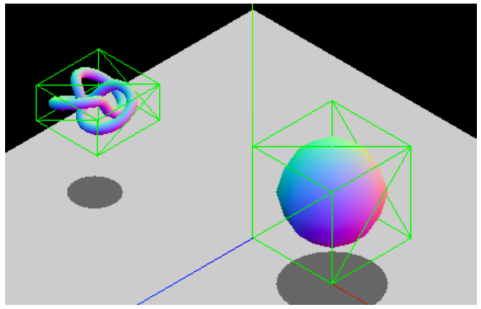
MDN web docs
moz://a

3D collision detection

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

Axis-aligned bounding boxes

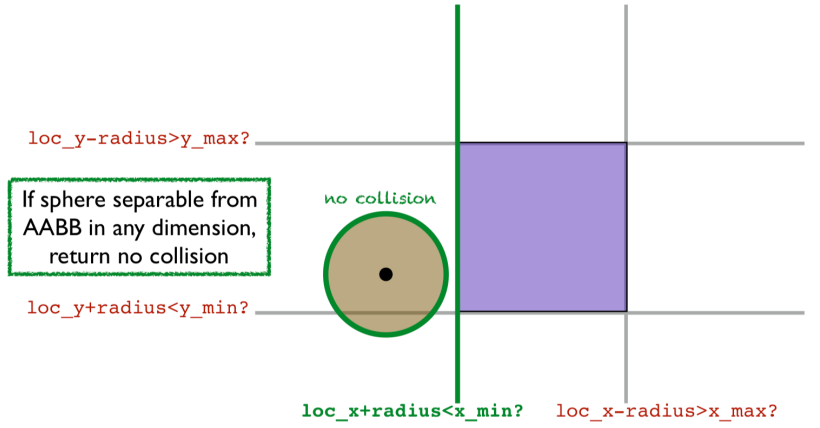
As with 2D collision detection, **axis-aligned bounding boxes** (AABB) are the quickest algorithm to determine whether the two game entities are overlapping or not. This consists of wrapping game entities in a non-rotated (thus axis-aligned) box and checking the positions of these boxes in the 3D coordinate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

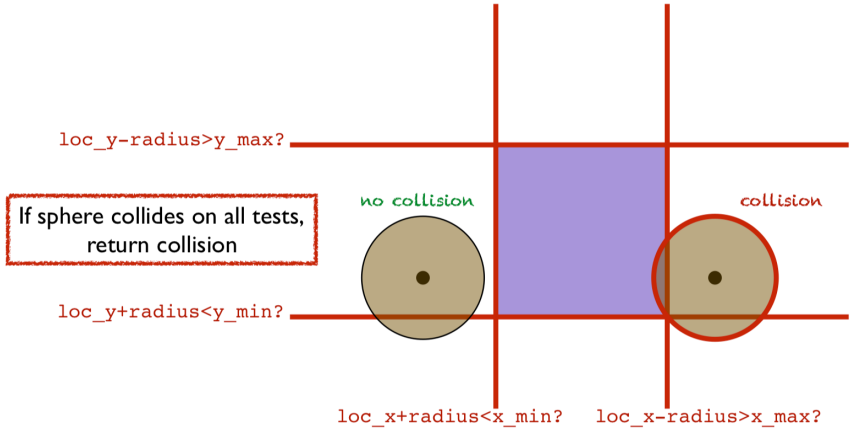


The **axis-aligned constraint** is there because of performance reasons. The overlapping area between two non-rotated boxes can be checked with logical comparisons alone, whereas rotated boxes require additional trigonometric operations, which are slower to calculate.

