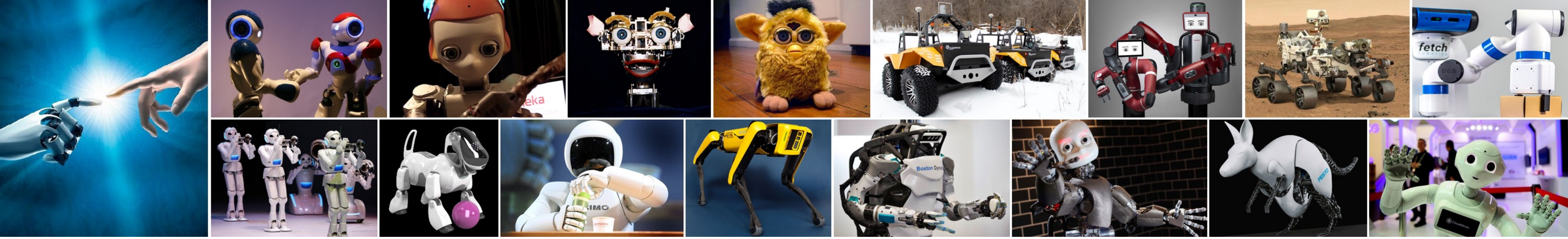


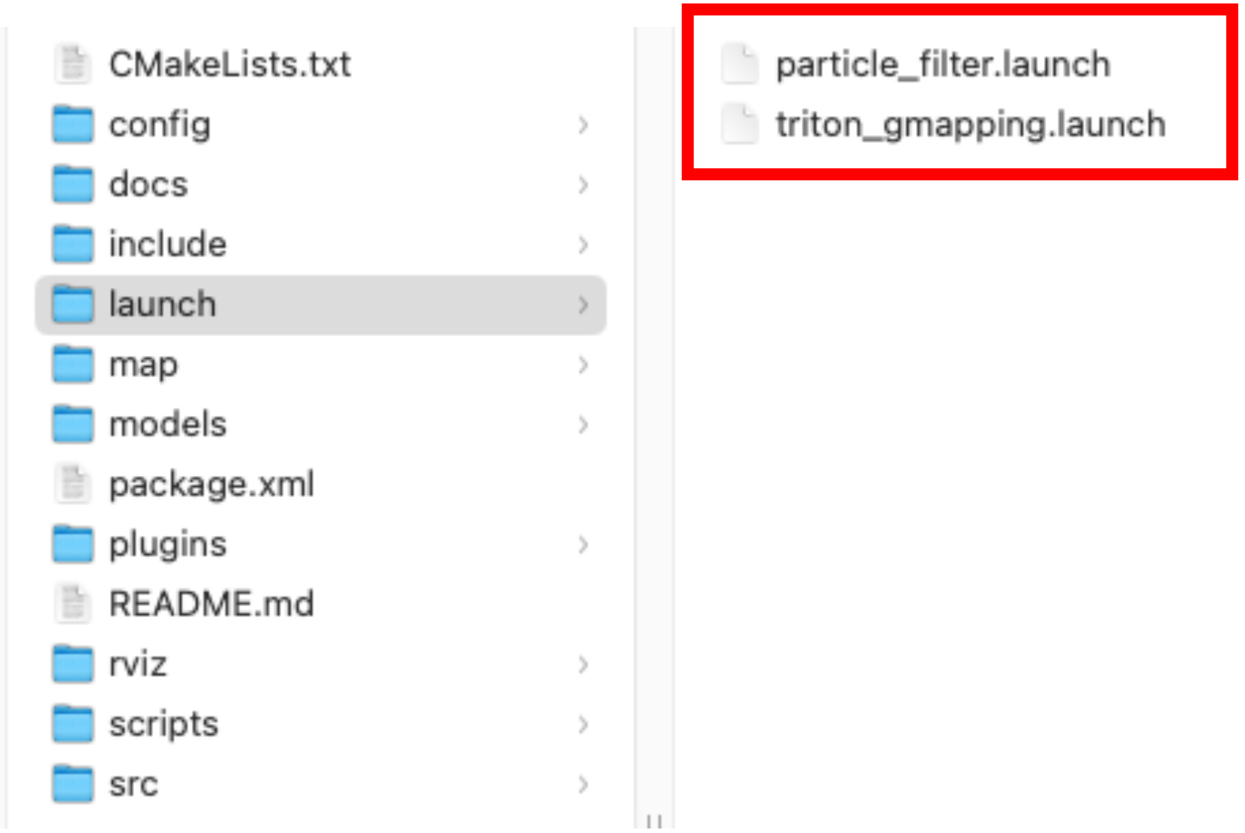
Project 3

Robot Localization by Particle Filter

Project 2 Summary

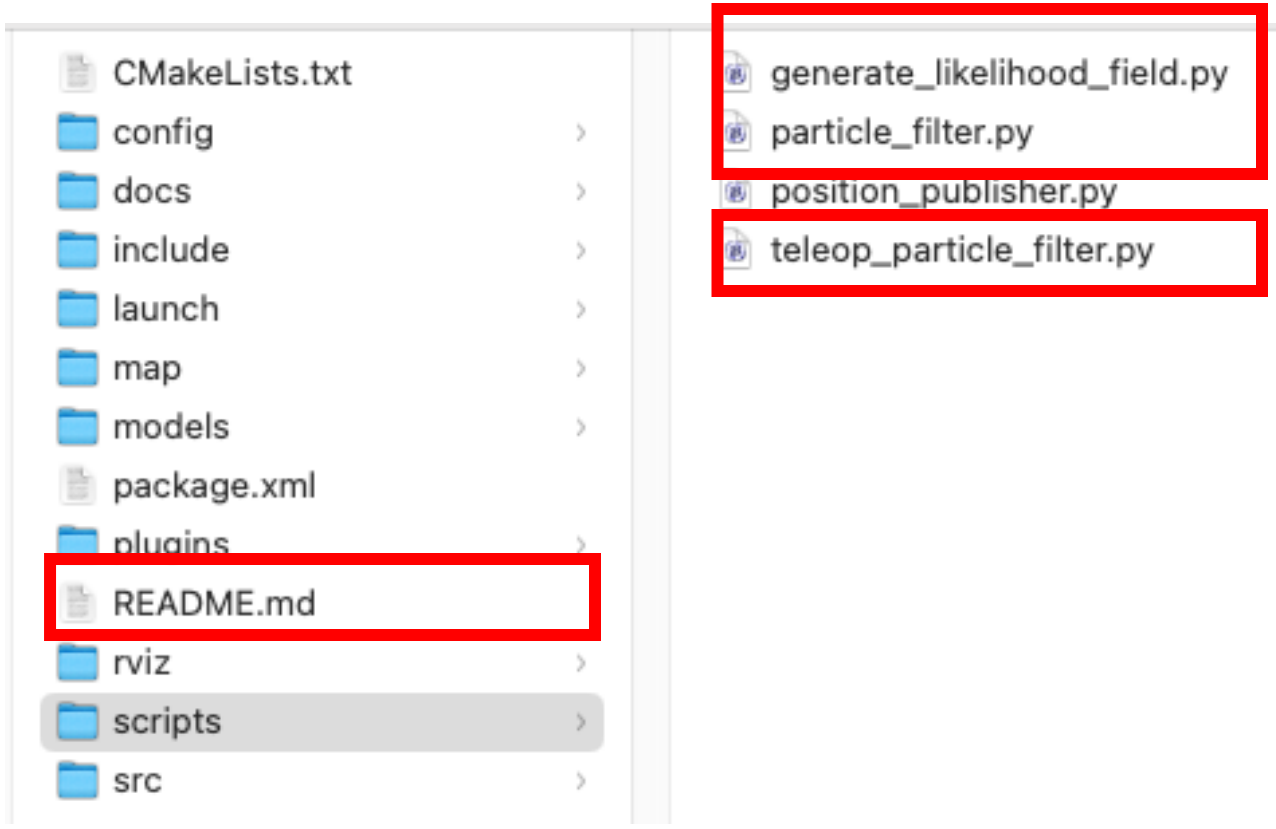
- Please put everything in a single .tar and it should be named as P2D1_peng_gao.tar
- Everything means your implementation, demo and also the stingray package, the grader needs to directly run your package and see the performance. Do NOT just submit your own code.
- The grading of the deliverable is based on the demo (functionalities) and the code (to be run and reproduce your demo).
- It is not worth to be late if you just want to improve your performance.

Package Structure



CMakeLists.txt
config
docs
include
launch
map
models
package.xml
plugins
README.md
rviz
scripts
src

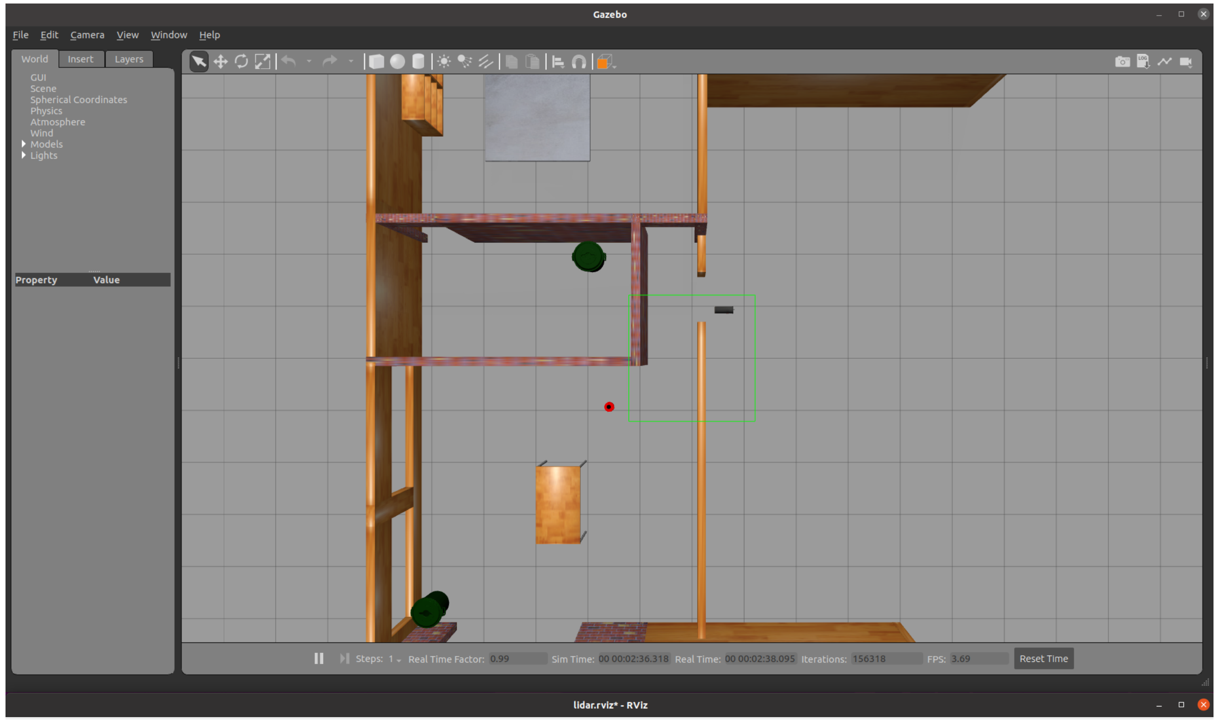
particle_filter.launch
triton_gmapping.launch

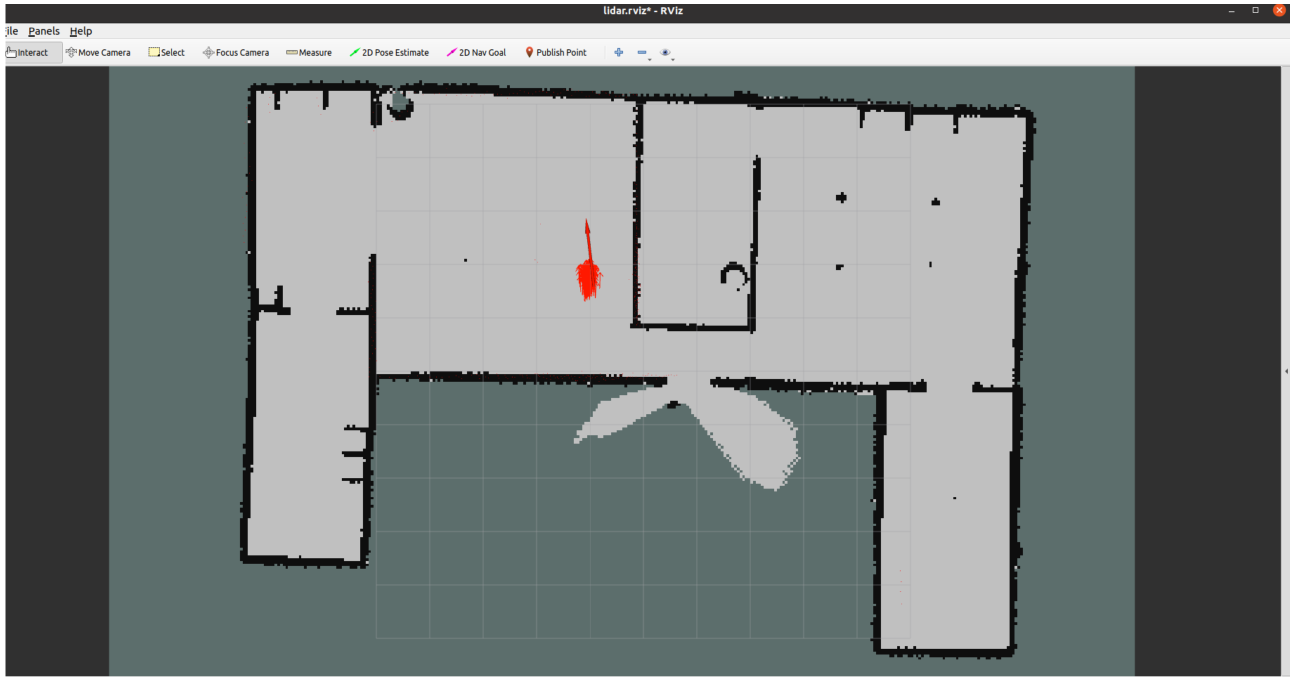


CMakeLists.txt
config
docs
include
launch
map
models
package.xml
plugins
README.md
rviz
scripts
src

generate_likelihood_field.py
particle_filter.py
position_publisher.py
teleop_particle_filter.py

