Home Assignment 4
Due date:20.06.19 23:30

Introduction:

The HydroCamel is the first and only Israeli Autonomous Underwater Vehicle (AUV). It was developed at Ben-Gurion University and is fully operational. An AUV can perform many tasks such as: archeological surveying, underwater search and rescue missions, geological research and mines detection.

Figure 1 The HydroCamel II AUV

In this work you will write a simulation for AUV mine detection. The AUV has a sonar system that can find mines, the sonar has a triangle shaped Field of View (FOV), with an 'R' range and 'θ' angle to each side:

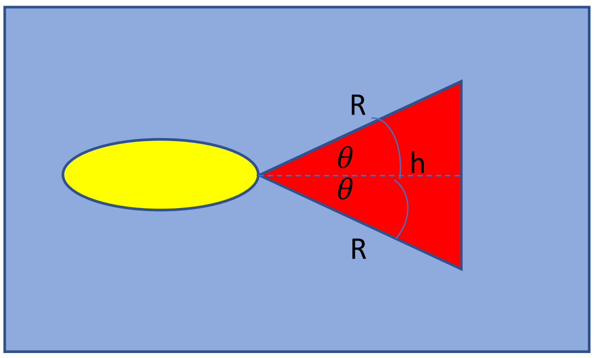


Figure 2 Illustration of the sonar FOV

Assignments:

You are asked to implement the HydroCamel() class, this class will act as a simulator for an AUV. Given an initial position and velocity the simulator will move the AUV in the map. Using the sonar model, the simulation will check if there are any mines in range (all of the mines location is given to the simulator).
The simulator will receive a set of velocities and durations (each velocity has its own duration) and will advance the vehicle accordingly.

For your convenience the HW is divided into two assignments:

1. Implement the HydroCamel class as detailed in the next section. In this section you can assume that the AUV heading is zero, i.e. the AUV's velocity is [[0, Vx]] or more generally, could be of the form: [[0, Vx1], [0, Vx2],…,[0, Vxn]]
2. Make adjustments to the HydroCamel class, so that it could receive velocities in various directions. (the Direction will be a multiply of 45 degrees)

HydroCamel class:

The HydroCamel() class has the following methods:

Methods:	Input	Output	Description
get_mines()	-	A list of tuples. Each tuple holds the coordinates (y_i, x_i) of found mines. List should be sorted.	Returns the position of all the mines that the AUV has found
get_sonar_fov()	-	A dictionary. The keys of the dictionary are tuples of the (y_i, x_i) coordinates and the value should be Boolean True	Returns all the current (y_i, x_i) coordinates of the map which are in range for the sonar
display_map()	-	-	Display the current map.
get_heading()	-	Returns the Direction of movement of the AUV in Degrees (see Tips and comments section)	The heading will be between 0-360. With respect to the x and y axes of the map.
set_course()	Velocity as list of lists. Duration as list of integers	-	Receive new values for the velocity and duration properties. Append the new values to the current ones
time_step()	-	-	Propagate the simulation by one step (one second) if duration >0
start()	-	-	Activate the simulation and run until duration has ended

The inputs to the constructor are:

name	Type	Description
sonar_range	int	The sonar range in cells $3 \leq R \leq 9$
sonar_angle	int	The sonar Field of View angle to each direction in Degrees (see Figure 2). $15 \leq \theta \leq 85$
map_size	(int, int)	The (Height , Width) of the map
initial_position	(int, int)	The (y, x) initial position on the map
velocity	[[int, int]]	The Horizontal and Vertical velocity in cell/sec [[Vy,Vx]]. len(velocity)=len(duration)
duration	[int]	The number of seconds to travel in the desired velocity. len(velocity)=len(duration)
Mines_map	2D list	A matrix with dimensions equal to Map_size. The matrix will hold a value of one if a mine is present in that location, and zero otherwise.

Other Requirements:

The list of found mines (output of get_mines()) should return a sorted list. The list should be sorted according to:

- First priority x coordinate
- Then according to y coordinate

You may **not use** a built-in sort function. This will be checked! Instead you may implement whatever sorting you wish.

Success Criteria:

your work will be tested in the following manner:

1. Creating an instance of the HydroCamel() class
2. Check that the FOV was calculated correctly
3. Advancing the AUV in the x-axis direction and testing the mines and FOV dictionary.
4. Advancing the AUV in various directions and testing the mines and FOV dictionary.