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

Sphere Obstacle Collision

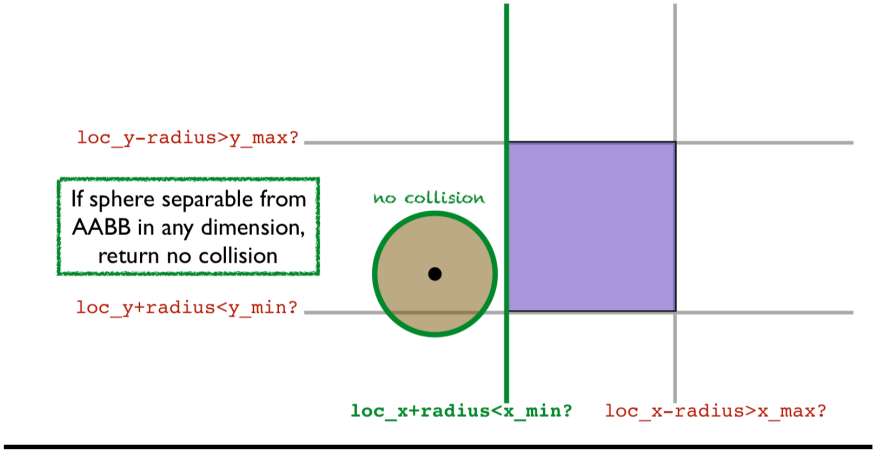


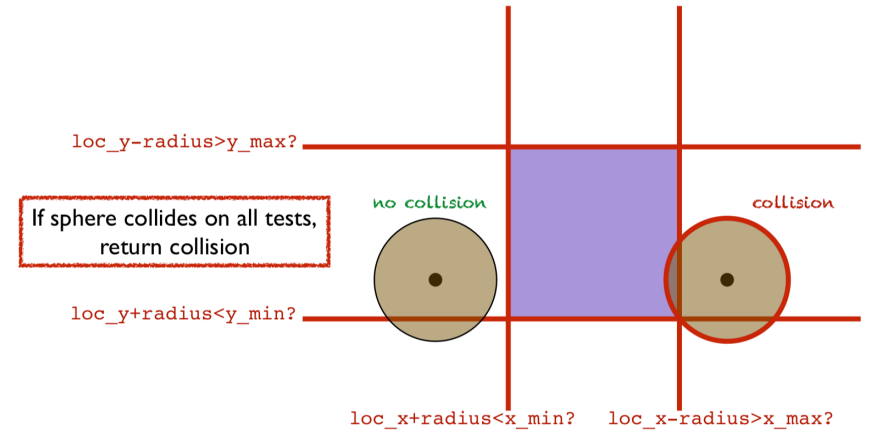


kineval_collision.js

kineval_collision.js

```
64  function traverse_collision_forward_kinematics_link(link,mstack,q) {
65
66      /* test collision FK
67      console.log(link);
68      */
69      if (typeof link.visual !== 'undefined') {
70          var local_link_xform = matrix_multiply(mstack,generate_translation_matrix(link.visual.origin.xyz[0],link.visual.origin.xyz[1],link.v
71      }
72      else {
73          var local_link_xform = matrix_multiply(mstack,generate_identity());
74      }
75
76      // test collision by transforming obstacles in world to link space
77  /*
78      mstack_inv = matrix_invert_affine(mstack);
79  */
80      mstack_inv = numeric.inv(mstack);
81
82      var i;
83      var j;
84
85      // test each obstacle against link bbox geometry by transforming obstacle into link frame and testing against axis aligned bounding box
86      //for (j=0;j<robot_obstacles.length;j++) {
87      for (j in robot_obstacles) {
88
89          var obstacle_local = matrix_multiply(mstack_inv,robot_obstacles[j].location);
90
91          // assume link is in collision as default
92          var in_collision = true;
93
94          // if obstacle lies outside the link extents along any dimension, no collision is detected
95          if (
96              (obstacle_local[0][0]<(link.bbox.min.x-robot_obstacles[j].radius))
97              ||
98              (obstacle_local[0][0]>(link.bbox.max.x+robot_obstacles[j].radius))
99          )
100                 in_collision = false;
101         if (
102             (obstacle_local[1][0]<(link.bbox.min.y-robot_obstacles[j].radius))
103             ||
104             (obstacle_local[1][0]>(link.bbox.max.y+robot_obstacles[j].radius))
105         )
106                 in_collision = false;
107         if (
108             (obstacle_local[2][0]<(link.bbox.min.z-robot_obstacles[j].radius))
109             ||
110             (obstacle_local[2][0]>(link.bbox.max.z+robot_obstacles[j].radius))
111         )
112                 in_collision = false;
```





kineval_collision.js

```
49    kineval.poseIsCollision = function robot_collision_test(q) {
50        // perform collision test of robot geometry against planning world
51
52        // test base origin (not extents) against world boundary extents
53        if ((q[0]<robot_boundary[0][0])||(q[0]>robot_boundary[1][0])||(q[2]<robot_boundary[0][2])||(q[2]>robot_boundary[1][2]))
54            return robot.base;
55
56        // traverse robot kinematics to test each body for collision
57        // STENCIL: implement forward kinematics for collision detection
58        //return robot_collision_forward_kinematics(q);
59
60    }
```

STENCIL: Check each link for collision with spherical obstacles

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_rrt_connect.js

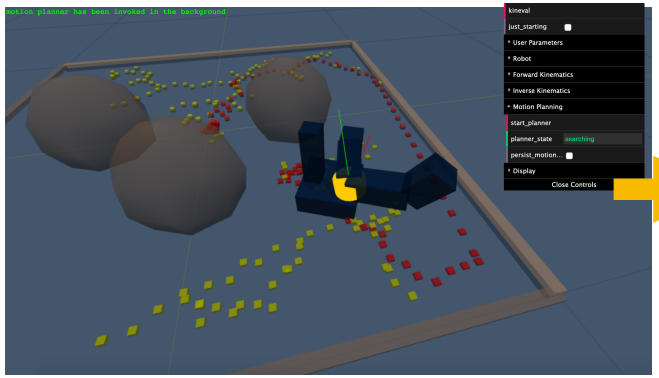
kineval_rrt_connect.js

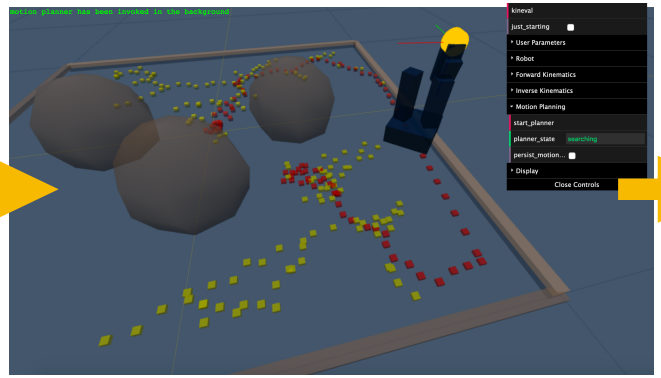
```
132  function robot_rrt_planner_iterate() {
133
134      var i;
135      rrt_alg = 1;  // 0: basic rrt (OPTIONAL), 1: rrt_connect (REQUIRED)
136
137      if (rrt_iterate && (Date.now()-cur_time > 10)) {
138          cur_time = Date.now();
139
140      // STENCIL: implement single rrt iteration here. an asy
141      //   is used instead of a for loop to avoid blocking an
142      //   in the browser.
143      //
144      //   once plan is found, highlight vertices of found pa
145      //     tree.vertices[i].vertex[j].geom.material.color =
146      //
147      //   provided support functions:
148      //
149      //   kineval.poseIsCollision - returns if a configuration is in collision
150      //   tree_init - creates a tree of configurations
151      //   tree_add_vertex - adds and displays new configuration vertex for a tree
152      //   tree_add_edge - adds and displays new tree edge between configurations
153      }
154
155  }
```

IMPLEMENT `robot_rrt_planner_iterate()` AS A SINGLE ITERATIVE STEP OF THE RRT PLANNING ALGORITHM. INCLUDE ANY ADDITIONAL HELPER FUNCTIONS IN THIS FILE

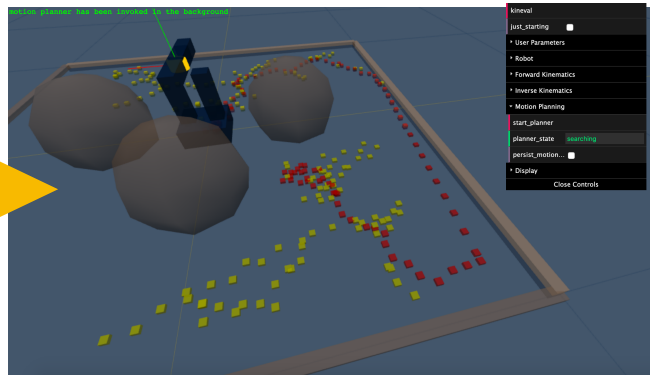
Rapidly-exploring Random Trees (RRT)

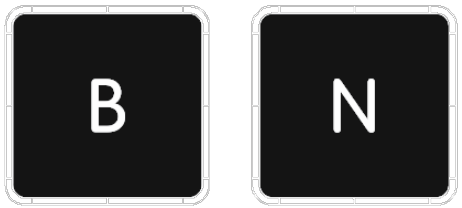
Arbitrary Initial Configuration





Collision Free Path to Home





Backward Step along Motion Plan

Forward Step along Motion Plan