Overview


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:02:36.318 Real Time: 00 00:02:38.095 Iterations: 156318 FPS: 3.69
Reset Time
lidar.rviz* - RViz

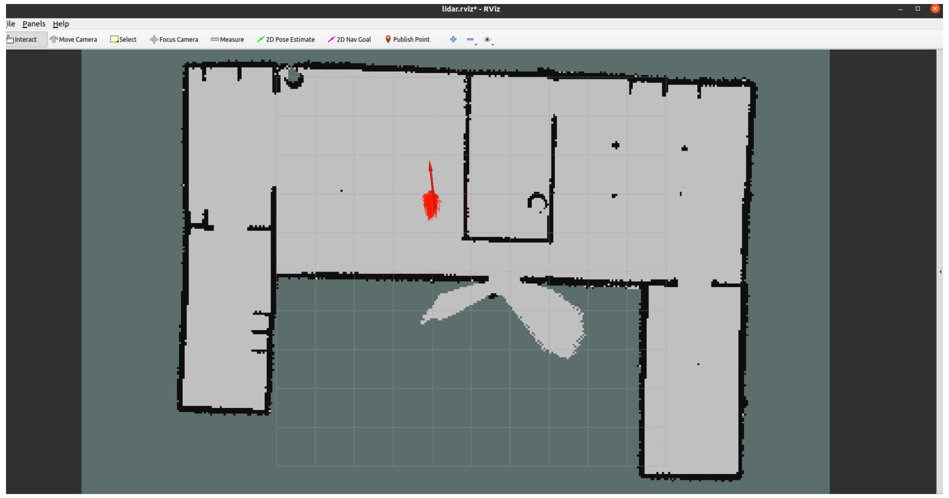

lidar.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point

Deliverable 1

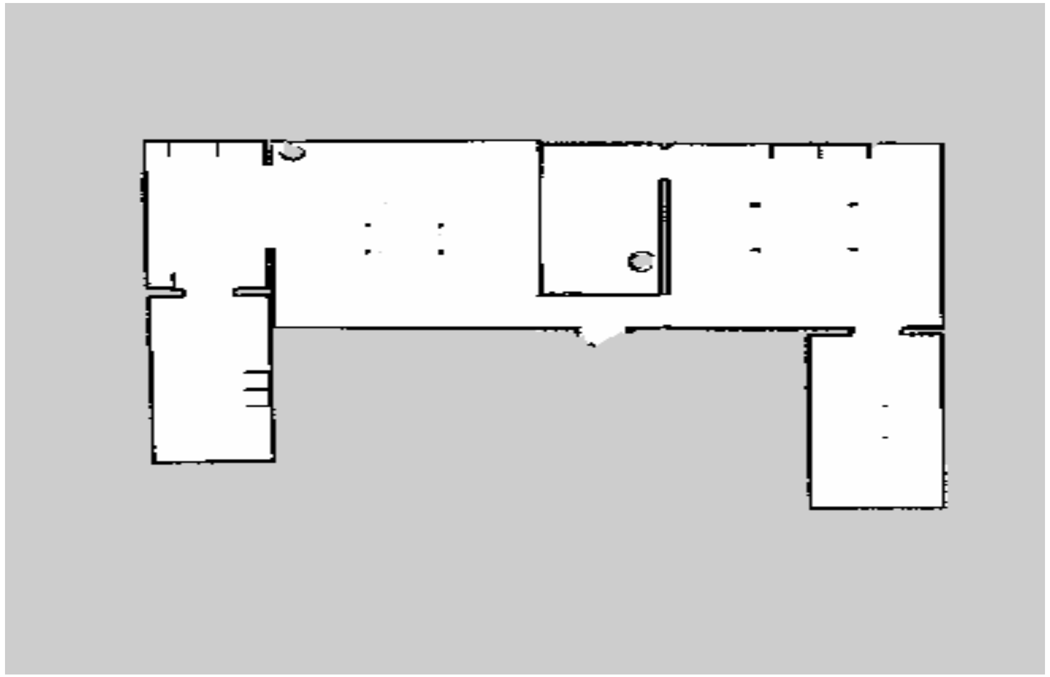
- Create a Map(.pgm and .yaml)
  - `triton_gmapping.launch`
  - `teleop_particle_filter.py`.