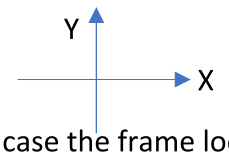
Tips and comments:

- Read the instructions twice before you begin to code!!
- You can assume that the AUV/sonar won't reach the edge of the map
- Mines that are currently seen don't have to be colored Red on the map.
- You can use numpy array instead of list
- Use the display_map() method for your testing. You can use the matplotlib library.
- Rotation matrix:

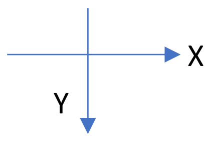
  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

  A rotation matrix is recommended for the rotation of the sonar field of view to the direction of movement.
  https://en.wikipedia.org/wiki/Rotation_matrix

- You should use the four quadrants inverse tan -> atan2(y,x) to calculate the heading angle. But notice that atan2(y,x) returns the angle in the following frame:

  But in our case the frame looks like this:

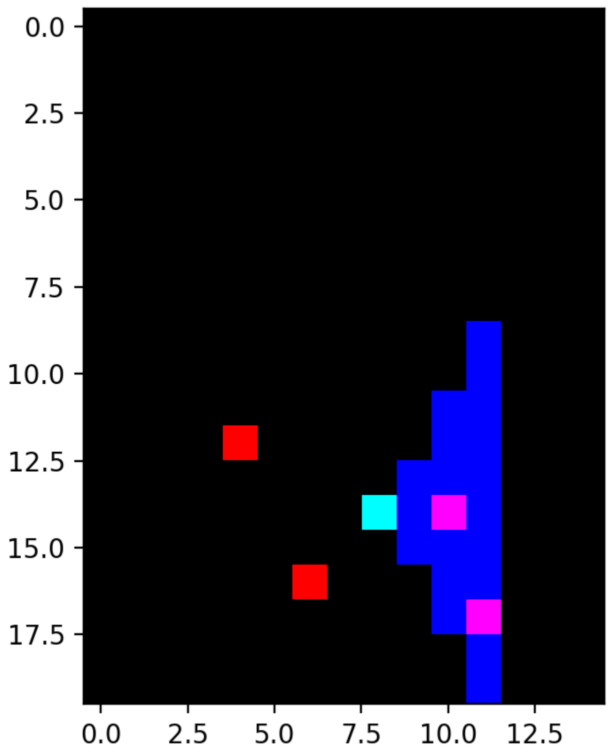
- You are advised to solve a variant of the Point In Polygon (PIP) problem https://en.wikipedia.org/wiki/Point_in_polygon
- Document your code - for yourself, and for good practice. Your comments should be written in **English only.**
- You should submit a **single .py** file through the Moodle website with the name of YOUR_ID_NUMBER.py where YOUR_ID_NUMBER should be your id number.
- The run time is limited to 20 minutes.
- Avoid copying!

Additional comments (updated 13.6):

- When the simulation is initialized, the AUV has a heading of '0' degrees. And the sonar can detect objects at that location.
- The AUV movement and sonar scans are discreet, which means that it can only detect objects after each movement and won't detect objects that were in its way. For example, if the velocity was [10,10] it can miss a lot of mines in between each step.
- Notice that each step of the simulation isn't supposed to take a "real" second. It only "simulate" a duration of one second.

Example 1:

The following image was created with these parameters:
Sonar_range = 6
Sonar_ang = 60
Initial_position = (14, 1)
Velocity = [[0, 1]]



get_mines() will return – [(12, 4), (16, 6), (14, 10), (17, 11)]
Get_sonar_fov() will return – {(14, 8): True, (13, 9): True, (14, 9): True, (15, 9): True, (11, 10): True, (12, 10): True, (13, 10): True, (14, 10): True, (15, 10): True, (16, 10): True, (17, 10): True, (9, 11): True, (10, 11): True, (11, 11): True, (12, 11): True, (13, 11): True, (14, 11): True, (15, 11): True, (16, 11): True, (17, 11): True, (18, 11): True, (19, 11): True}

Example 2:

The following input for the constructor will cause the AUV to have a move in different directions:
Sonar_range = 5
Sonar_ang = 30
Initial_position = (10, 10)
duration = [2, 2, 2, 2]
Velocity = [[2, 2], [-2, -2], [0, 2], [2, 0]]

After the first step, Get_sonar_fov() will return – {(12, 12): True, (13, 13): True, (14, 13): True, (15, 13): True, (13, 14): True, (14, 14): True, (15, 14): True, (16, 14): True, (13, 15): True, (14, 15): True, (15, 15): True, (14, 16): True}

The AUV will move in the following way: