

# CS-475 Assignment 3

## Particle Filter

Michael Maravgakis  
maravgakis@csd.uoc.gr

Release date: 26/03/2025  
Deadline: 06/04/2025

### 1 Overview

In this assignment you will learn how to localize your robot by using the Particle Filter algorithm. Particle filter can work even if the noise is not gaussian (unlike KF and EKF) but it is more computationally demanding. In the previous assignments you used a GPS module and IMU rotation measurements to estimate the state of the robot with respect to the world frame. Now, you are provided with a custom made room and a drone that can navigate inside by using the teleop node. Instead of GPS you will use the provided height map of the room and displacement measurements of the drone.



Figure 1: Custom workspace + drone

## 2 Installation

In your .zip file, you will find the `assign3/` package, copy and paste it inside a catkin workspace and then download the following package:

```
$ git clone https://github.com/tahsinkose/sjtu-drone.git
```

After you've download the packages for the assignment navigate to:

```
$ cd <path>/assign3/worlds/
```

Open `new_world.world`, replace `"/home/michael/475_ws"` at lines 740 and 762 with your own `"/home/<username>/<workspace>/"`

Finally, modify the following file:

```
/sjtu-drone/include/plugin_drone.h
```

and change:

```
- #include "ignition/math4/ignition/math.hh"
```

```
+ #include "ignition/math6/ignition/math.hh"
```

Compile your workspace:

```
$ catkin_make
```

Use `$ rospack profile` to update ros file system, for ROS to find the new packages. To check if everything is fine, run:

```
$ roslaunch assign3 drone.launch
```

Gazebo and Rviz windows will open (give it some time for the first run) and you should see the drone inside the custom room from figure 1 (drone spawns at the corner). For this assignment you will need both Rviz and Gazebo to work simultaneously. Gazebo will be used for navigation and Rviz for visualization of your results. Finally the launch file opens the teleop node that will be used to move the drone in the environment. The window from figure 2 will pop up. Navigate in the environment after you take off by pressing "Z" in the popped-up window.

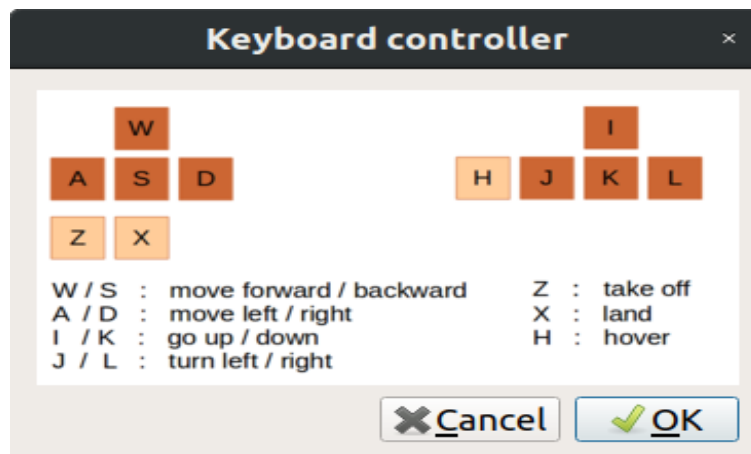


Figure 2: Teleop controls

### 3 Information and measurements

You will have to fill the code inside `particles.py` in order to implement the particle filter. The goal is your particles to estimate the 2D position (x,y) of your drone inside the room. The message that will be used in order to create, update and visualize your particles is `geometry_msgs/PoseArray`, keep  $z=1$  and  $[qx,qy,qz,qw] = [0,0,0,1]$ , only update x and y. Also make sure that your PoseArray message has always the attribute `msg.header.frame_id = 'map'`. The size of the room is (8m,15m). The measurements that are provided and can be utilized to create your algorithm are:

#### Height map

The `height_map` of the room contains the height of a specific point inside the room. So, `height_map[x_i,y_i] = Height_of_that_point`. Be careful, `x_i`, `y_i` are indices inside the grid that represent the positions of x and y in meters. The resolution of the grid is something you will find useful in order to convert meters to grid cells. The resolution is 0.01, e.g. the point (5,3) in meters is (500,300) inside the height map. In the following image I've made a visualization of the height map of the room to make things more clear. The bottom right