Map.yaml

```
image: map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

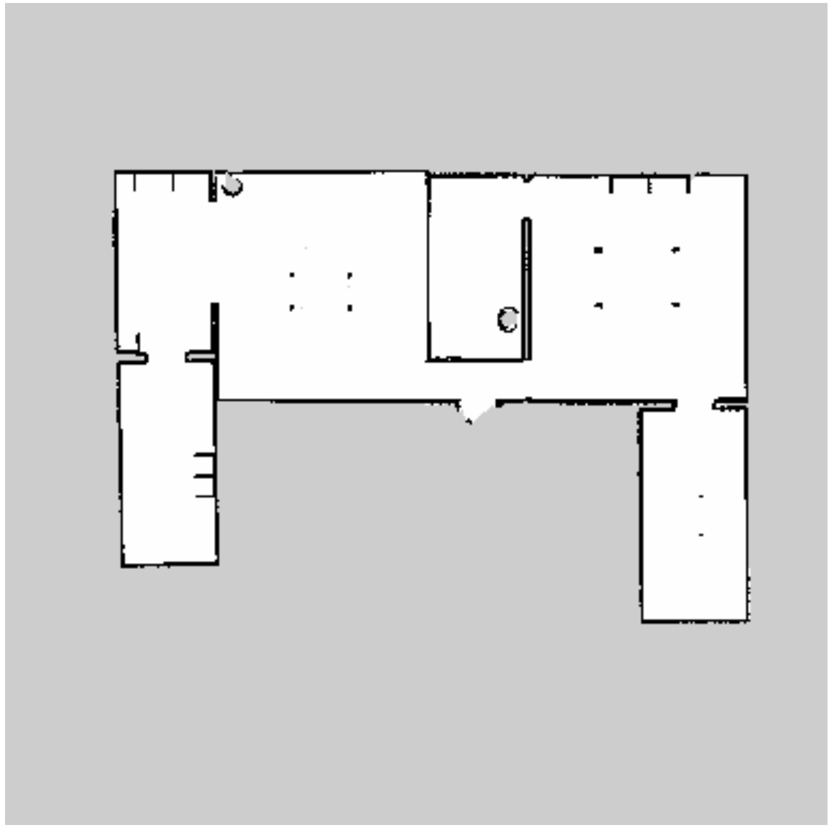


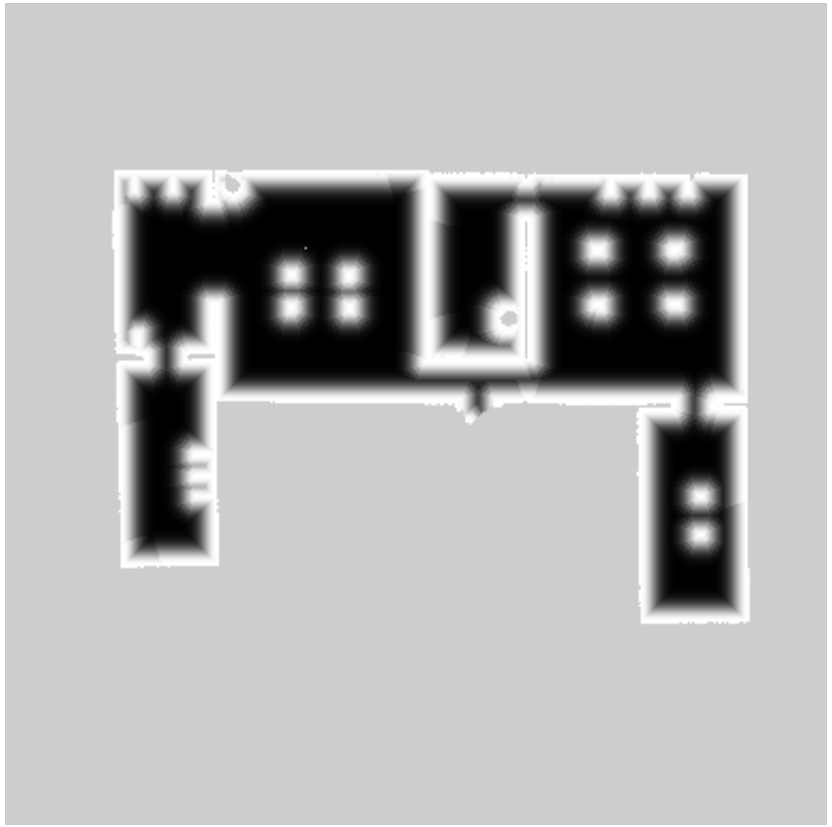
Map.pgm



Pre-computed Likelihood Field

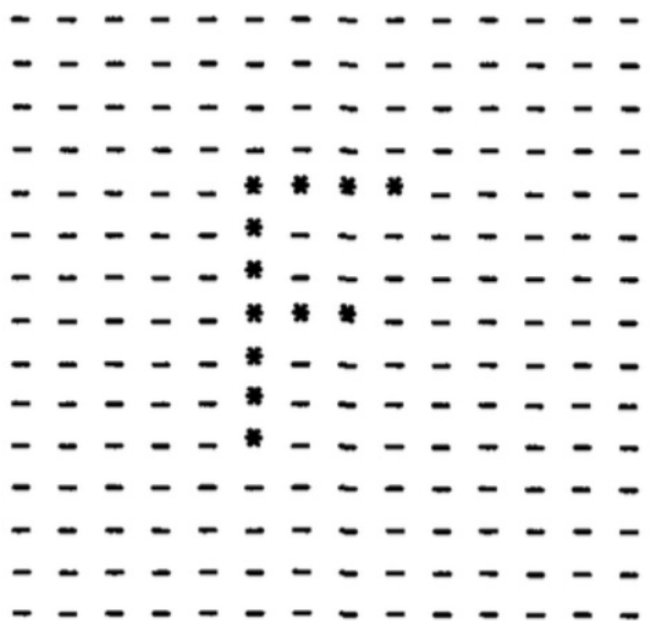
- Given the .pgm map to generate likelihood filed in generate_likelihood_field.py



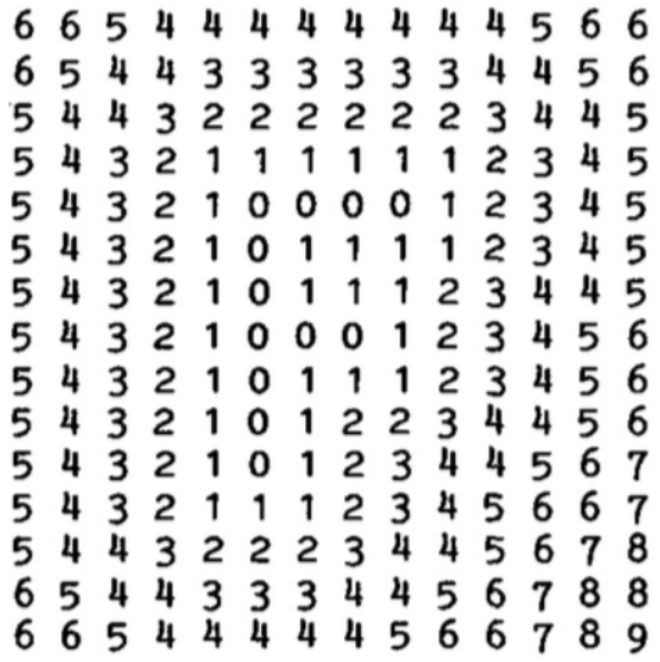


Likelihood Field Model

- The distance at each point to the nearest obstacle can be pre-computed, and stored into a distance matrix or field as a lookup table.
- The value at each point in this field indicates the distance to the nearest obstacle:
  - Brute force calculation
  - Distance transformation (from CV)
  - Brushfire algorithm (from planning)



-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	*	*	*	*	-	-	-	-	-
-	-	-	-	-	*	-	-	-	-	-	-	-	-
-	-	-	-	-	*	-	-	-	-	-	-	-	-
-	-	-	-	-	*	*	*	-	-	-	-	-	-
-	-	-	-	-	*	-	-	-	-	-	-	-	-
-	-	-	-	-	*	-	-	-	-	-	-	-	-
-	-	-	-	-	*	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-



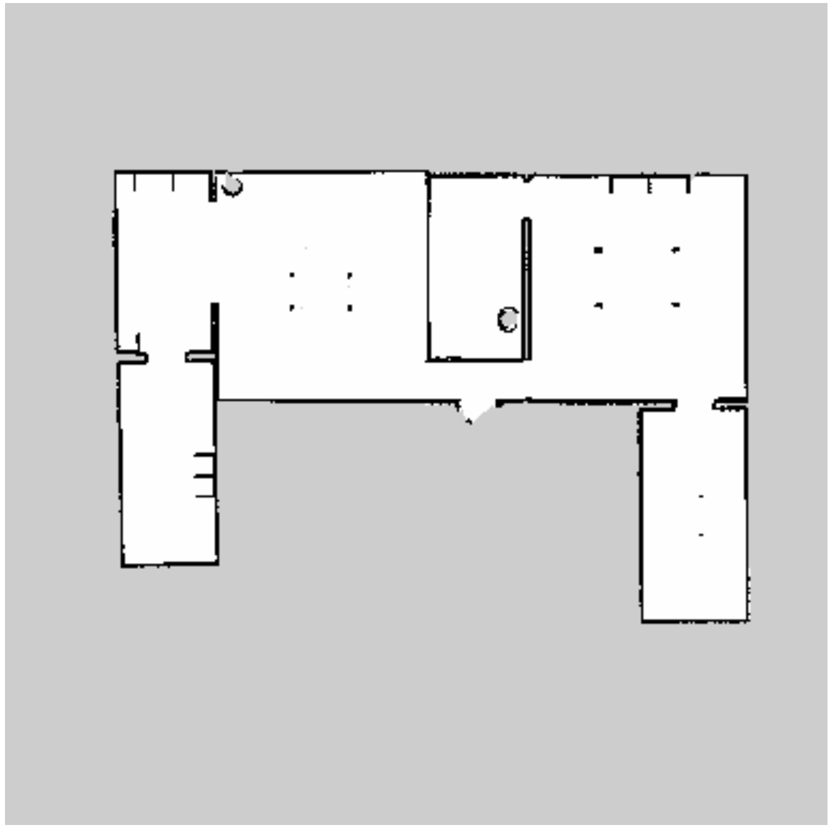
6	6	5	4	4	4	4	4	4	4	4	5	6	6
6	5	4	4	3	3	3	3	3	3	4	4	5	6
5	4	4	3	2	2	2	2	2	2	3	4	4	5
5	4	3	2	1	1	1	1	1	1	2	3	4	5
5	4	3	2	1	0	0	0	0	1	2	3	4	5
5	4	3	2	1	0	1	1	1	1	2	3	4	5
5	4	3	2	1	0	1	1	1	2	3	4	4	5
5	4	3	2	1	0	0	0	1	2	3	4	5	6
5	4	3	2	1	0	1	1	1	2	3	4	5	6
5	4	3	2	1	0	1	2	2	3	4	4	5	6
5	4	3	2	1	0	1	2	3	4	4	5	6	7
5	4	3	2	1	1	1	2	3	4	5	6	6	7
5	4	4	3	2	2	2	3	4	4	5	6	7	8
6	5	4	4	3	3	3	4	4	5	6	7	8	8
6	6	5	4	4	4	4	4	5	6	6	7	8	9

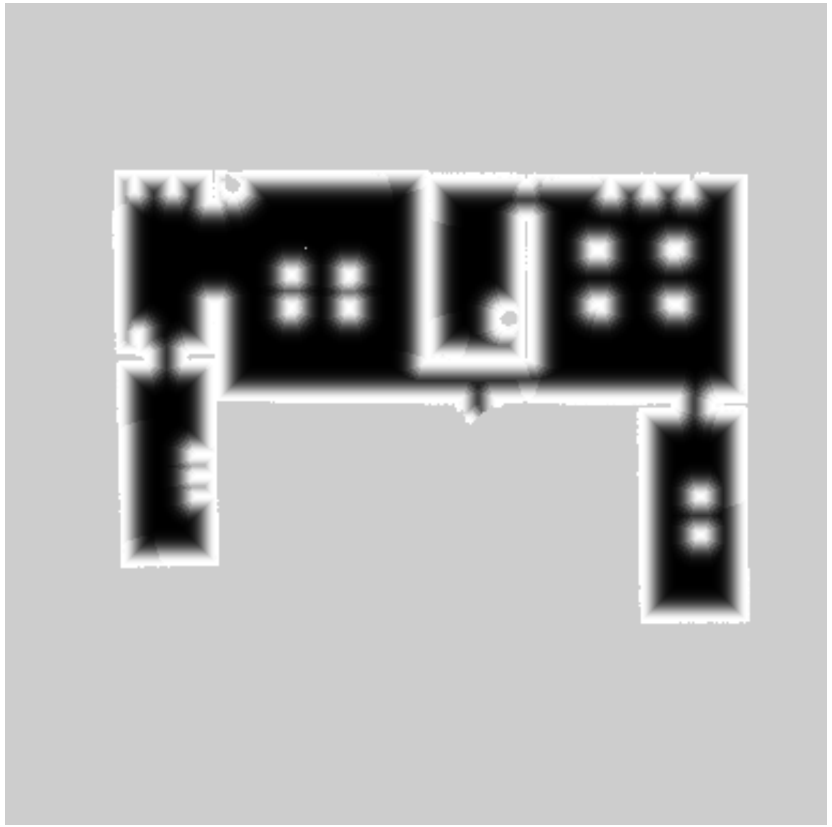
Process Images

- Option 1: OpenCV
  - Install pip install opencv-python
  - import cv2
  - cv2.imread('map/map.pgm’, cv2.IMREAD_GRAYSCALE)
  - cv2.distanceTransform (see details: https://www.geeksforgeeks.org/python-opencv-distancetransform-function/)
- Option 2: Pillow
  - Install pip install pillow
  - Import PIL.Image
  - metric_map=np.array(PIL.Image.open('map/map.pgm’))
  - metric_map==0 # there is an obstacle
  - Brute force to compute the likelihood map

Pre-computed Likelihood Field

- Given the .pgm map to generate likelihood filed in generate_likelihood_field.py





Sample_motion_model

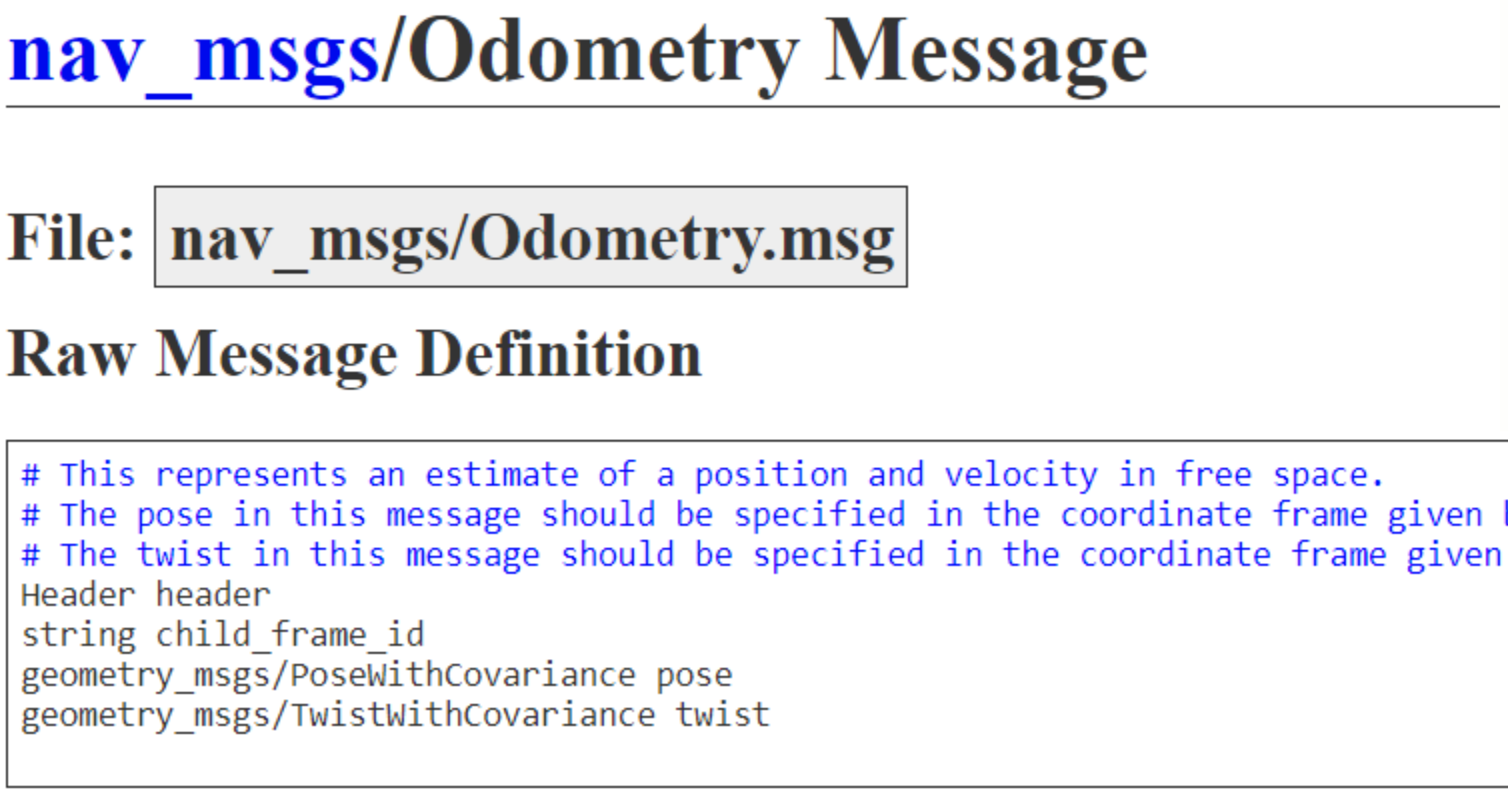
- Implement motion model in particle_filter.py
- u is subscribed from /odom
- X is your belief, you cannot have a real x in deliverable 1

```
1.  Algorithm **sample_motion_model**(u, x):
    $\boxed{u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle}, x = \langle x, y, \theta \rangle$
1.  $\hat{\delta}_{rot1} = \delta_{rot1} + \text{sample}(\alpha_1 |\delta_{rot1}| + \alpha_2\ \delta_{trans})$
2.  $\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3\ \delta_{trans} + \alpha_4\ (|\delta_{rot1}| + |\delta_{rot2}|))$
3.  $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 |\delta_{rot2}| + \alpha_2\ \delta_{trans})$

4.  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5.  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6.  $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

