```
"""
File that handles  the states of the robot
"""

from math import sin,cos,pi,sqrt

import Postman
import Trig

class RobotState():

    def __init__(self,padding):
        self.update()
        self.size = self.getActualSize() + padding
        self.maxSpeed = 1

    def update(self):
        self.laserScan = updateLaser()
        self.x, self.y = updatePosition()
        self.direction = updateDirection()

    def getSize(self):
        return self.size

    def getActualSize(self):
        """
            :return the actual size of robot
        """
        return 0.40000000596046448

    def getLaserScan(self):
        return self.laserScan

    def getPosition(self):
        return self.x, self.y

    def getDirection(self):
        return self.direction

    def getMaxSpeed(self):
        return 1

    def getCorners(self, x, y, angle, cornerNumber):
        """
            :param x, y: the position you want to investigate
            :param angle: the angle the robot would be
            :param cornerNumber: 0 for upper left, 1 for lower left, 2 for lower right, 3 f
or upper right
            :return: the x and y position of the corner
        """
        len = self.size / sqrt(2)

        # 0 is upperLeft, 1 is lowerLeft, 2 is lowerRight and 3 is upperRight
        cornerNumber = (pi / 2) * (cornerNumber % 4) + (pi / 4)

        cx = x + len * cos(angle + cornerNumber)
        cy = y + len * sin(angle + cornerNumber)

        return cx, cy

    def getLaserLengthFromIndex(self, index):
        """
            Returns the length of a laser from an index, if outside range, assume length of
 robot at least
        """
        if (index < 0 or index > 270):
            return (self.size / 2)
        else:
            return self.laserScan['Echoes'][index]

    def getLaserLength(self, angle):
        """

            Returns the laserlength of a certain angle,
             if the angle is between two lasers, a weighted average is used
             to derive a more exact distance
             uses: radToLaserFloat
        """
        index = Trig.radToLaserFloat(angle, self.direction)
        leftIndex = int(index)
        rightIndex = leftIndex + 1

        if (rightIndex < 0 or leftIndex > 270):
            dist = sqrt(2) * (self.size / 2)
        elif leftIndex == 270:
            dist = self.laserScan['Echoes'][270]
        elif rightIndex == 0:
            dist = self.laserScan['Echoes'][0]
        else:
            diffL = abs(index - leftIndex)
            diffR = abs(index - rightIndex)

            distL = self.laserScan['Echoes'][leftIndex]
            distR = self.laserScan['Echoes'][rightIndex]

            dist = (distL * diffR) + (distR * diffL)

        return dist


def updateLaser():
    return Postman.getLaser()

def updateDirection():
    """
        :return: The heading of the robot as radians from straight up
    """
    unitVector = Postman.getBearing()
    ux=unitVector['X']
    uy=unitVector['Y']
    return Trig.angleToPoint(0, 0, ux, uy)

def updatePosition():
    """
        :return: the position of the robot in the global positionsystem
    """
    data = Postman.getPose()
    x = data['Pose']['Position']['X']
    y = data['Pose']['Position']['Y']
    return x,y
```