15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 7:

The World Map

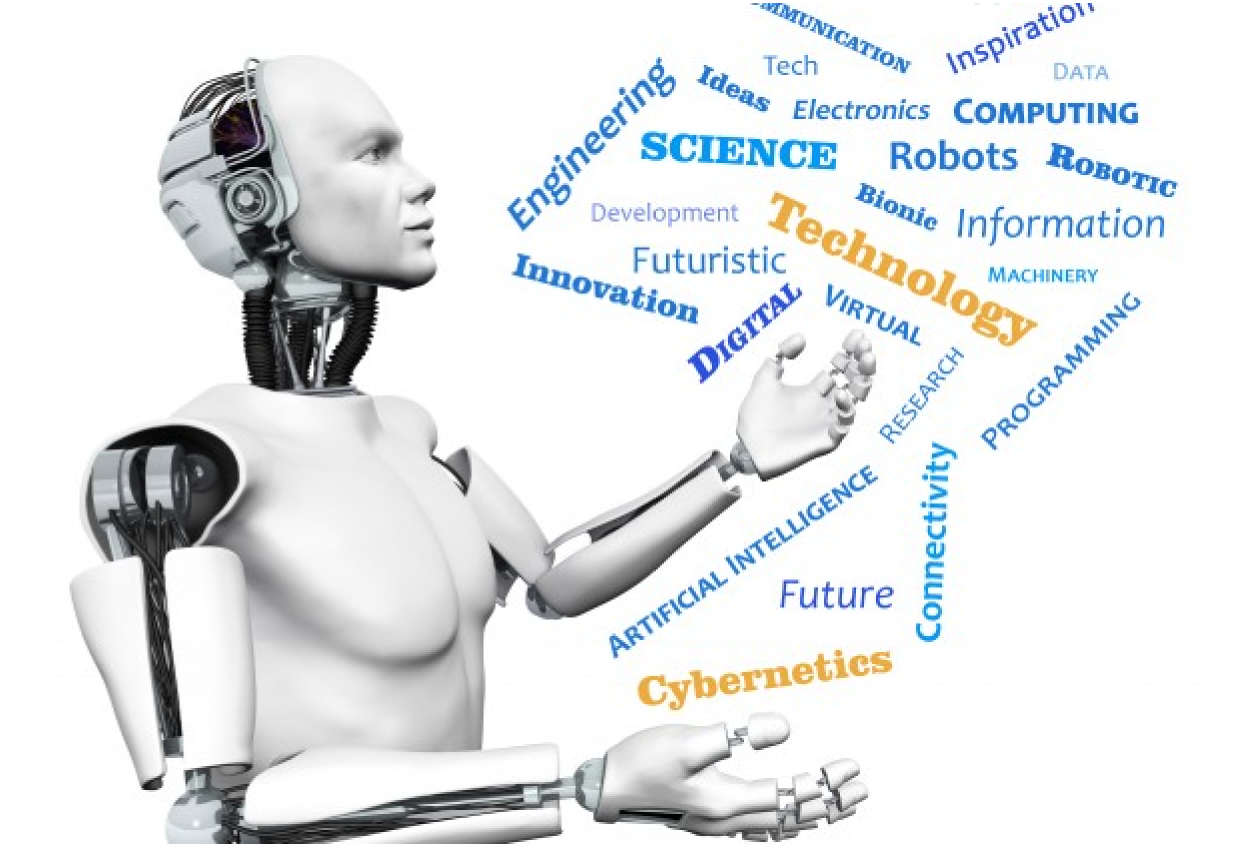


Image from http://www.futuristgerd.com/2015/09/10

Outline

- Why have a world map?
- What's in Cozmo's native world map?
- Cozmo localization
- Object pose: quaternions
- Designing our own world map
- Obstacle detection

Why Have A World Map?

- Represent objects available to the robot.
- Landmarks to be used for localization.
- Obstacle avoidance during path planning.

What's In Cozmo's Native World Map?