7.  Return $\langle x', y', \theta' \rangle$
```

sample_normal_distribution

Sample_motion_model

- Robot coordinate frame
- Odometry from proprioceptive sensors

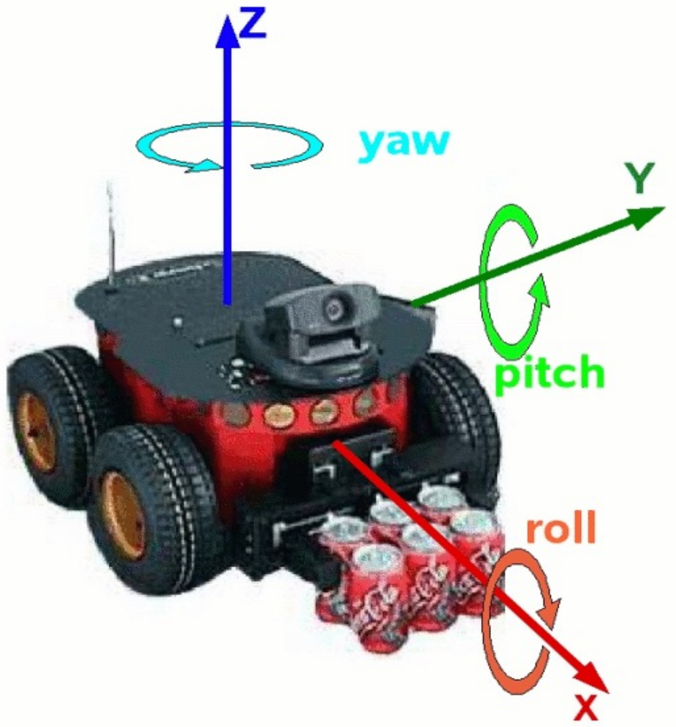


nav_msgs/Odometry Message

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```



Sample_motion_model

$$\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$$

$$\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$$

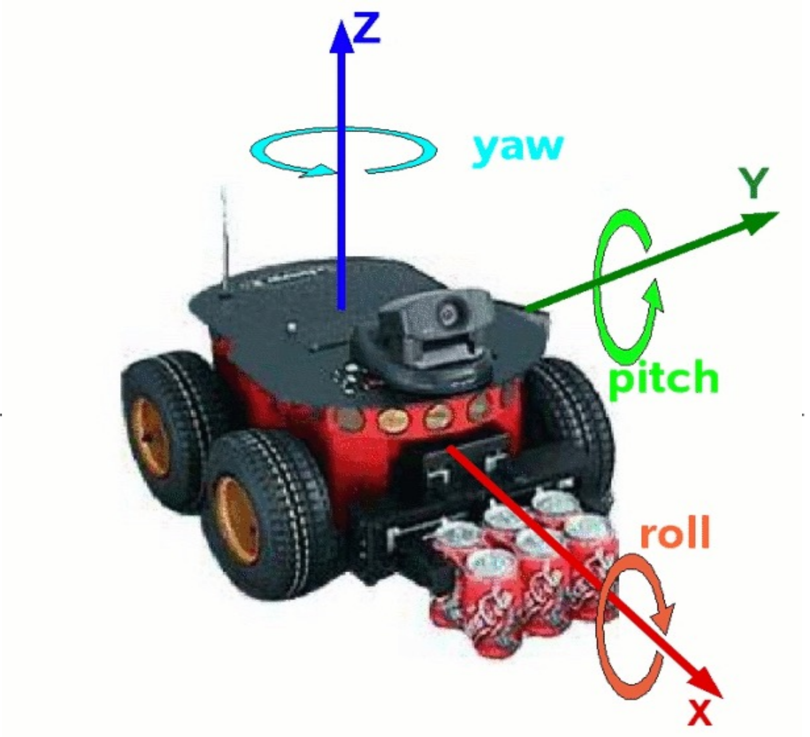
$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$

```
1.  Algorithm sample_motion_model(u, x):
    $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$
1.  $\hat{\delta}_{rot1} = \boxed{\delta_{rot1}} + \text{sample}(\alpha_1 |\delta_{rot1}| + \alpha_2 \delta_{trans})$
2.  $\hat{\delta}_{trans} = \boxed{\delta_{trans}} + \text{sample}(\alpha_3 \delta_{trans} + \alpha_4 (|\delta_{rot1}| + |\delta_{rot2}|))$
3.  $\hat{\delta}_{rot2} = \boxed{\delta_{rot2}} + \text{sample}(\alpha_1 |\delta_{rot2}| + \alpha_2 \delta_{trans})$

4.  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5.  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6.  $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

7.  Return $\langle x', y', \theta' \rangle$
```

sample_normal_distribution

Sample_motion_model

- Robot coordinate frame
- Odometry from proprioceptive sensors



nav_msgs/Odometry Message

File: `nav_msgs/Odometry.msg`

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

theta =transformations.euler_from_quaternion(quaternion_arr)

Sample_motion_model

Option 1:

np.random.normal()

Option 2:

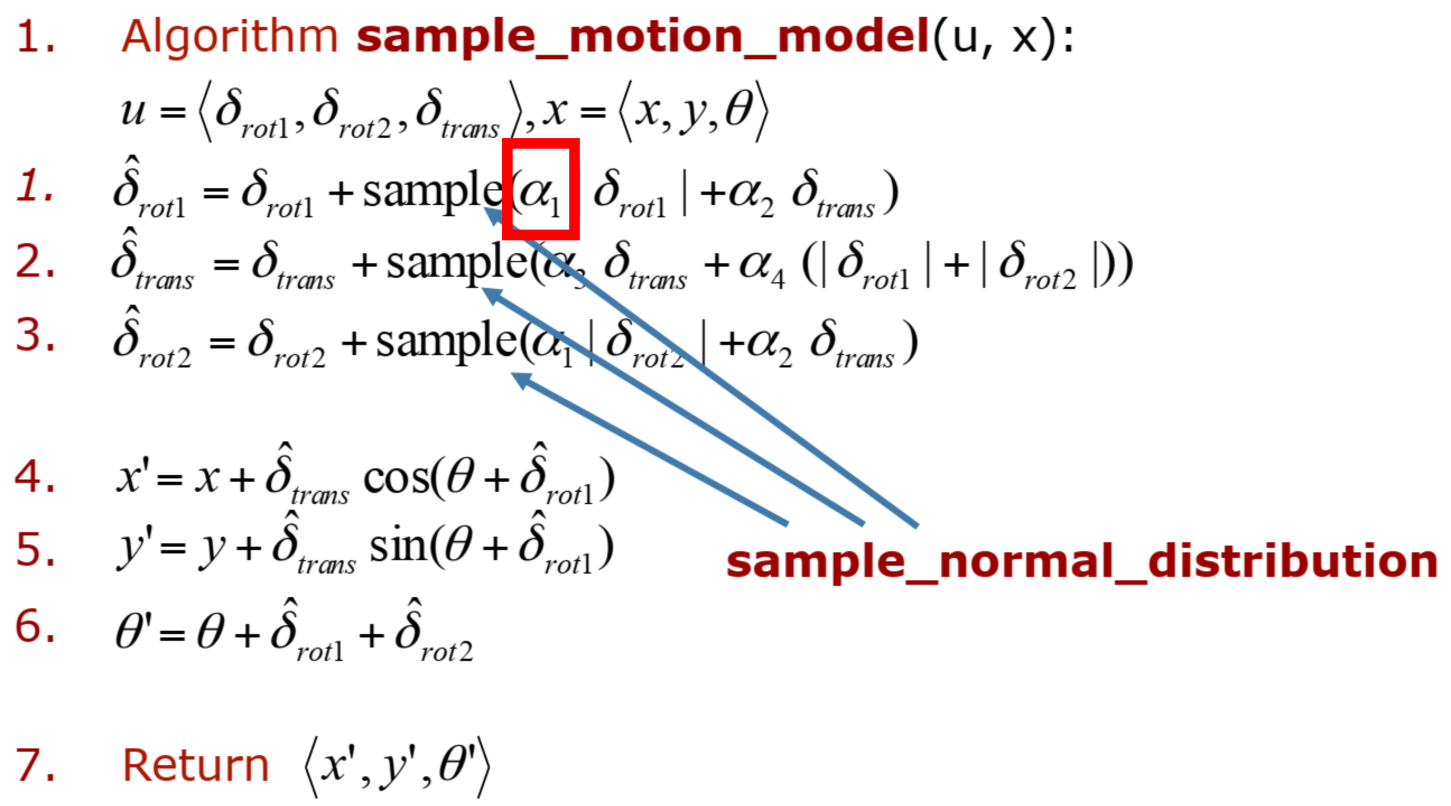
```
1.  Algorithm **sample_normal_distribution**($b$):
2.  return  $\frac{1}{2}\sum_{i=1}^{12} rand(-b, b)$
```

```
1.  Algorithm **sample_motion_model**(u, x):
    $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$
1.  $\hat{\delta}_{rot1} = \delta_{rot1} + \boxed{\text{sample}}(\alpha_1 |\delta_{rot1}| + \alpha_2\, \delta_{trans})$
2.  $\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3\, \delta_{trans} + \alpha_4\, (|\delta_{rot1}| + |\delta_{rot2}|))$
3.  $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 |\delta_{rot2}| + \alpha_2\, \delta_{trans})$

4.  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5.  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6.  $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

7.  Return $\langle x', y', \theta' \rangle$
```

sample_normal_distribution

Sample_motion_model

- α_i is manually defined parameter
- In the simulation, it will not influence too much, you may find your own best value.



```
1.  Algorithm **sample_motion_model**(u, x):
    $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$
1.  $\hat{\delta}_{rot1} = \delta_{rot1} + \text{sample}(\boxed{\alpha_1} | \delta_{rot1} | + \alpha_2 \delta_{trans})$
2.  $\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3 \delta_{trans} + \alpha_4 (| \delta_{rot1} | + | \delta_{rot2} |))$
3.  $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 | \delta_{rot2} | + \alpha_2 \delta_{trans})$

4.  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5.  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6.  $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

7.  Return $\langle x', y', \theta' \rangle$
```

Sensor model

- Implement sensor model in particle_filter.py

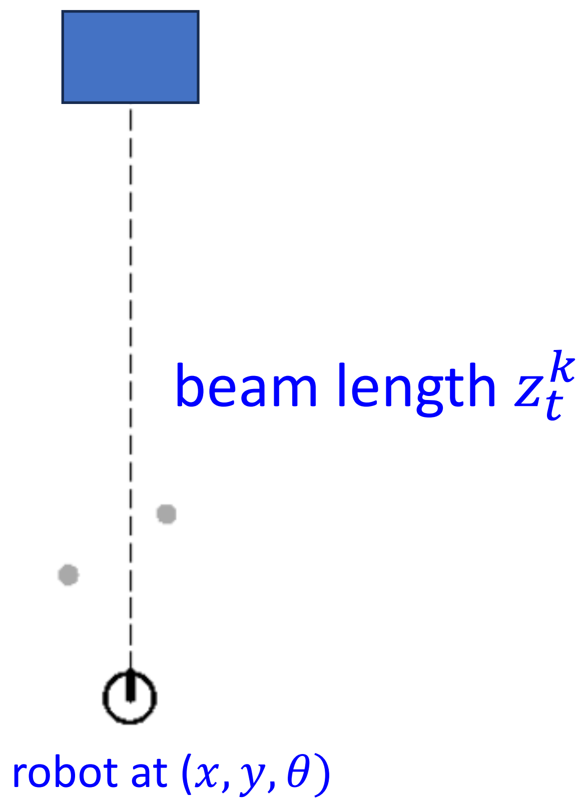
```
1:  **Algorithm likelihood_field_range_finder_model($z_t, x_t, m$):**
2:      $q = 1$
3:      for all $k$ do                          Multiply the individual values of $p(z_t^k \mid x_t, m)$
4:          if $z_t^k \neq z_{\max}$            If the sensor reading is a max range reading, ignore it
5:              $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$
6:              $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$
                (Compute end point location)
7:              $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$
                (Check lookup table (dist matrix))
8:              $q = q \cdot \left( z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$
                (Compute probability density)
9:      return $q$
```

Why sensor model estimate the probability of seeing the expected measurement?

- True pose

$$x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$$
$$y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$$

Compute end point location given the pose of robot x, y, θ and its beam length z_t^k

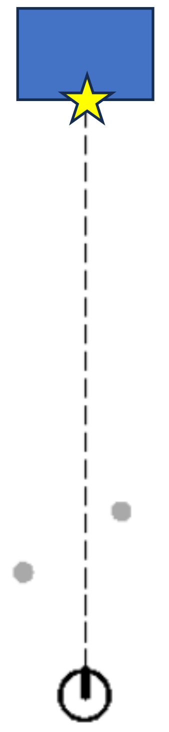


Why sensor model estimate the probability of seeing the expected measurement?

- True pose

$$x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$$
$$y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$$
$$dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$$
$$q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$$

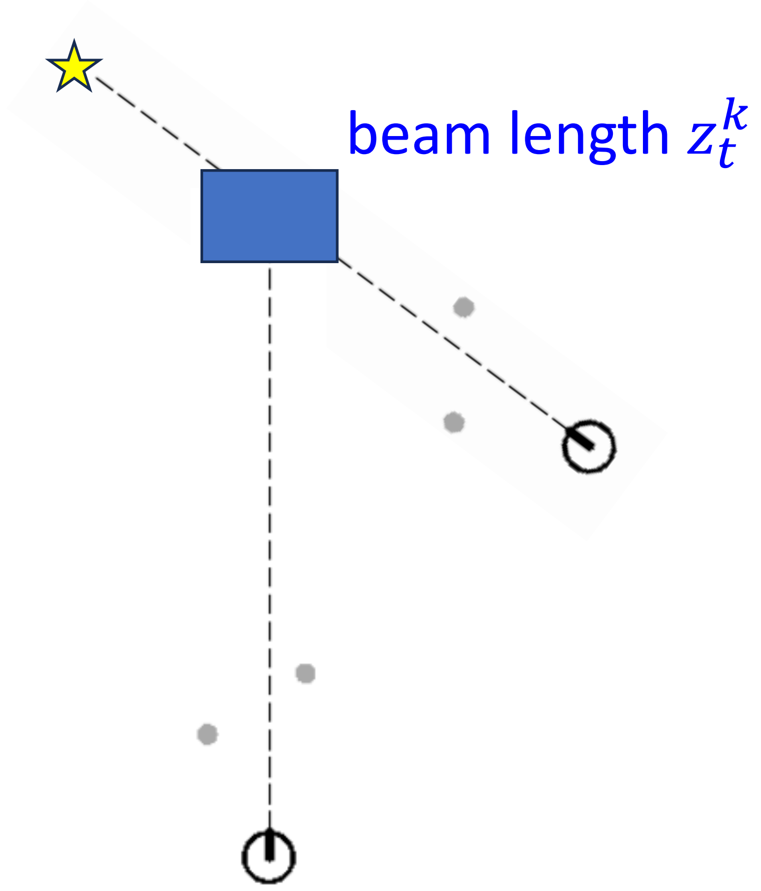
$$\boxed{\text{prob}(dist^2, \sigma_{\text{hit}}^2) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} \exp\left(-\frac{dist^2}{2\sigma_{\text{hit}}^2} \right)}$$



Why sensor model estimate the probability of seeing the expected measurement?

- Wrong pose

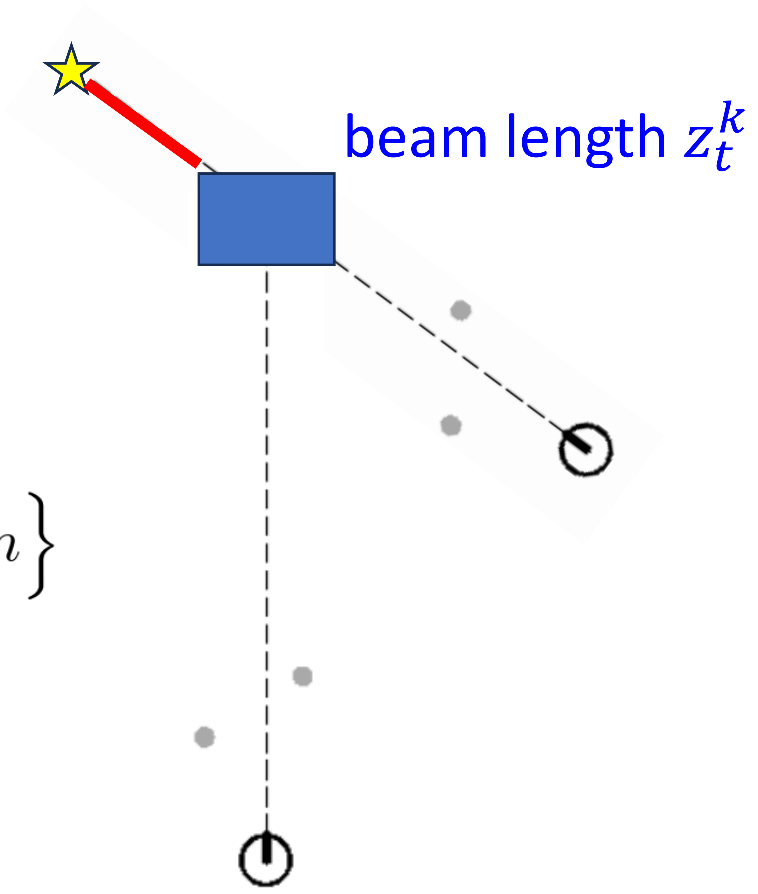
$$x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$$
$$y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$$



Why sensor model estimate the probability of seeing the expected measurement?

- Wrong pose

$$x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$$
$$y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$$
$$dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$$
$$q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$$

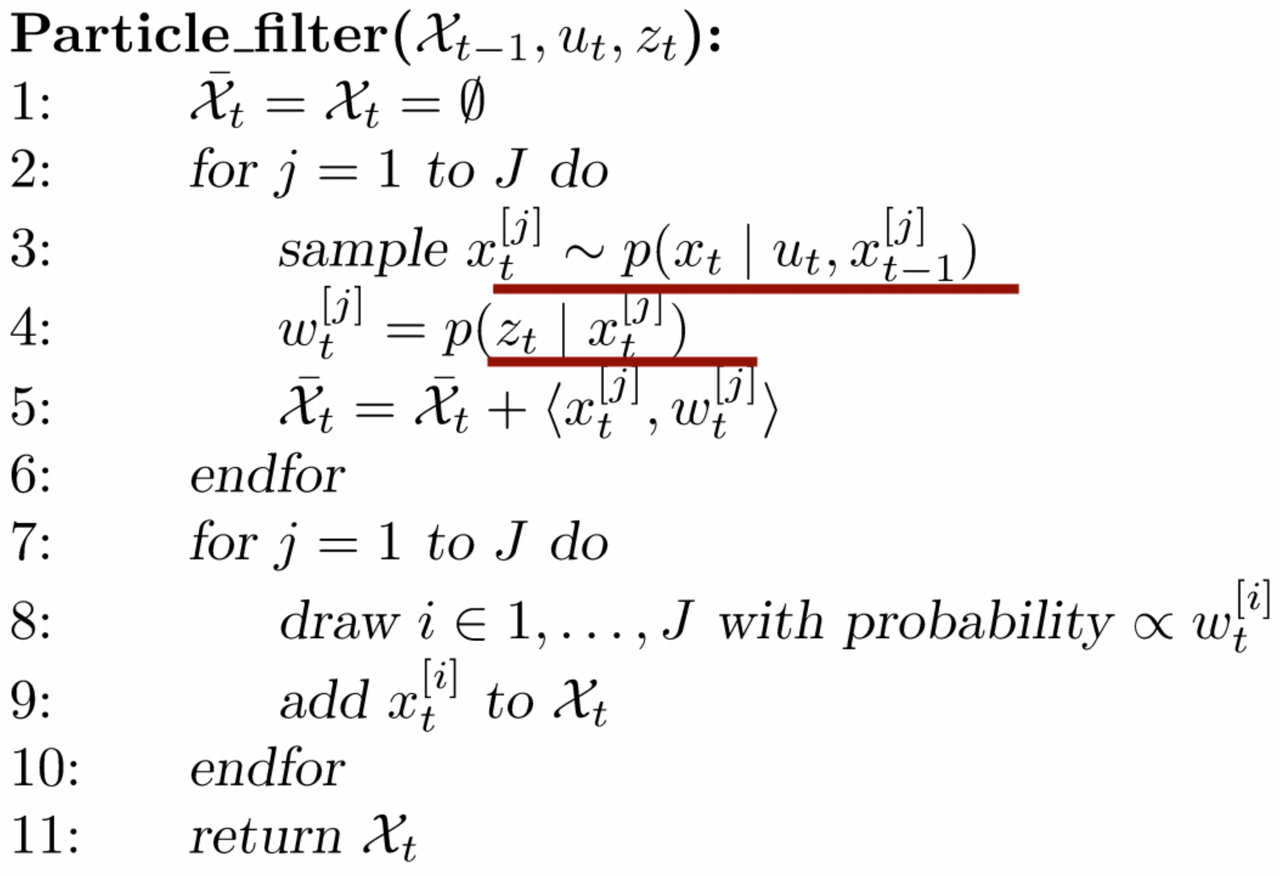


Tips

- You can not verify if your code can work correctly.
- The likelihood field map, motion and sensor models will be used for particle filter for robot localization.
- Manually-defined parameters will not influence a lot in the simulation environment, but just used for testing your understanding of the mathematical formulation.

Deliverable 2

- Implement particle filter in particle_filter.py
- 1000 particles are sufficient



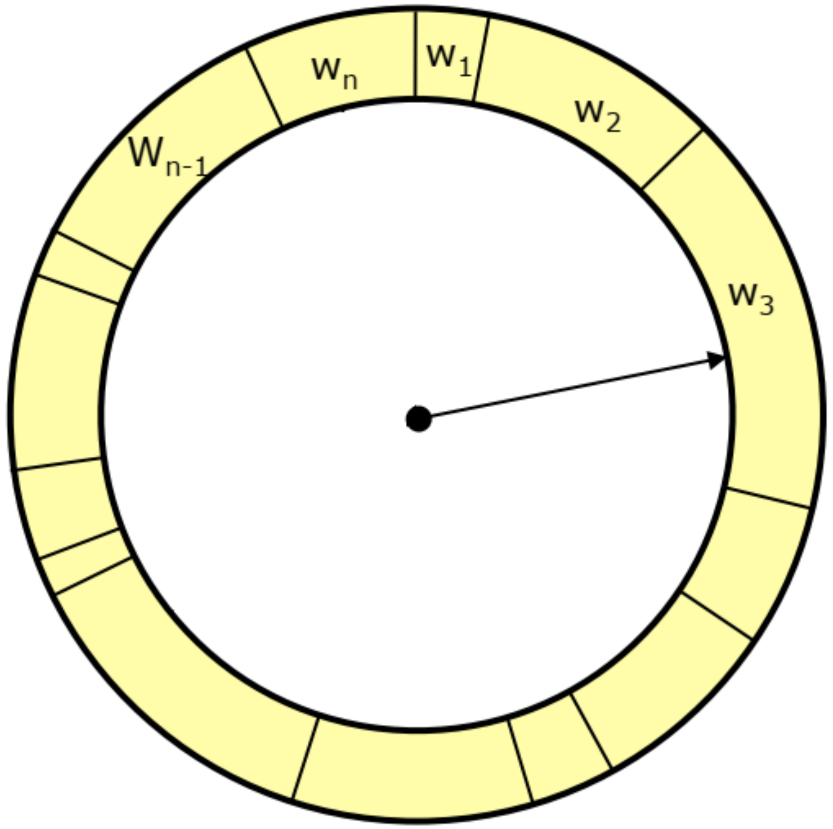
```
**Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
1:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2:      for $j = 1$ to $J$ do
3:          sample $x_t^{[j]} \sim \underline{p(x_t \mid u_t, x_{t-1}^{[j]})}$
4:          $w_t^{[j]} = p(\underline{z_t \mid x_t^{[j]})}$
5:          $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6:      endfor
7:      for $j = 1$ to $J$ do
8:          draw $i \in 1, \ldots, J$ with probability $\propto w_t^{[i]}$
9:          add $x_t^{[i]}$ to $\mathcal{X}_t$
10:     endfor
11:     return $\mathcal{X}_t$
```

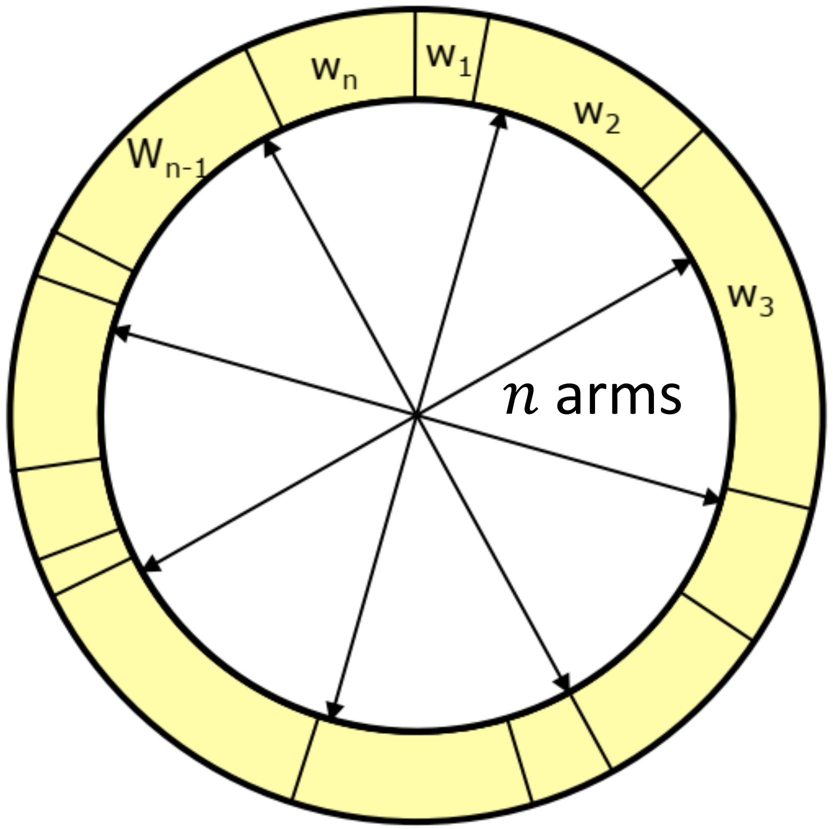
Resampling

- Draw sample i with probability $w_t^{[i]}$, repeat n times.
- Informally: “Replace unlikely samples by more likely ones”
- “Trick” to avoid that many samples cover unlikely states
- Needed as we have a limited number of samples

Resampling

np.random.choice





- Roulette wheel with n bins
- Brute force $O(n*n)$, or binary search $O(nlogn)$

- Stochastic universal sampling
- Also called low variance sampling
- Complexity: $O(n)$

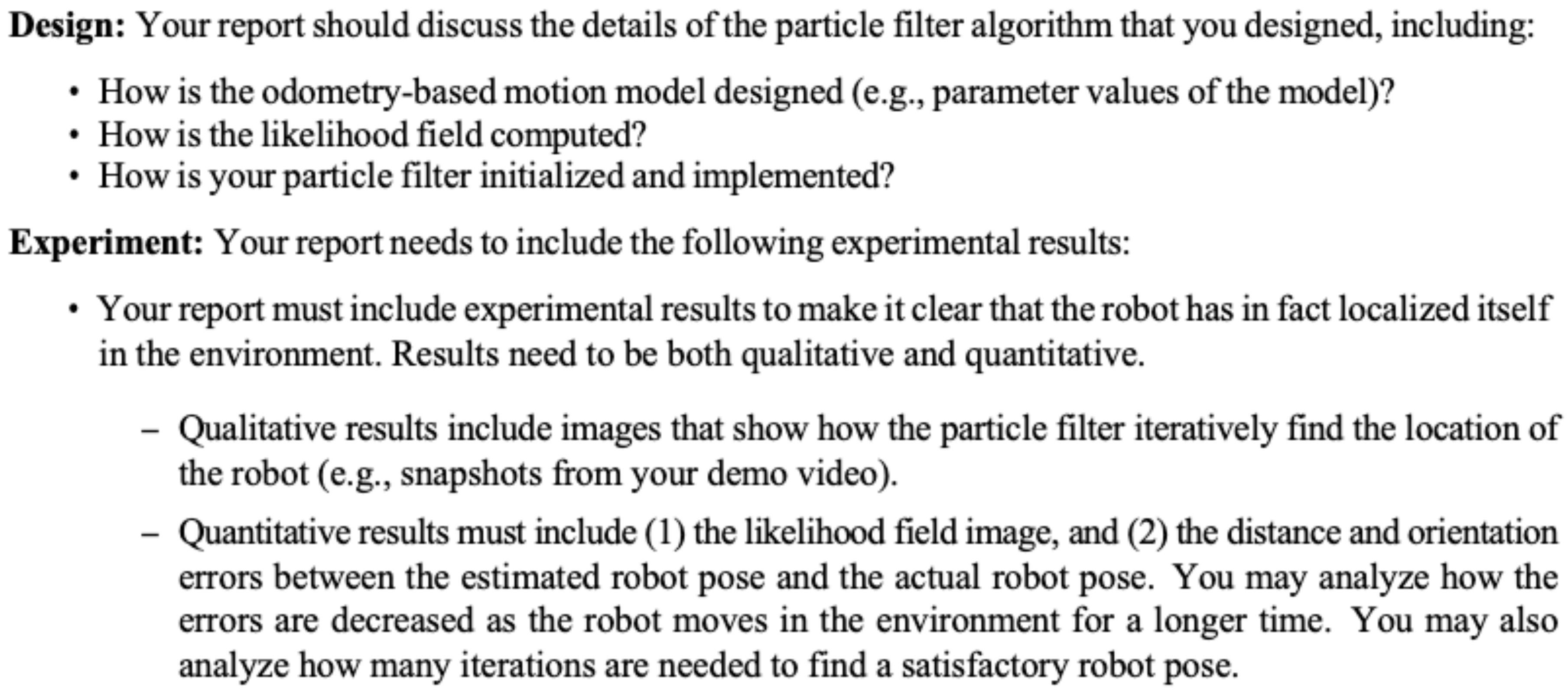
Tips

- At least 2 subscribers: \scan and \odom