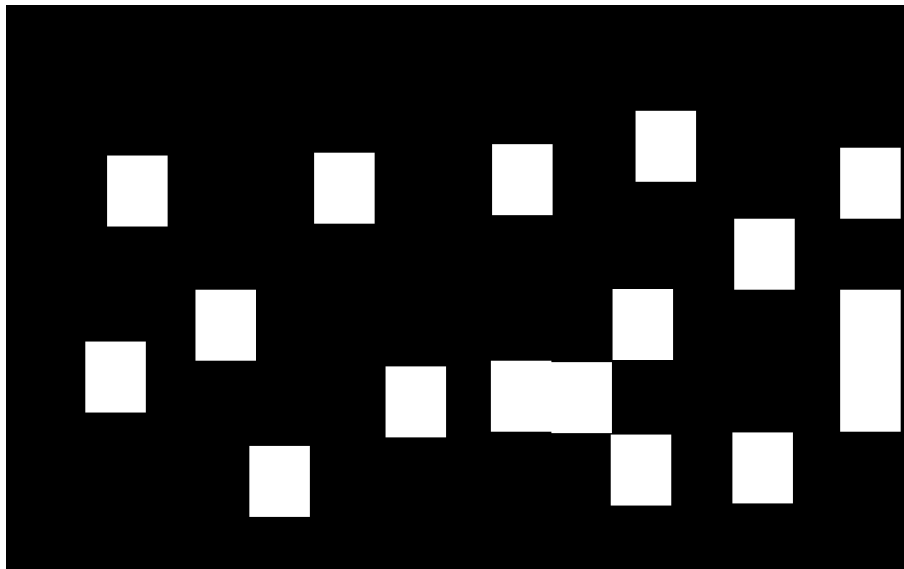


Figure 3: Height map

corner can be considered as (0,0). X-axis runs from bottom to top and Y-axis from left to right. The white pixels represent the distance from ground not equal to zero. Although the room contains only cubical boxes, note that the problem is not binary (height either one or zero), I've encoded the height from

the center of the box to the edges to decrease. This information will not affect you, it was done in order to avoid binary weights. You can imagine each box to be a mountain with its center being the top.

## Displacement

You are provided with 2 displacement measurements namely `dx`, `dy`. These represent the difference between the previous and the current position of the drone at x and y axis respectively. **IMPORTANT:** Because the orientation of the robot is not monitored avoid rotating the yaw of the robot, so use WASD to navigate around but don't change its heading.

## Range finder

The `range_finder` variable contains the measurement of a radar that can estimate (with noise) the vertical distance below the drone. Lets say that the drone flies above a box at height 6m from the ground. The height of the box is 1m, so the range finder will give you a measurement around 5m.

## Actual height

The `actual_height` variable contains information regarding the vertical distance between the drone and the ground in meters.

# 4 Implementation

The steps that you need to take are the following:

1. Initialize particles: Firstly, define your message type that will represent the particles. Then, randomly distribute all the particles inside the room. (tip: You can use `random.uniform(low_lim,high_lim)`)
2. Initialize weights
3. (Loop starts)
4. Sample particles: From the already existing particles distribution, pick randomly N of them. (tip: displace them a bit e.g. -0.1,+0.1 from their current position)
5. Update particles: Use your displacement measurements to update the position of all your particles. In case that one of new particle's position gets out of the bounds of the room you can assign to it random values.
6. Calculate weights: You'll need to find a metric that evaluates and assigns a weight to all particles given the current measurements and the height map. This metric should quantify the estimated distance vs the measured distance.

7. Resample particles: You can use the low variance resampling algorithm to eliminate some of the lower weight particles. (Probabilistic Robotics book)

## 5 Measurements

In the given code, there are some functions inside a DO NOT MODIFY block. You should use these functions to get your measurements and not subscribe to any rostopics.