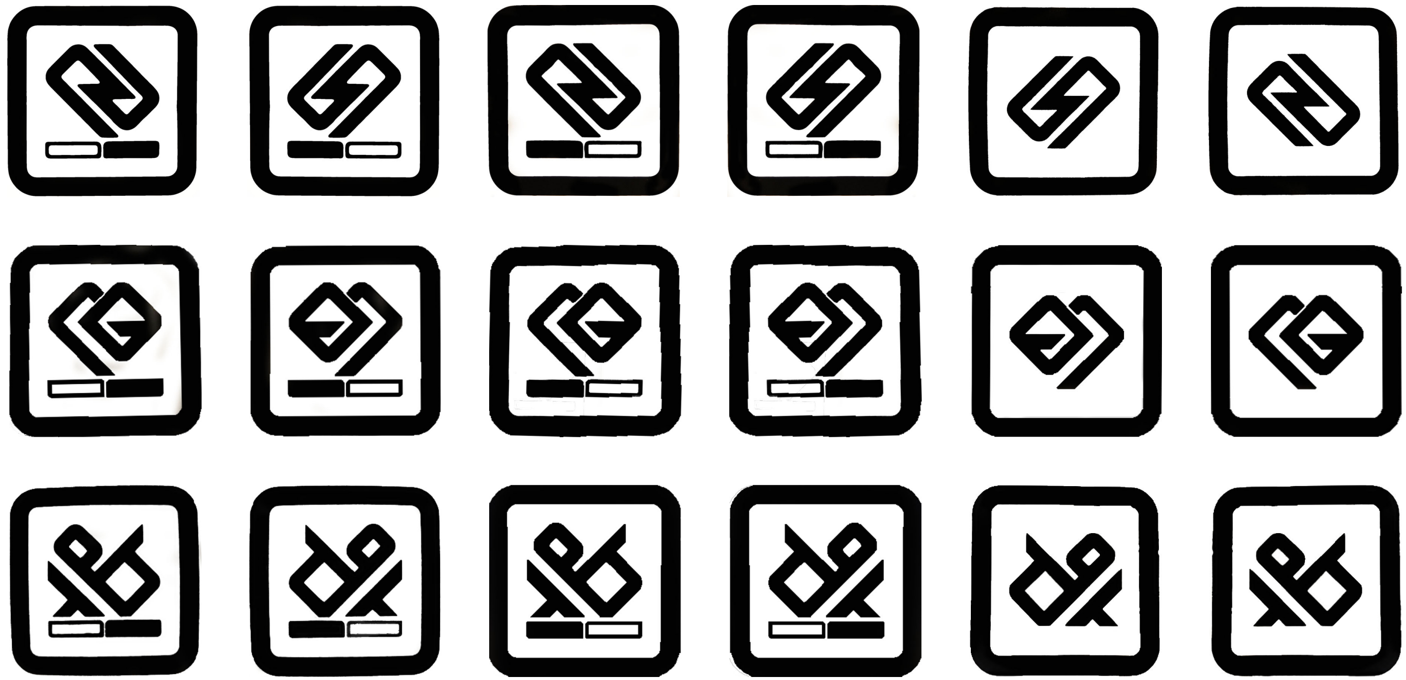
- Cozmo himself
- The light cubes, once seen
- The charger, once sensed or seen
- Faces that have been detected
- Custom markers that have been detected
- To access the world map:
  - robot.world is a cozmo.world.World object
  - components: robot.world.light_cubes, etc.

Light Cube Markers

“Paperclip” **1:**

“Anglepoise Lamp” **2:**

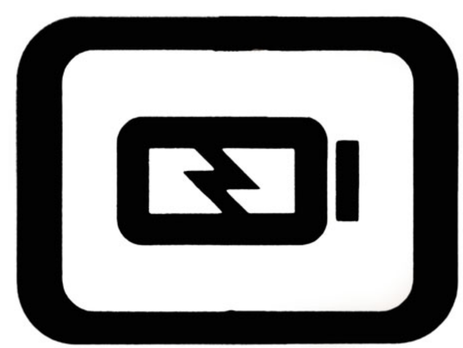
“Deli Slicer” **3:**



0° 90° 180° 270° bot top

The Charger

- Base frame is front lip; marker in back.
- robot.is_on_charger is True or False
- Robot won't move while on the charger



Custom Markers

Origin ID

- The robot's origin_id starts at 1.
- Every time Cozmo is picked up and put down, he may get a new origin_id value.
- Landmarks can pull him back to an old id.

- Object origin_id's start at -1 (invalid).
- Every time the robot sees an object, that object's origin_id is updated to match the robot's.
- Object poses are only valid if their origin_id matches the robot's.

Cozmo's Localization

- The cubes serve as visual landmarks that contribute to Cozmo's localization.
- The charger also contributes, if Cozmo has seen the marker.
- When Cozmo is on the charger, he knows exactly where he is.
- If a cube changes position, did the cube move, or did Cozmo move?
  - Cozmo knows when he has moved.

Object Pose

- The robot, cubes, and charger have a *pose* attribute that is an instance of cozmo.util.Pose.
- robot.pose.position is (x,y,z) coordinates.
- robot.pose.rotation is complicated:
  - a <u>quaternion</u> gives the full 3D pose
  - angle_z gives the orientation about the z-axis, which is usually all you care about

Quaternions

- A quaternion q is a four-dimensional complex number (w,x,y,z) or (q_0,q_1,q_2,q_3).
- w is a point on the real axis, and x,y,z are points on the i,j,k imaginary axes.

$$\mathrm{q} = \mathrm{w} + \mathrm{x}\cdot i + \mathrm{y}\cdot j + \mathrm{z}\cdot k$$

$$i\cdot i = j\cdot j = k\cdot k = i\cdot j\cdot k = -1$$

$$i\cdot j = k \qquad j\cdot k = i \qquad k\cdot i = j$$

$$j\cdot i = -k \qquad k\cdot j = -i \qquad i\cdot k = -j$$

Quaternions and Rotations

The mathematical properties of quaternions mirror those of 3D rotations: multiplication is not commutative!

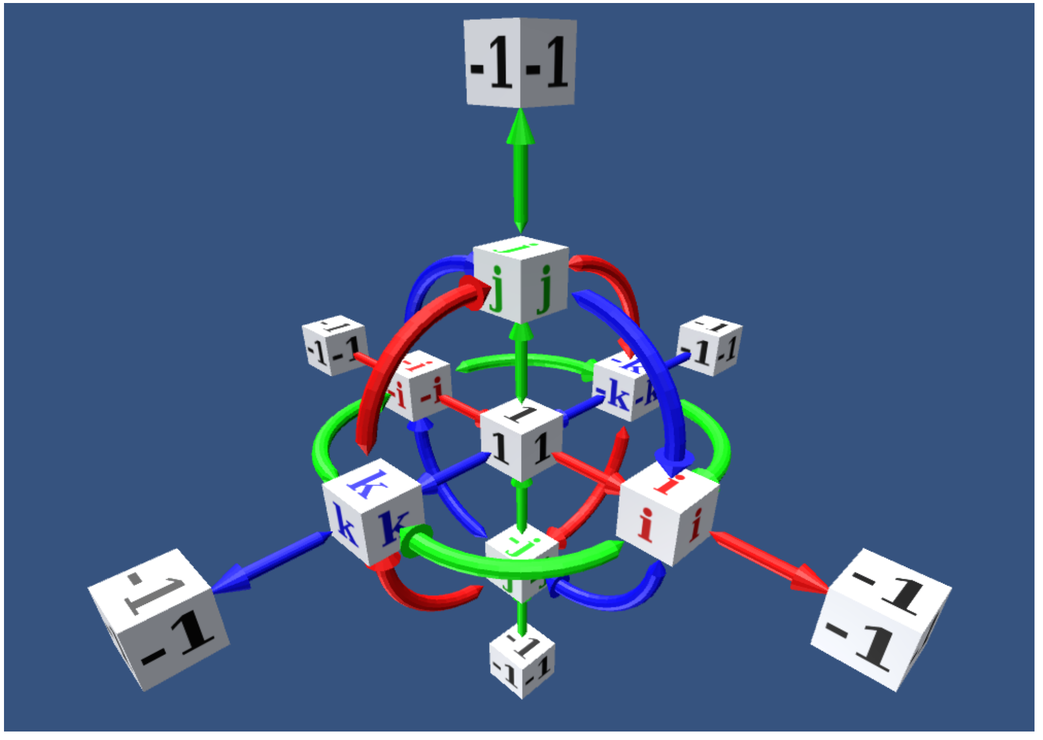


Image source: https://aha.betterexplained.com

Magnitude of a Quaternion

$$q = w + x \cdot i + y \cdot j + z \cdot k$$

$$\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

For pure rotations we want $\|q\| = 1$.

Quaternions as Poses

- Quaternions describe *rotations* in terms of an axis of rotation and an angle θ.
- Think of a pose as a rotation from the world reference frame (z up, x forward) to the object's reference frame.
- We can also represent rotations using 4x4 transformation matrices.
- To compose rotations:
  - Multiply the transformation matrices, or
  - Multiply the quaternions

Simple Cases

- "No rotation":

  $q = (1, 0, 0, 0)$

- Rotation by θ about the z axis:

  $q = (\cos \theta/2, 0, 0, \sin \theta/2)$

- General case:
  - The magnitude of the rotation is $\sin \theta/2$.
  - The direction of rotation is indicated by distributing $\sin \theta/2$ among the i,j,k axes.
  - Real $w = \cos \theta/2$ is a normalization term.

Motion Model

- The cozmo-tools particle filter includes a motion model.
- How does it get motion information?
- Look for changes in robot.pose every few milliseconds.
- If pose changes but origin_id remains the same, we have valid motion info.
- If origin_id changed, cannot subtract poses this time. Wait for next time.

The cozmo-tools World Map

Why make our own world map?

- To support additional object types:
  - ArUco markers
  - Walls
  - Chips
  - Other cameras
  - Other robots (shared world map)
- To allow our own particle filter to maintain the map (SLAM algorithm)

The cozmo-tools World Map

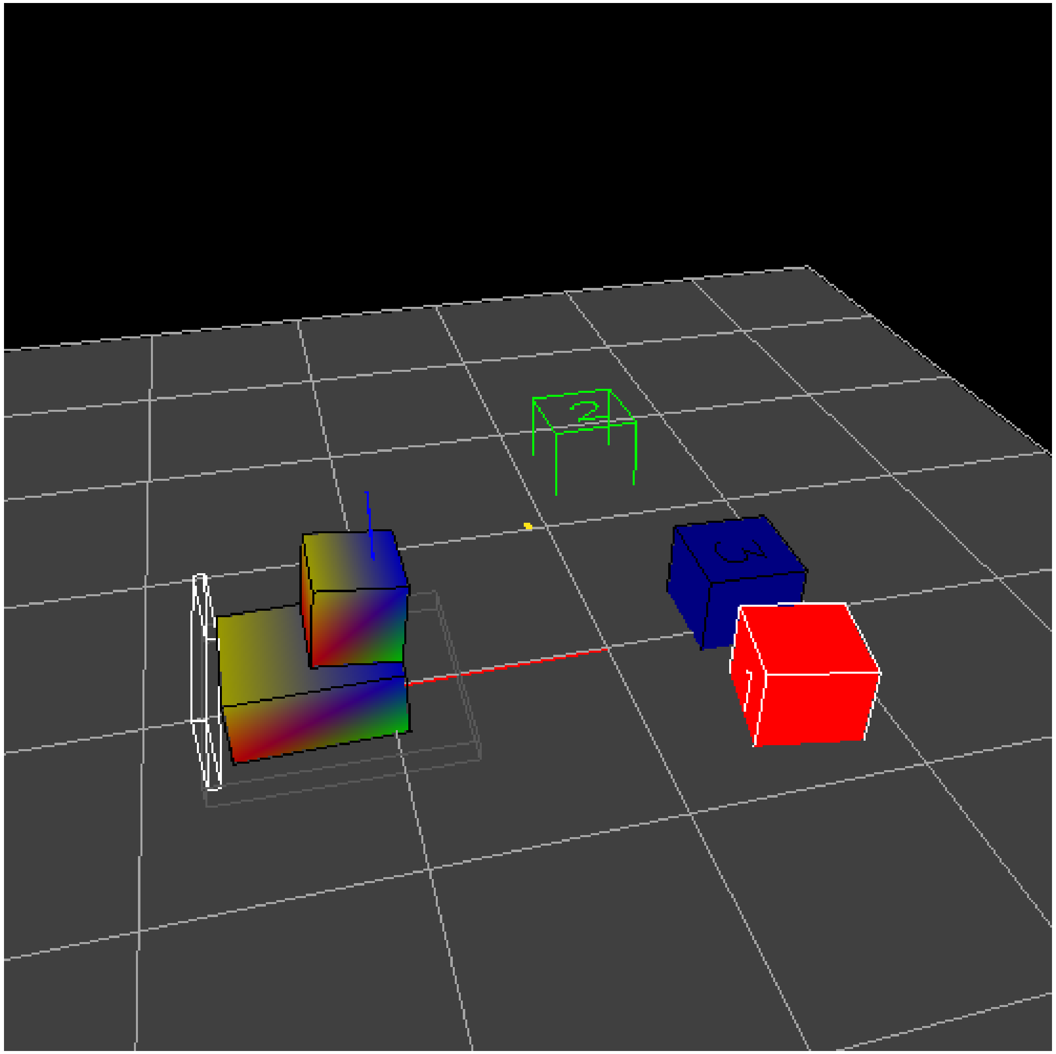
- Stored in robot.world.world_map
- Objects are in a dictionary: robot.world.world_map.objects
- Subclasses of WorldObject:
  - LightCubeObj
  - ChargerObj
  - CustomMarkerObj
  - etc.
- Type “show objects” to display.