- At least 1 publisher: \PoseArray to visualize the particles in the map
    - pose_array = PoseArray()
    - pose_array.poses = particle # quaternion_from_euler
    - pose_array.header = Header()
    - pose_array.header.stamp = rospy.Time.now()
    - pose_array.header.frame_id = 'map'
    - particles_publisher.publish(pose_array)

- Do not control you robot to move forward and rotate at the same time.

Deliverable 3

- A single 4-page report using Latex



Design: Your report should discuss the details of the particle filter algorithm that you designed, including:

- How is the odometry-based motion model designed (e.g., parameter values of the model)?
- How is the likelihood field computed?
- How is your particle filter initialized and implemented?

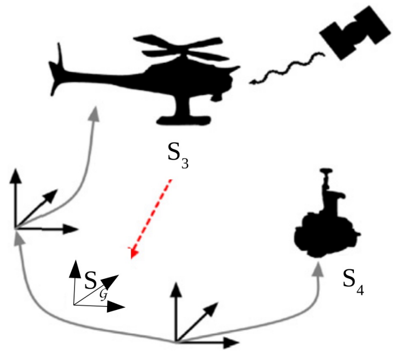
Experiment: Your report needs to include the following experimental results:

- Your report must include experimental results to make it clear that the robot has in fact localized itself in the environment. Results need to be both qualitative and quantitative.
  - Qualitative results include images that show how the particle filter iteratively find the location of the robot (e.g., snapshots from your demo video).
  - Quantitative results must include (1) the likelihood field image, and (2) the distance and orientation errors between the estimated robot pose and the actual robot pose. You may analyze how the errors are decreased as the robot moves in the environment for a longer time. You may also analyze how many iterations are needed to find a satisfactory robot pose.

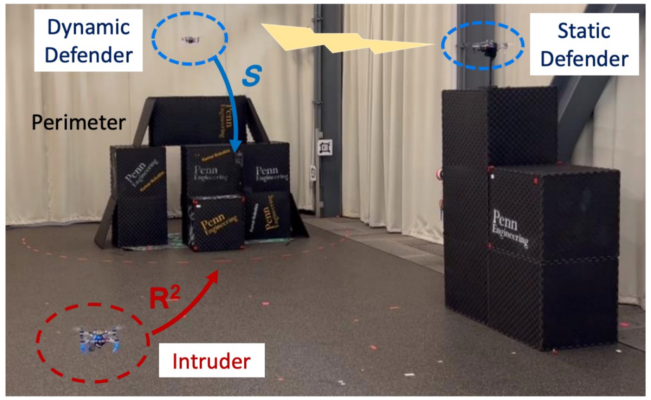
Global Localization

Relative Pose Estimation in CSLAM

- Collaborative SLAM requires to estimate the relative pose of heterogeneous robots with similar sensing modalities or even different sensors.

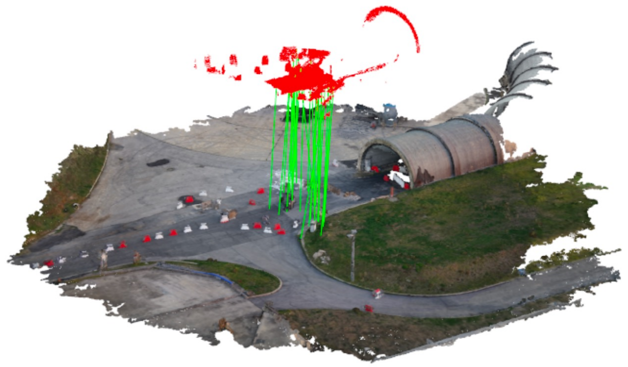


GPS or same starting configuration



Robot-robot detection

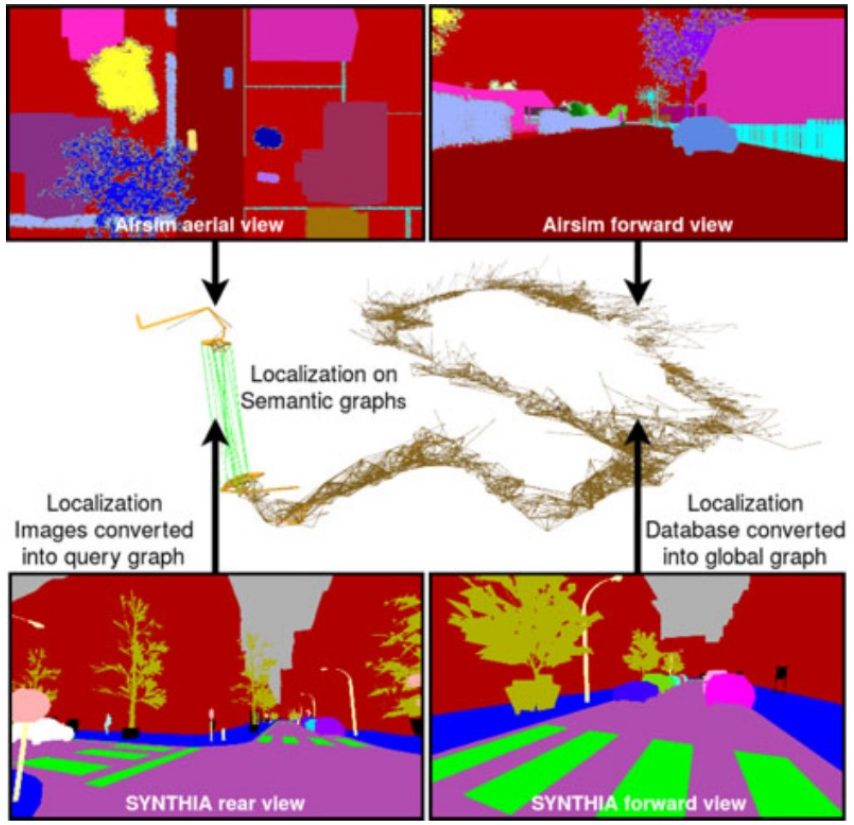
(ArXiv 2022)



Satellite Imagery (c) Google, GeoBasis-DE/BKG

