CMSC828F Project 1: Obstacle avoidance

Build a simple robot simulator in Matlab. The task is to perform obstacle avoidance.

Inputs to the program:

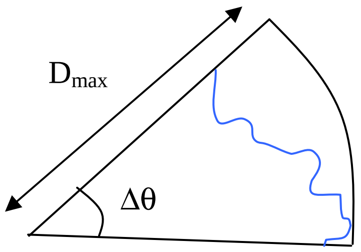
1) Map M: A map of the environment will be given to you as a black and white image, with the black areas (valued 0) denoting obstacles, and white areas (value 255) denoting free space.

2) Starting location x,y and robot orientation θ will be given.

3) The robot moves with a fixed speed v, which is given. Our simulator will only control the direction to begin with. Later, we will remove this restriction.

Data:
You will have to maintain the state of:

1) The robot vehicle R: position x,y, orientation θ, and an array of N sensors S. You can start with one forward pointing sensor (and maybe two side sensors).
2) Each sensor S_i will have its own position x,y and orientation θ, relative to the robot. It will also maintain a field called Value, which stores the current value of the sensor measurement. Each sensor will also have fixed properties like its angular field of view Δθ and its maximum range D_{max} (in pixels) as shown below.



Functions:

1) MoveRobot (Robot R, speed v, angular speed ω, time Δt): This function moves robot for time interval Δt with forward speed v, and changing its orientation with angular speed ω. This function must update the internal x,y, θ of the robot R to new values, and update the positions and orientations of each sensor.

2) UpdateSensorValues(Map M, Robot R): As explained in class, this function must use the map M to figure out the Value of each sensor S_i attached to the robot R. As shown in the figure above, the blue line indicates the closest surface lying in

the range of the sensor. If the average distance of this surface from the sensor is D, then the Value of the sensor should be set to:

$$\text{Value} = 0 \quad \text{if} \quad D >= \text{Dmax}$$
$$= \text{MaxVal} * (1 - x/\text{Dmax})$$

MaxVal is the maximum possible sensor value (set this to what you like, e.g., 1). This function is a simple linear function. The real response to distance is inverse-square. You can add an additional noise term if you wish.

As explained in class, useful functions to find D from the image are *bwselect* and *bwboundaries*. You can think of other creative ways to implement this.

3) ω = Control(sensorvalues) : The control function will take as input all the sensor values and returns the angular speed ω. Use simple proportional control to start with: $\omega = s*K*V_{front}$, where s = +1 or -1, K is a positive constant, and V_{front} is the value of the front sensor. If you have values of the left and right sensors, then you can choose s = +1 (turn left) if $V_{left} < V_{right}$, and s = -1 otherwise. The goal of the control mechanism is **obstacle avoidance**. You can play with the control to achieve the best results.

4) DrawRobot (Robot R, Map M): This function should draw the map using imshow, and you can use the "hold on" command to draw the robot on top of the map, using functions like plot, rectangle, or quiver (whatever you feel conveys the position and orientation of the robot appropriately).

5) Your main script should allow us to easily change the map image filename M, and other inputs like speed v, starting location x,y, and orientation of the robot. Also include a variable for specifying how long the script should be run (number of seconds). On running the main script, we should be able to see the animation of the robot moving in the environment. Shown below is an example.