The cozmo-tools World Map

- Robot position maintained by our particle filter, not robot.pose
  - Type “show pose” to see both

- Cubes and charger imported from their SDK repesentations:
  - wcube1, wcube2, wcube3, wcharger
  - wcube1.sdk_obj is cube1

show worldmap_viewer



Obstacle Avoidance

- Having a world map allows us to do path planning to avoid obstacles.

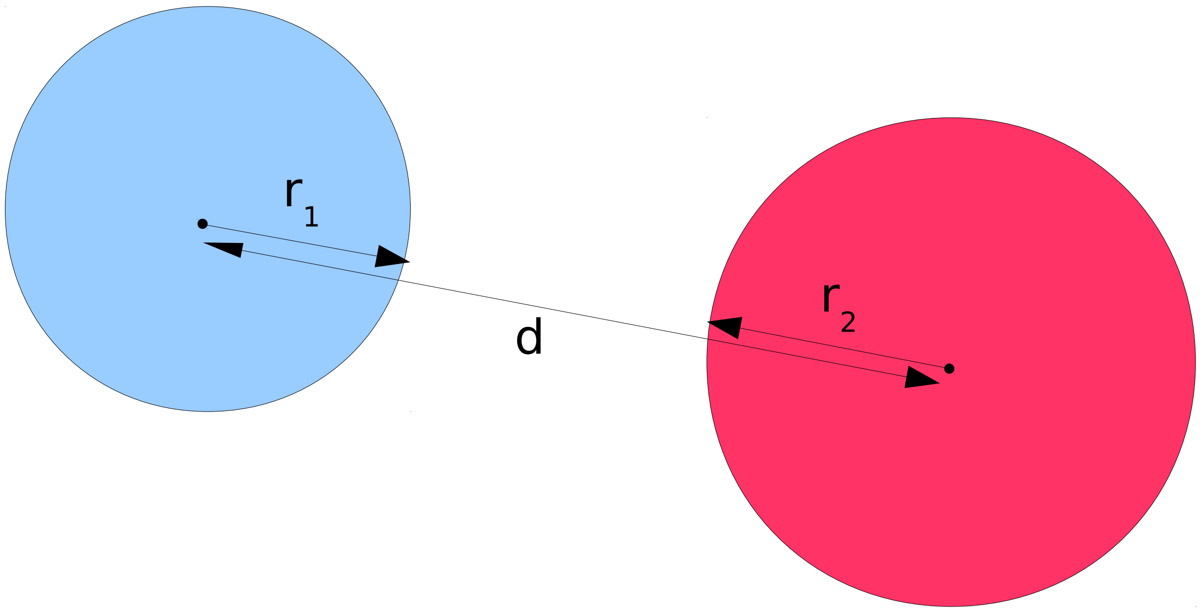
- The path planner needs a way to detect when two objects collide.

- How can we detect collisions?

Collision Detection

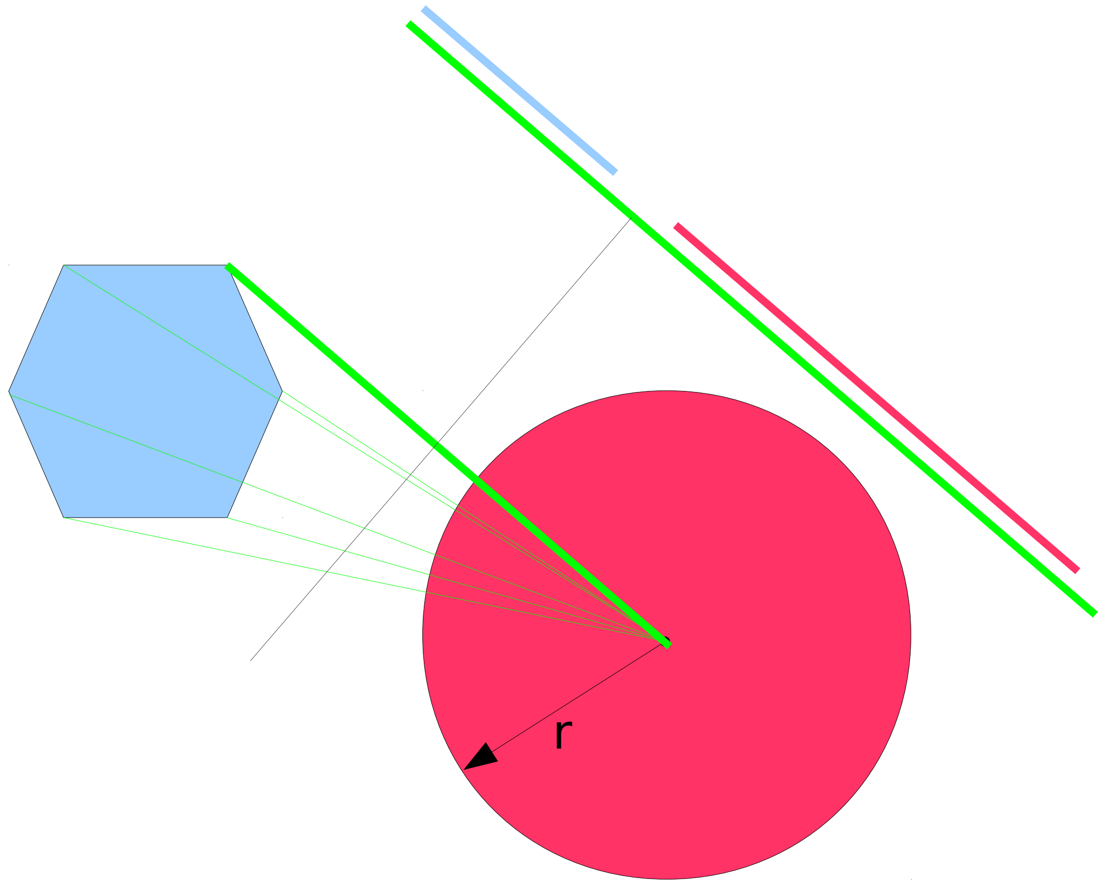
- Represent the robot and the obstacles as **convex polygons**.
- In 2D, use the Separating Axis Theorem to check for collisions.
  - Easy to code
  - Fast to compute
- In 3D, things get more complex.
  - The GJK (Gilbert-Johnson-Keerthi) algorithm is used in many physics engines for video games.