1. `def create_height_map(resolution)` creates and returns the height map in an array form. (use it once as the height map does not change)
2. `def get_measurement(x, y)` Given the previous position (x,y) returns the dx and dy with some error.
3. `def find_vertical_distance(height_map, resolution)`. Given the height map and its resolution, it returns two measurements: i) The distance between the drone and the closest vertical object and ii) The height of the drone above ground.

## 6 Results

If you've implemented the algorithm correctly you will need to navigate to the room for a while in order for the algorithm to converge (not too long). In the following figure you can see that although the algorithm has not converged yet, is on the right way since it has eliminated all possible positions except the top of the boxes. This is the initial output of the algorithm if the drone is sitting on top of a box when you run the algorithm (after a few iterations):

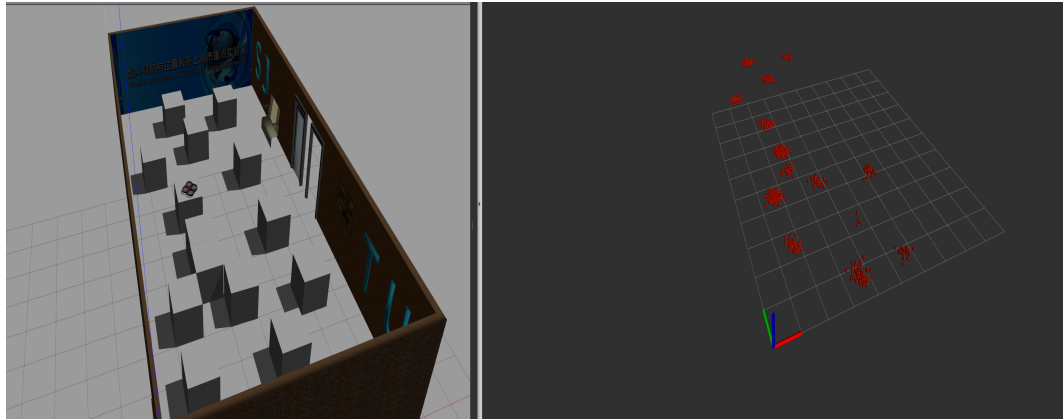


Figure 4: Initial output

If you move the robot around, the particles that can not fit will be eliminated sooner or later and the expected result during navigation will be:

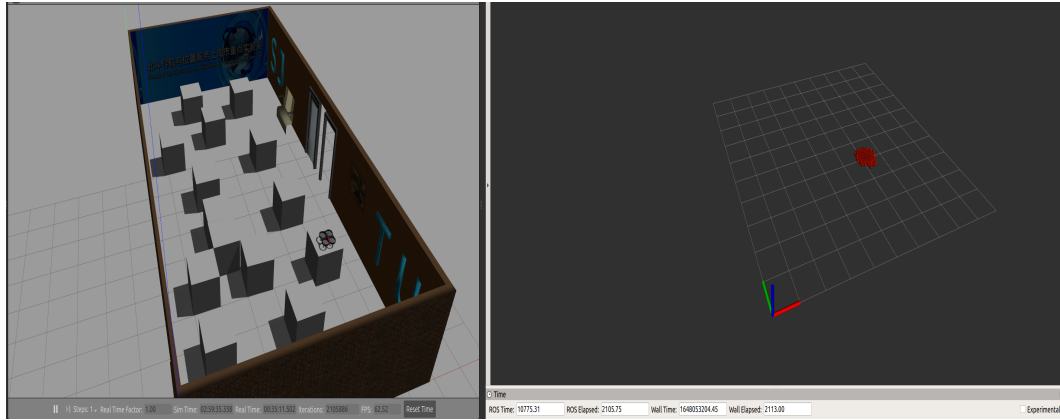


Figure 5: Converged particle filter

## 7 Submission

Sent your node (`particles.py`) attached via email at: [maravgakis@csd.uoc.gr](mailto:maravgakis@csd.uoc.gr) with subject "[CS-475] Assignment 3 submission" Don't forget to mention your name and registration number. The deadline is at **06/04/2025 23:59**