# Homework Assignment 3:
# *Mapping*

**Overview**

In this assignment, you will implement the “occupancy grid” mapping technique as described in class.

**Homework submission notes**

Please mark your team number in the filenames of the files you submit – at least in the filename of the .zip file (if submitting a single .zip) or the .pdf write-up file (if submitting individual files).

**Coding style requirements**

Each function must be documented with its purpose, arguments, and outputs, like this:

```
def get_motor_speeds(lin_vel,ang_vel):
     “””Calculates Sparki motor speeds from linear and
angular velocities.
     Arguments:
          lin_vel: linear velocity in cm/s
          ang_vel: angular velocity in rad/s
     Returns:
          Left motor speed, left motor direction, right
motor speed, and right motor direction
     “””
     # calculations go here
     return left_motor_speed, left_motor_dir, ...
```

Variables should have understandable names and comments documenting their purpose. Classes and significant pieces of code should be commented as well.

Include a README file which explains how to run your code.

**Assignment components**

1. Implement a class to hold the occupancy grid. (20 points)

The ObstacleMap class (previously called OccupancyMap) contains the "true" map loaded from an image and simulates rangefinder readings.

In this part, you will create an OccupancyGrid class that contains the probabilistic occupancy grid that will be updated over time.

i. The OccupancyGrid should maintain a Numpy matrix containing the log odds of occupancy for each cell. The log odds should be initialized to zero for each cell.

ii. Write a function to compute the probability of occupancy of each cell. The probability of occupancy for cell m can be obtained by:

$$p(m) = 1 - 1 / (1 + \exp(\text{ log odds of m }))$$

Note that, while you could use a for loop to compute this for each cell, Numpy functions (like numpy.exp) can be used to compute on each cell in parallel – this will be much faster.

Copy the draw() function from ObstacleMap to your OccupancyGrid class and modify it to draw the **probabilities** of the occupancy grid.

You should also modify mapping.py so that it draws the OccupancyGrid instead of the ObstacleMap. When initialized to all zeros (50% probability) the occupancy grid should be all grey.

2. Create a class for the rangefinder sensor model. (40 points)

In this part, you will create a Rangefinder class that computes the "inverse sensor model," i.e. the log odds of occupancy for each cell given a rangefinder measurement.

i. Create the Rangefinder class and include parameters for the cone width (radians) and obstacle width (centimeters).

When a rangefinder measurement is received, compute the log odds of occupancy for each cell in the occupancy grid using the cone model described in class:

ii. Let x,y be the coordinates of the cell in the sensor's reference frame, and let z be the rangefinder measurement. We first convert to polar coordinates (r,$\theta$):

$r = \mathrm{sqrt}(x^2+y^2)$
$\theta = \mathrm{arctan2}(y,x)$

iii. Compute the log odds for each cell in the grid.

L_occ = log odds for occluded region (e.g. 5)
L_free = log odds for free region (e.g. -5)
L_0 = prior for log odds outside cone region (e.g. 0)

If $-\beta/2 \le \theta \le \beta/2$ and $z - \alpha/2 \le r \le z + \alpha/2$, the log odds is L_occ
If $-\beta/2 \le \theta \le \beta/2$ and $0 \le r < z - \alpha/2$, the log odds is L_free
Otherwise, the log odds is L_0

Note that, again, this will be very slow if you compute the log odds for each cell sequentially using for loops.

I have provided a function meshgrid in RobotLib.Math which computes x and y coordinates for the grid and optionally transforms them with a given transformation matrix. This will enable you to operate on all cells in parallel using the normal comparison operators and other Numpy functions such as numpy.bitwise_and and numpy.bitwise_or. If A is a matrix and B is a matrix of Booleans with the same size as A, Numpy will allow you to access all the elements where B is True like this:

```
A[B] = 5
```

That will set A to 5 wherever B is true.

It is fine to implement it with a double for loop, but if you want it to run at an acceptable frame rate, you will need to write it using Numpy operations as described above.

iv. Write a function which performs the Binary Bayes Filter update on each cell in the occupancy grid, given the current rangefinder measurement and robot pose.

For one cell of the grid, if L_meas is the log odds of occupancy according to the measurement, L_{t-1} is the current log odds of occupancy and L_0 is the prior log odds of occupancy, the new log odds L_t is:

$$L_t = L_{t-1} + L_{meas} - L_0$$

3. Update and draw the occupancy grid as the robot navigates the scene. (10 points)

i. In the update loop in mapping.py, update the occupancy grid with the current rangefinder reading (real or simulated) and current robot pose.

4. Evaluate and improve your mapping system. (30 points)

In this step, you will evaluate your system and present results in a write-up document. Submit the write-up as a Word doc or PDF.

i. Modify the get_first_hist() function in ObstacleMap so that it simulates rangefinder noise. You can use uniform noise on the range [-n,n] or you could use Gaussian noise (see the numpy.random module). Is the mapping algorithm able to cope with this noise? What happens as you increase the noise?

ii. Test different cone widths and obstacle thicknesses for the rangefinder model. How does this affect the mapping results?

iii. Try your mapping system using the real robot. Does it seem to work? Why or why not?