Collision Detection: Circles



- Let d = distance between centers
- Let r_1, r_2 be the radii
- No collision if $d > r_1 + r_2$

Collision Detection: Circle and Convex Polygon

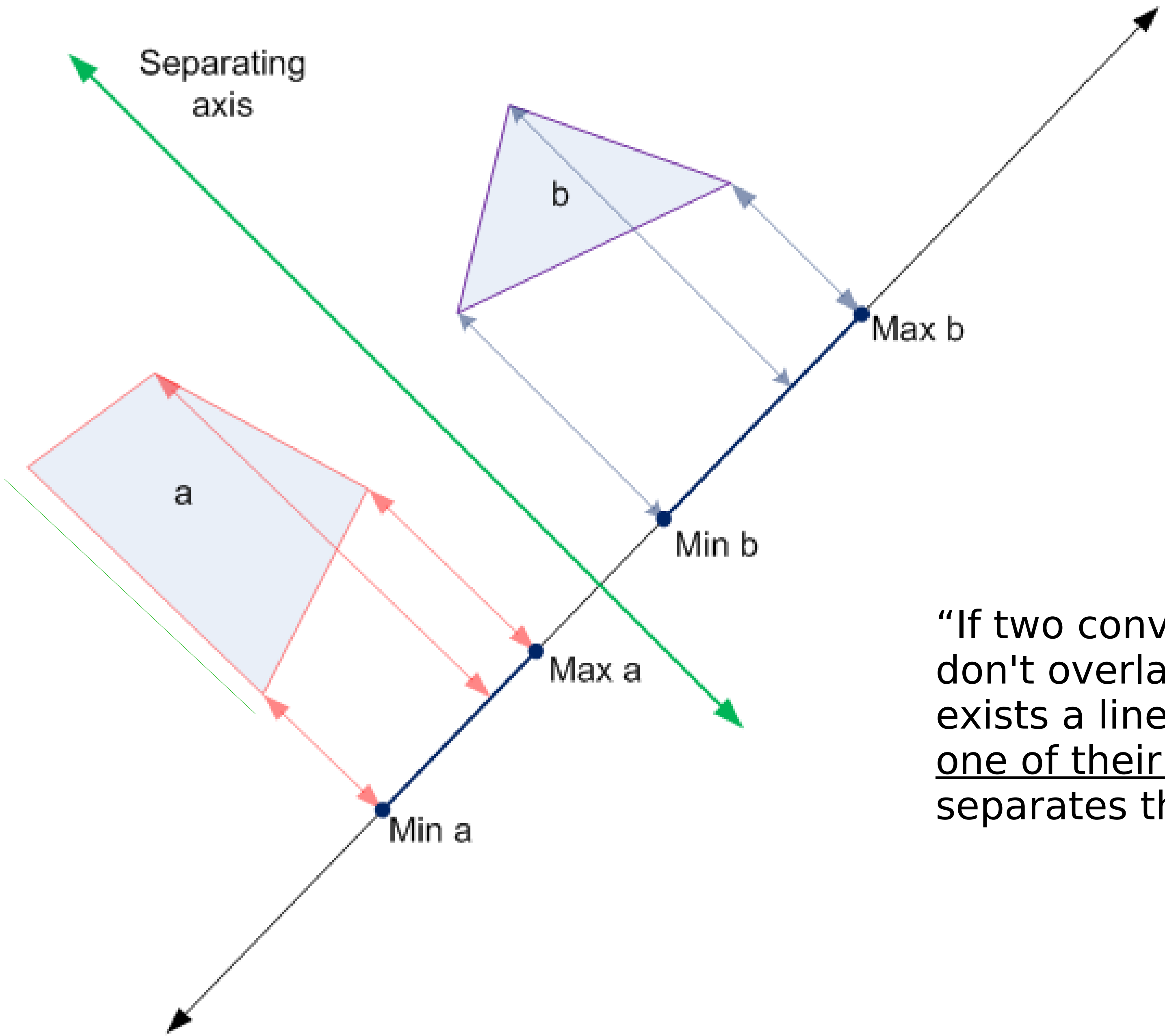


- Separating axes to check are orthogonal to the lines joining the center of the circle to the vertices of the polygon.