Static global map registration

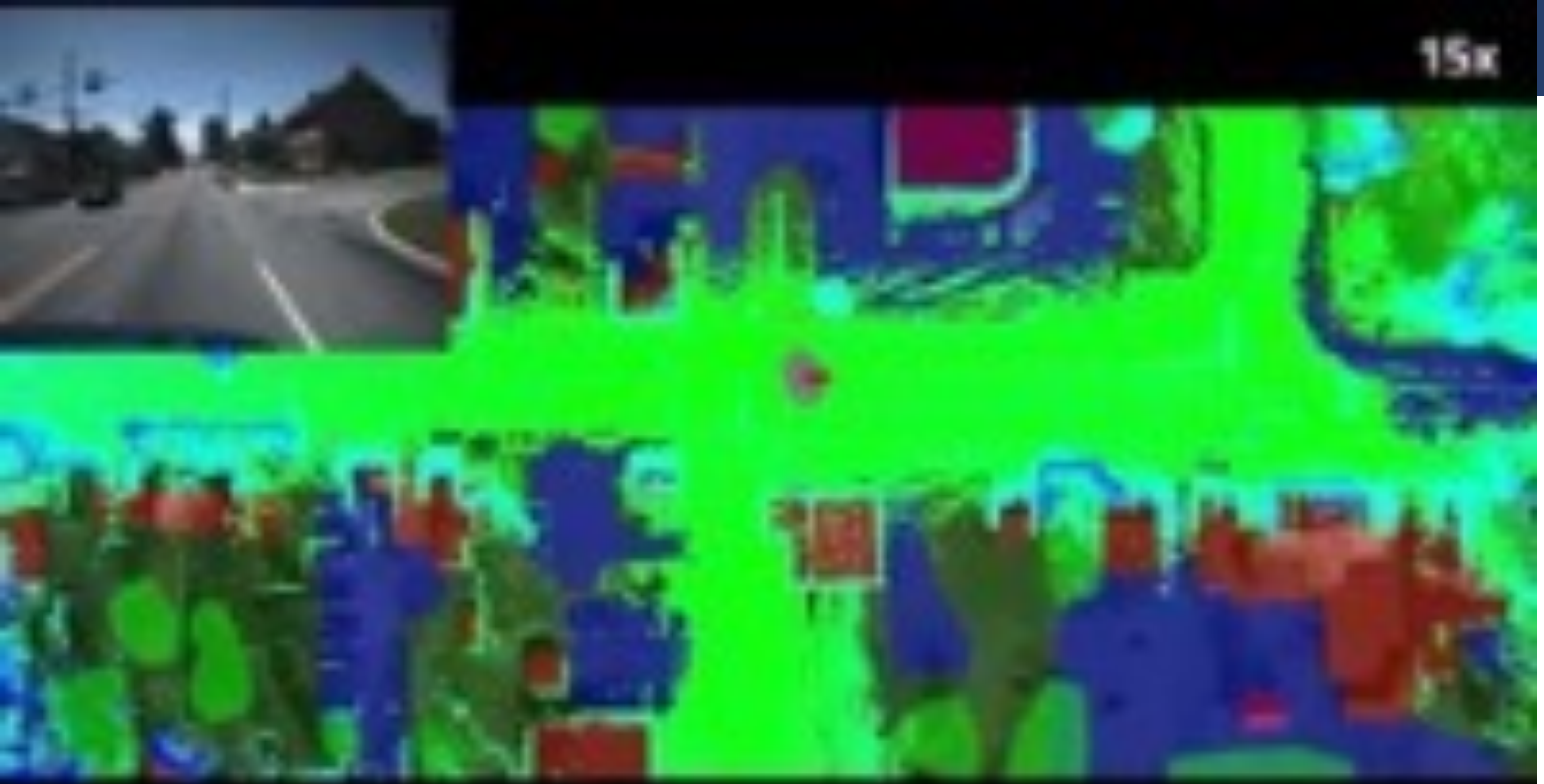
(SSRR 2017, RAL 2021)



Airsim aerial view
Airsim forward view
Localization on Semantic graphs
Localization Images converted into query graph
Localization Database converted into global graph
SYNTHIA rear view
SYNTHIA forward view

IR-LCD with references

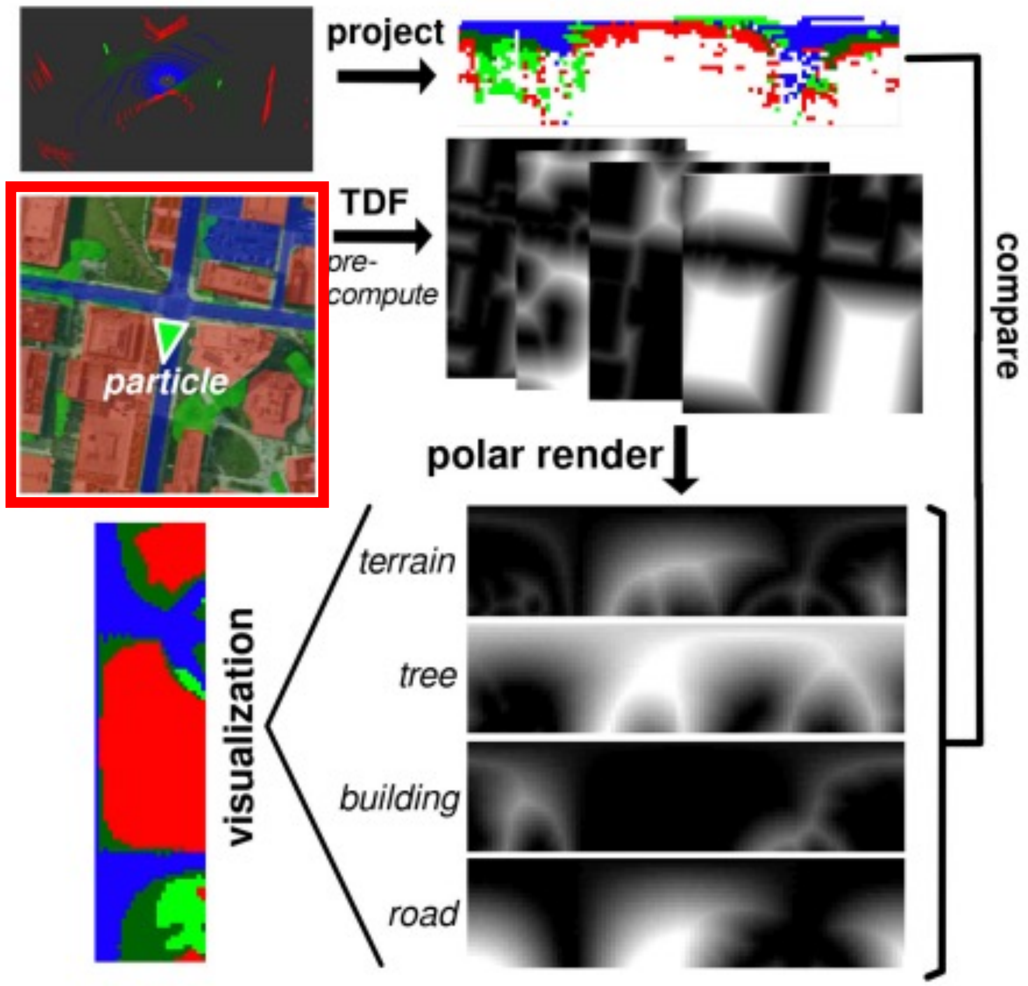
(RAL 2018)



Morgantown, PA

Sensor model

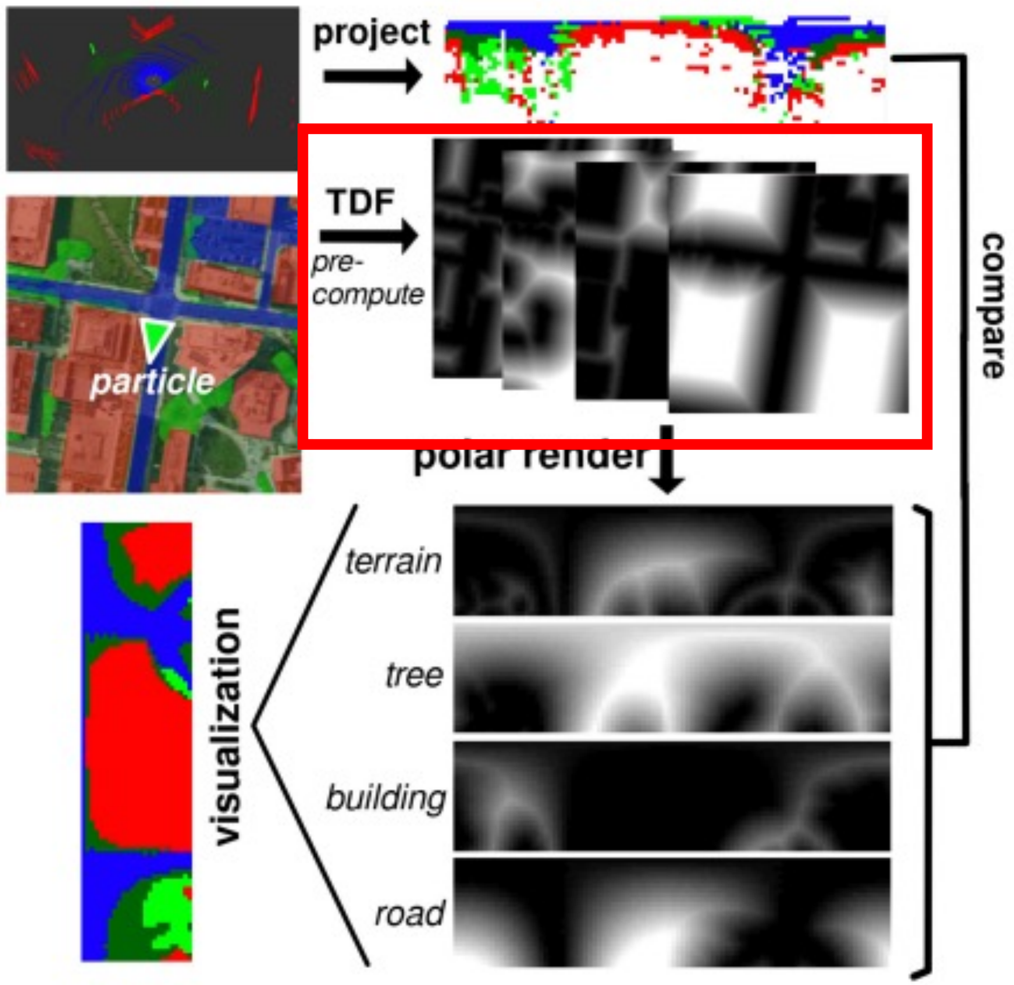
- The global map is first processed by performing semantic segmentation



Sensor model

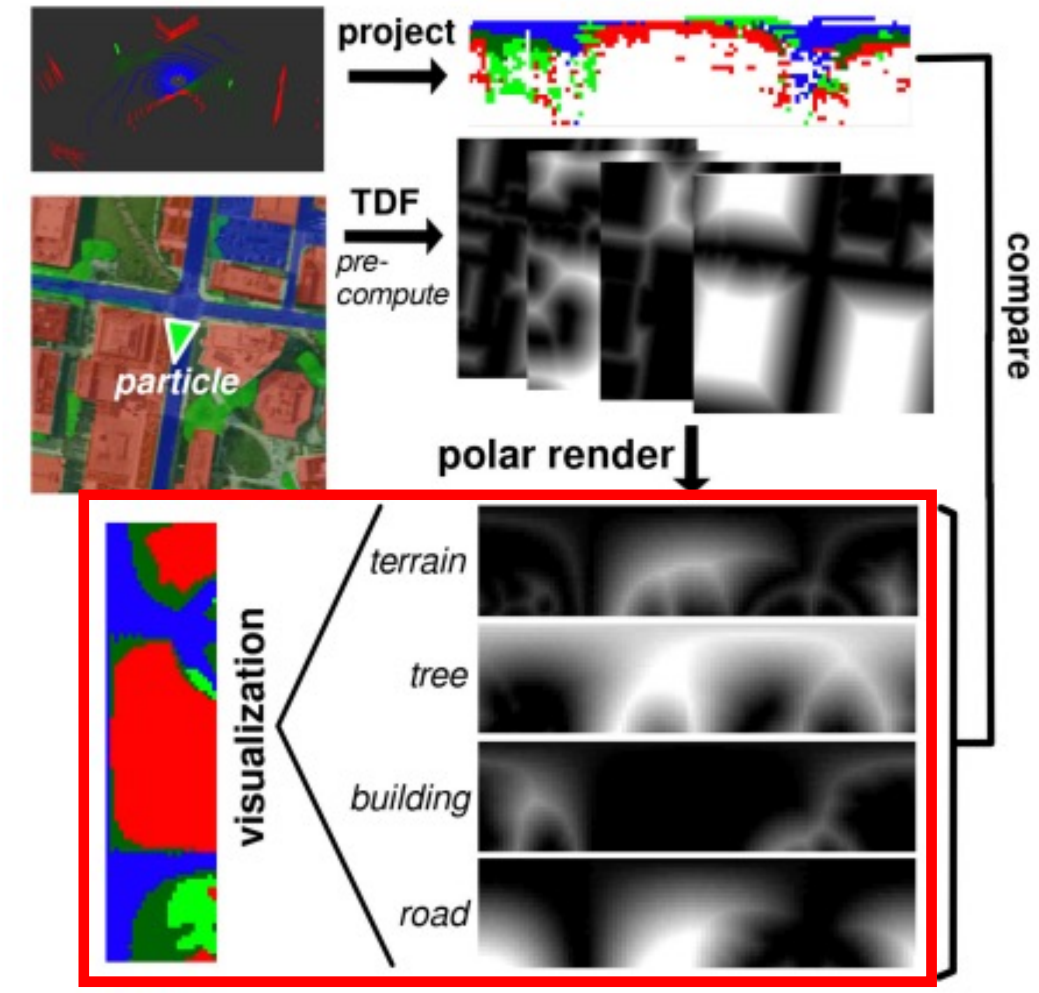
- The global map is first processed by performing semantic segmentation.

- For each type of semantics, they compute the likelihood distance map, here is also called truncated distance field (TDF)



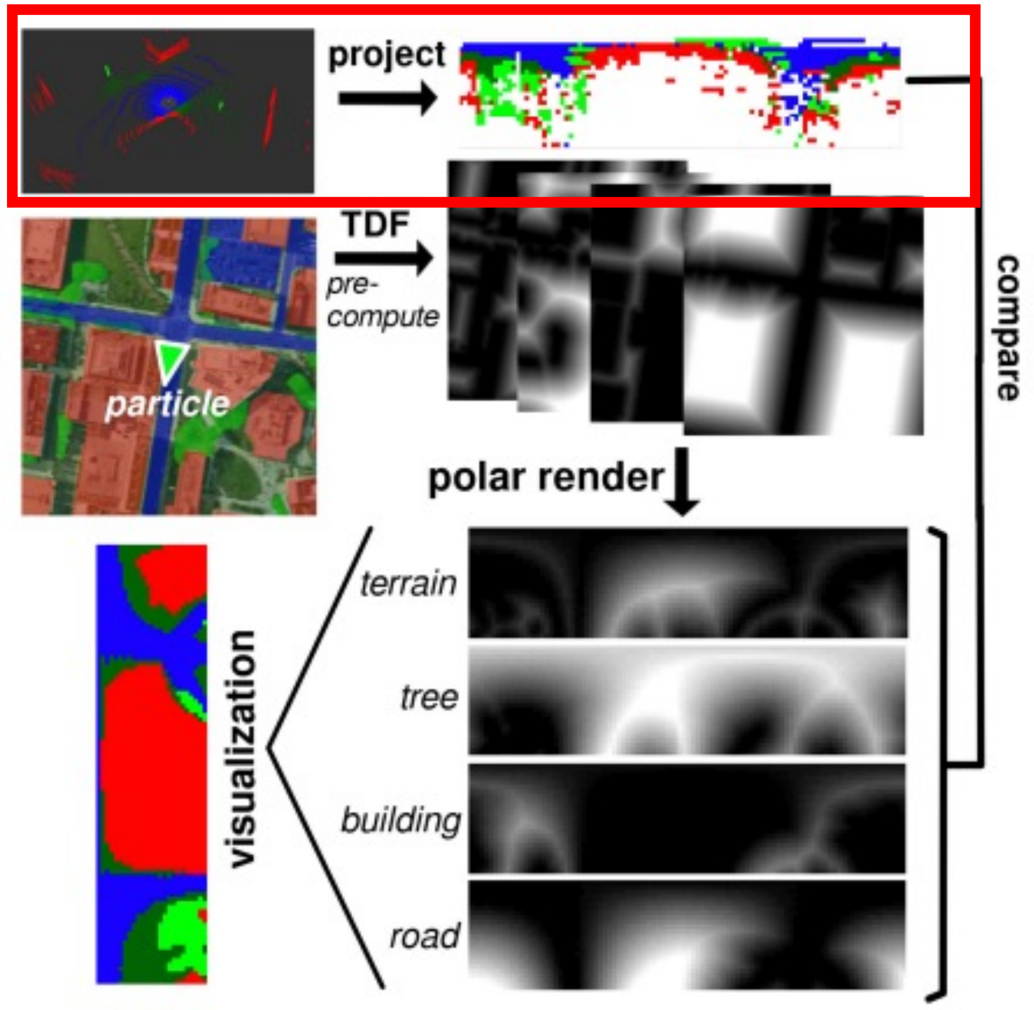
Sensor model

- The global map is first processed by performing semantic segmentation.
- For each type of semantics, they compute the likelihood distance map, here is also called truncated distance field (TDF)
- Generate the TDF features for each type of semantics.

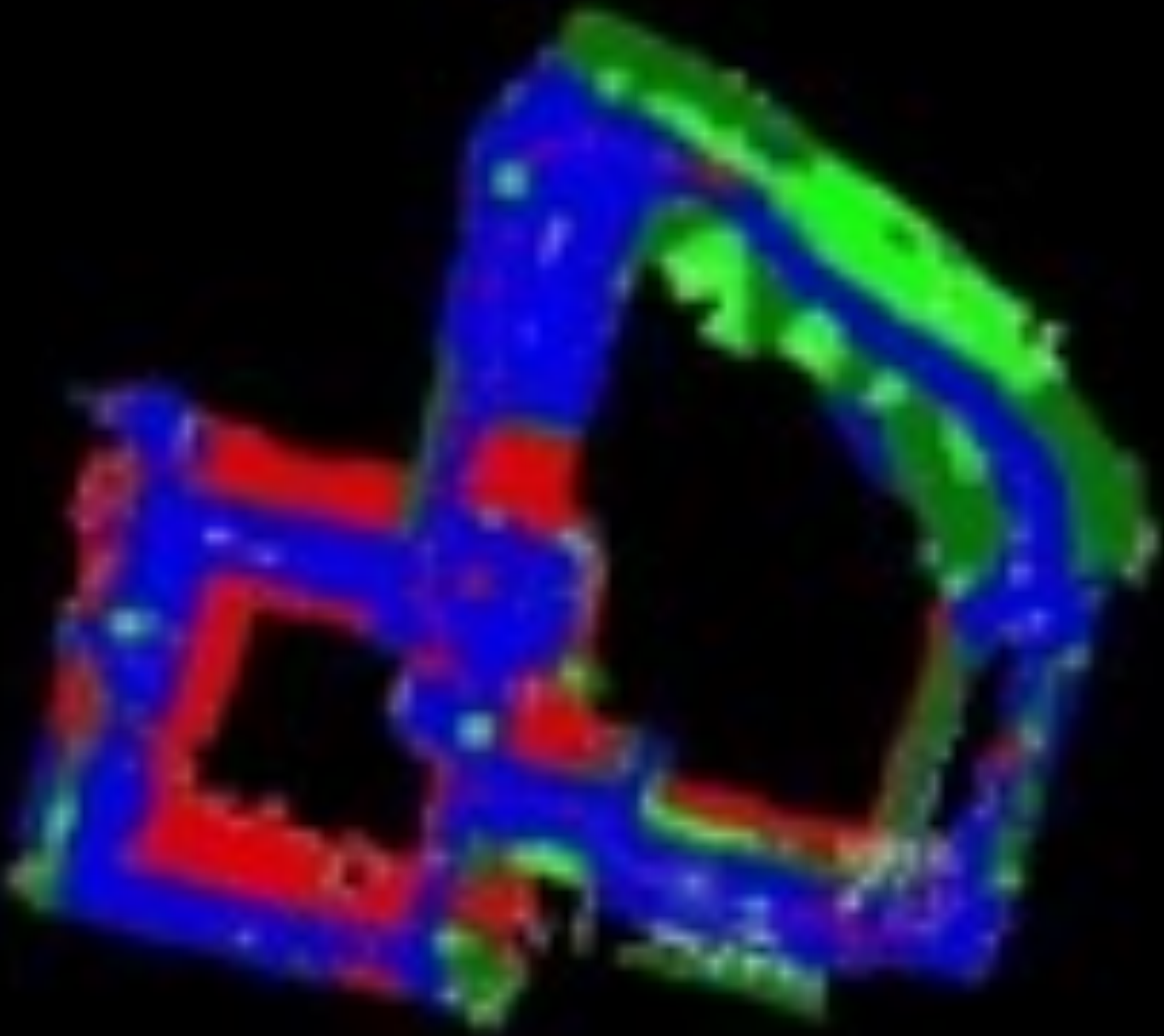


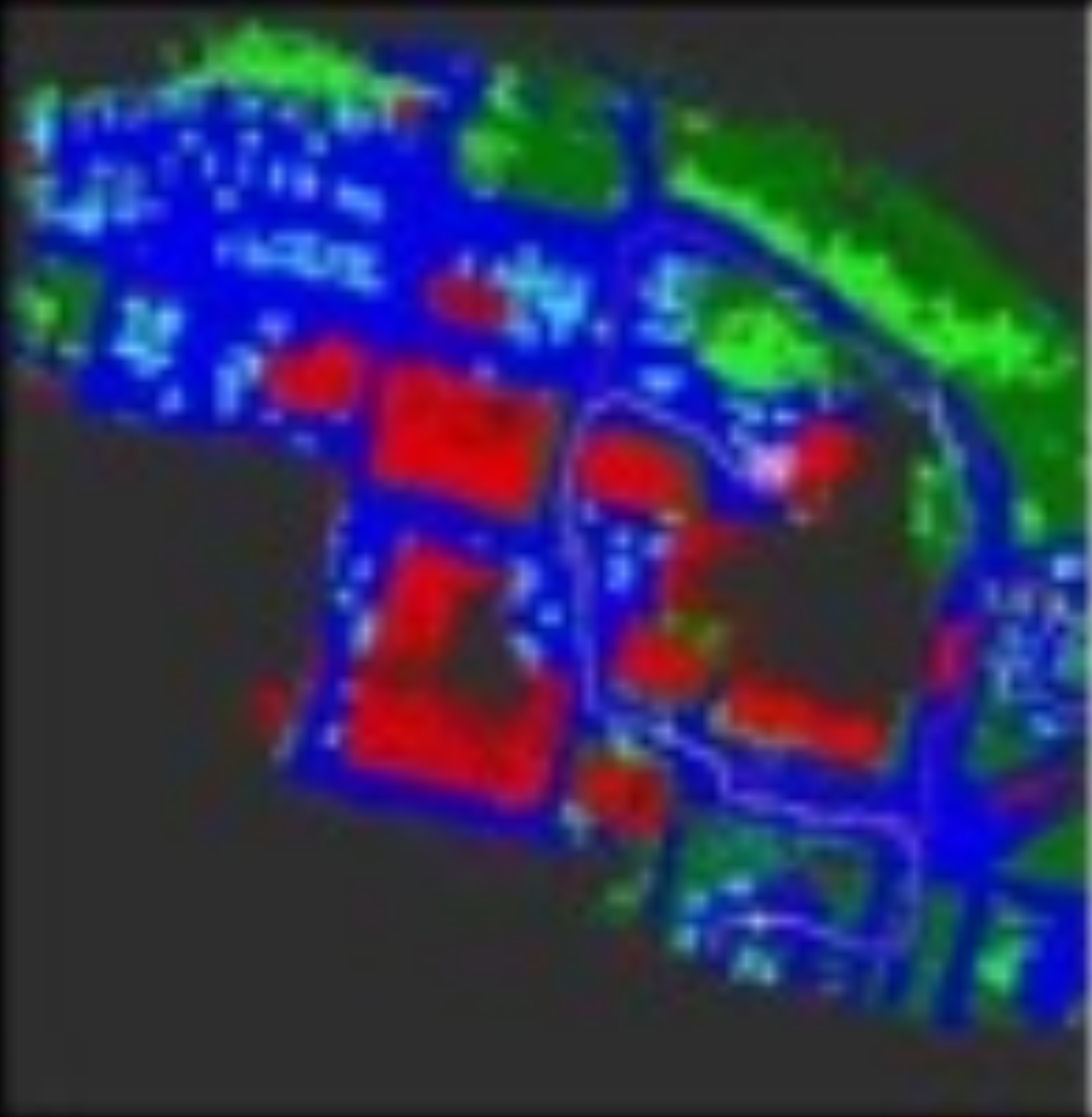
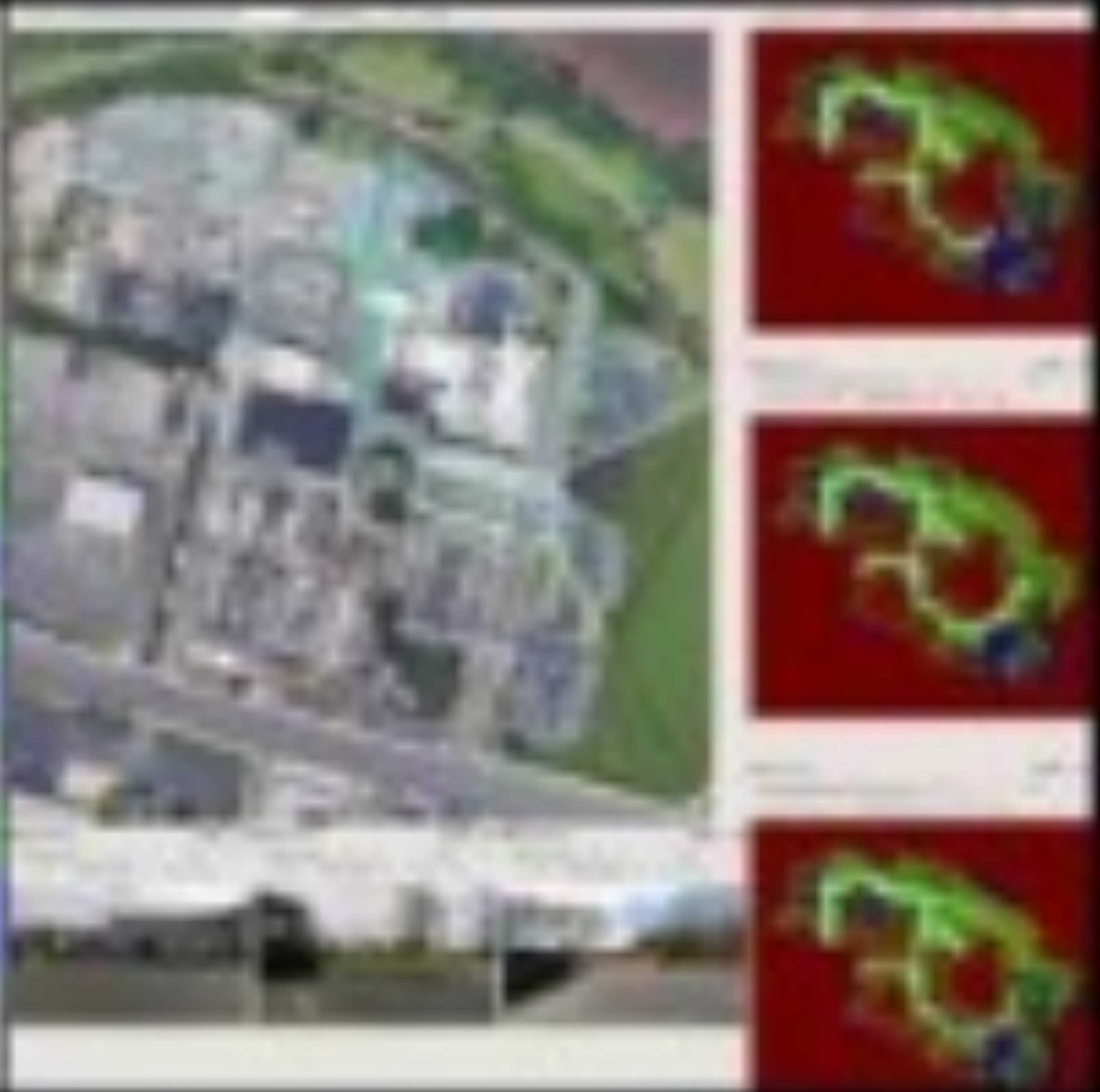
Sensor model

- The global map is first processed by performing semantic segmentation.
- For each type of semantics, they compute the likelihood distance map, here is also called truncated distance field (TDF)
- Generate the TDF features for each type of semantics.
- For the robot measurements, perform the semantic segmentation and compute the TDF again.



5X



Project 4-Paper Reading and Presentation

- Select any one paper published in recent 5 years from the following conferences:
  - Robotics Science and Systems (RSS)
  - IEEE International Conference on Robotics and Automation (ICRA)
  - IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
- Prepare a 10 mins presentation with 2 mins Q&A
  - Nov 18
  - Dec 02
- First come first server
- We can discuss about the paper you found