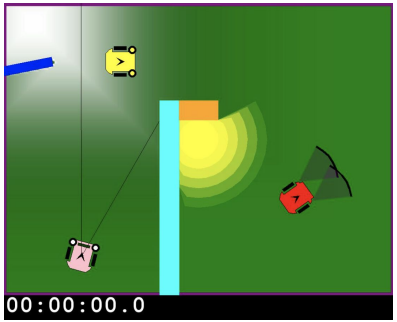
Cheat sheet for **aitk robots**

github.com/ArtificialIntelligenceToolkit/aitk



Features

- A lightweight Python mobile robotics simulator
- Explore wheeled robots with range, camera, light, and smell sensors
- Design worlds with walls, bulbs, and food
- Suitable for the classroom and research
- Creates reproducible experiments
- Easy to integrate with existing machine learning and AI systems

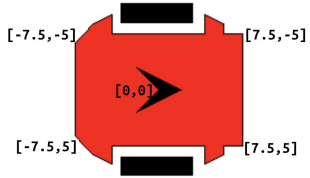
Creating Worlds and Robots

A world is a rectangular area with a given width and height that may contain walls, bulbs, food, and robots (see above)

- Items in the world are given coordinates, where the origin is defined to be the upper-left hand corner
- Angles are given in degrees, where 0 is east, and angles increase in the counterclockwise direction

```
from aitk.robots import World, Scribbler
world = World(width=200,height=150)
# add_wall(color, x1, y1, x2, y2)
world.add_wall("blue",0,35,25,30,box=False) # angled wall
world.add_wall("cyan",80,50,90,150) # box is default
world.add_wall("orange",90,50,110,60)
# add_bulb(color, x, y, z, brightness)
world.add_bulb("yellow",100,70,0,75.0)
# add_food(x, y, pixel_std_dev), by default is white
world.add_food(10, 10, 50)
bot1 = Scribbler(x=150,y=100,a=35)# red is default color
bot2 = Scribbler(x=40,y=130,a=75,color="pink")
bot3 = Scribbler(x=60,y=30,a=0,color="yellow")
world.add_robot(bot1)
world.add_robot(bot2)
world.add_robot(bot3)
world.watch() # see the simulated world
```

Equipping Robots with Sensors



A robot is defined by a bounding box, with the origin at the center. Sensors are placed relative to this bounding box.

RangeSensors return distances in cm and may have a width
LightSensors return brightness [0,1]; light is blocked by walls
SmellSensors return reading [0,1]; odor spreads around walls
Cameras return images that include walls, bulbs, food, and robots

```
from aitk.robots import RangeSensor, LightSensor, SmellSensor, Camera
# red robot has two range aensors with width like InfraRed sensors
bot1.add_device(RangeSensor(position=(6,-6),width=57.3,max=20,name="left-ir"))
bot1.add_device(RangeSensor(position=(6,6),width=57.3,max=20,name="right-ir"))

# pink robot has semll sensors and a camera
bot2.add_device(SmellSensor(position=(6,-6),name="left-smell"))
bot2.add_device(SmellSensor(position=(6,6),name="right-smell"))
bot2.add_device(Camera())
# robots can also maintain state information, for example a timer
# could be used to ensure that a particular action is repeated N times
bot2.state["timer"] = 0

# yellow robot has two light sensors
bot3.add_device(LightSensor(position=(6,-6),name="left-light"))
bot3.add_device(LightSensor(position=(6,6),name="right-light"))
```

Robot Movement

Set targets for translation and rotation:
- Use range [-1,1]
- Positive translation is forward, negative is back
- Positive rotation is left, negative is right

```
robot.move(translation, rotation)
```

Set velocity targets individually:

```
robot.translate(translation)
robot.rotate(rotation)
```

Reverse the current targets:

```
robot.reverse()
```

Halt the robot:

```
robot.stop()
```

Or set motor speeds for wheels in range [-1,1]:

```
robot.motors(left_spd, right_spd)
```

Accessing Sensors & State

Access sensors by name (string) or by index (integer) in the order that they were added:

```
robot[sensor_name]
robot[sensor_index]
```

Get `RangeSensor` data:

```
robot[item].get_distance()
```

Get `LightSensor` data:

```
robot[item].get_brightness()
```

Get `SmellSensor` data:

```
robot[item].get_reading()
```

Get `Camera` data:

```
robot[item].get_image()
```

Access robot state information by key (string):

```
robot.state[key]
```

Robot Controllers

A robot controller is a function that:
- Takes a single parameter: either world or robot
- Returns `True` to end simulation immediately
- Checks state and sensors to choose move
- Does <u>not</u> use loops

The simulation repeatedly executes the controller multiple times per second.

```
def controller(robot):
    """Wander and avoid obstacles"""
    if robot.stalled:
        return True
    v = robot["left-ir"].get_max()
    if robot["left-ir"].get_distance()<v:
        robot.move(0.1, -0.3)
    elif robot["right-ir"].get_distance()<v:
        robot.move(0.1, 0.3)
    else:
        robot.move(1, random()-0.5)
```

Other Robot Data & Methods

Determine velocity or whether stalled:

```
robot.get_velocity()
robot.stalled #True when stuck
```

When food is close, the robot may eat it:

```
robot.eat() #returns True when eaten
```

Create a speech bubble:

```
robot.speak(string)
```

Position the robot in world or find its position:

```
robot.set_pose(x, y, a)
robot.get_pose() #returns (x,y,a)
```

Running the Simulator

There are three ways to run the simulator.

1. Indefinitely:

```
world.run(function, …)
```

2. For a time limit:

```
world.seconds(seconds, function, …)
```

3. For a step limit:

```
world.steps(steps, function, …)
```

You must specify either a single function that takes a world, or a list of functions that each take a robot.

Running Experiments

After a run concludes you may reset the robots and world to their saved configuration:

```
world.reset()
```

Set a new random seed for the simulator:

```
world.set_seed(seed)
```

Set a new random position for a robot:

```
robot.set_random_pose()
```

Record a run:

```
recorder = world.record()
```

Execute the simulator as fast as possible:

```
world.run(function, real_time=False)
```

Watch the recorded experiment:

```
recorder.watch()
```

Save the recorded experiment as an animated GIF or mp4:

```
recorder.save_as(filename)
```