Collision Detection: Two Convex Polygons

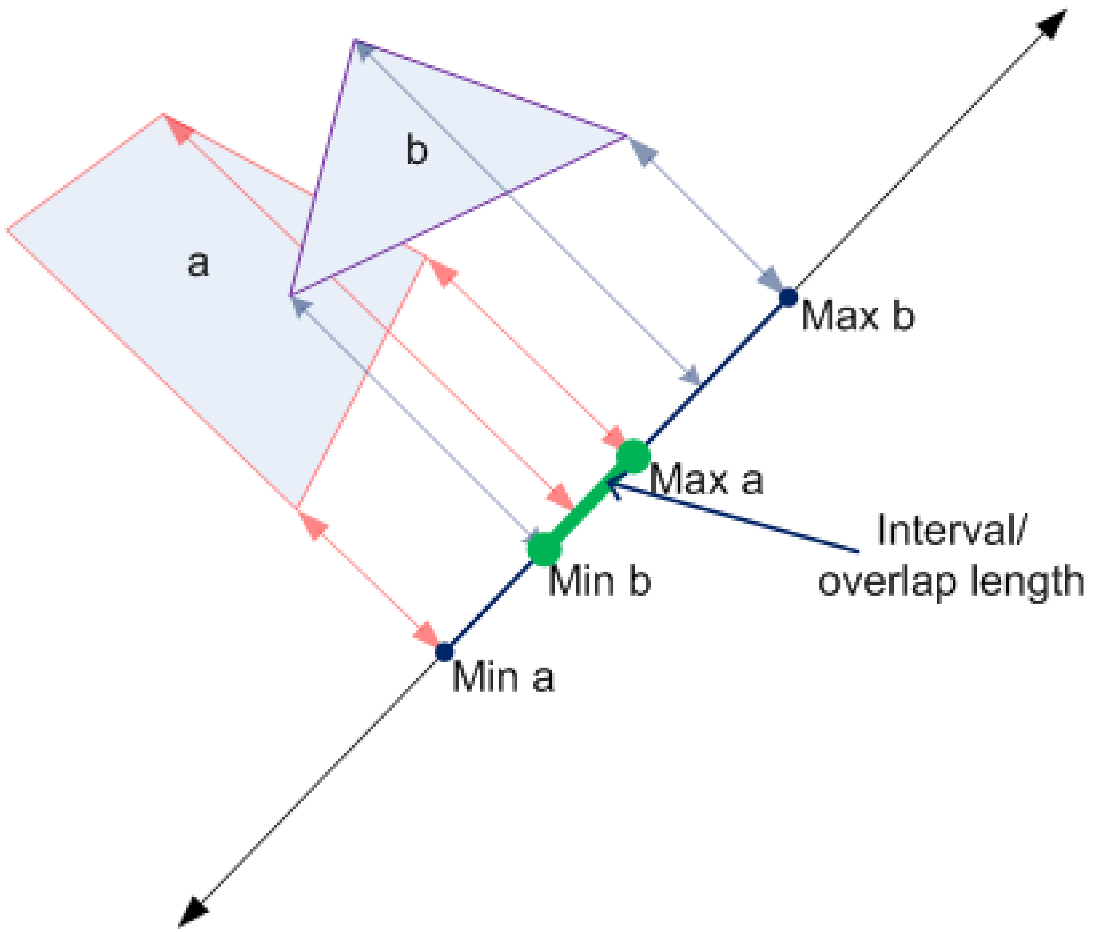
- The Separating Axis Theorem can be used to detect collisions between two <u>convex</u> polygons.
- Time is proportional to the number of vertices.
- To handle <u>non-convex</u> polygons, decompose them into sets of convex polygons and check for collisions between any two components.

Separating Axis Theorem



"If two convex polygons don't overlap, then there exists a line, parallel to <u>one of their edges</u> that separates them."

Separating Axis Theorem



Collision Detection Algorithm

We only need to find one separating axis to be assured of no collision.

```
def collision_check(poly1,poly2):
  for axis in Edges(poly1) ∪ Edges(poly2):
     base = pependicular_to(axis)
     proj1 = project_verts(poly1, base)
     proj2 = project_verts(poly2, base)
     if not overlap(proj1,proj2):
        return False
  return True
```

How To Build A World Map

- SLAM: Simultaneous Localization and Mapping algorithm.
- Each particle stores:
  - a hypothesis about the robot's location
  - a hypothesis about the map, e.g., a set of landmark identities and locations.
- Particles score well if:
  - Landmark locations match the sensor values predicted by the robot's location.