SLAM-Particle Filter

Jinwook Huh

University of Pennsylvania

with contributions from Bhoram Lee

Penn Engineering | GRASP Laboratory
General Robotics, Automation, Sensing & Perception Lab

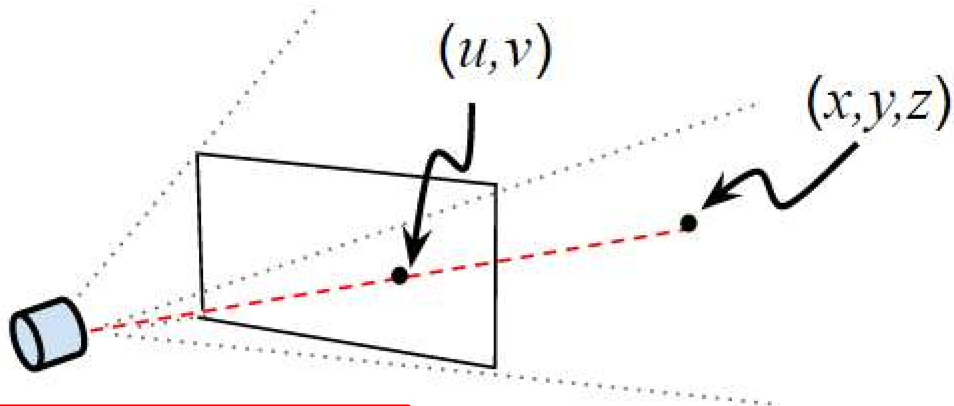
Contents

I. Ground Detection

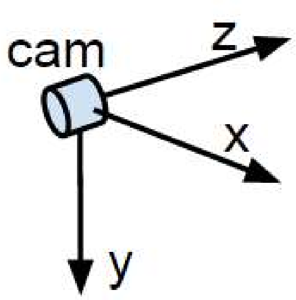
II. RGB-Depth Image Alignment

III. Texture Mapping

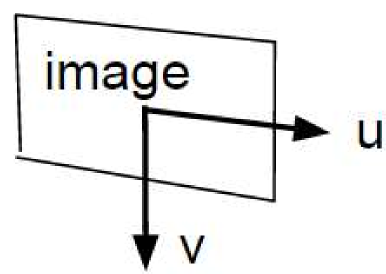
Preliminaries



$$\begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

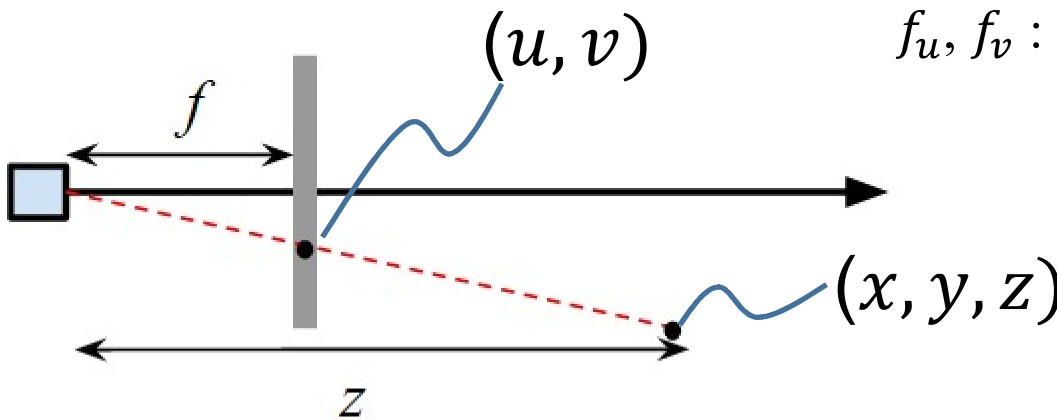


(3D) Camera coordinate frame



(2D) Image coordinate frame

Preliminaries

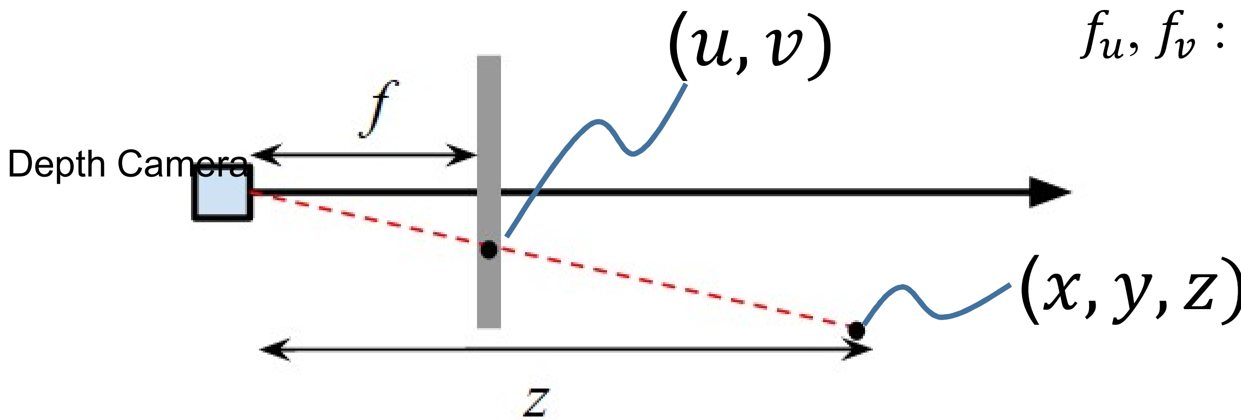


f_u, f_v : focal length

$$\frac{u}{f_u} = \frac{x}{z} \qquad \frac{v}{f_v} = \frac{y}{z}$$

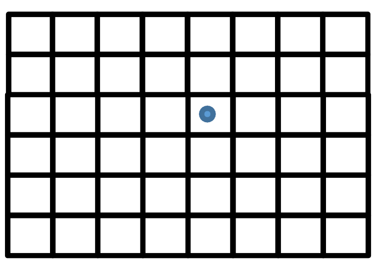
Projection of a 3D point on the image plane

Understanding Depth Image



f_u, f_v : focal length

Depth Image



Pixel value at (u,v): z-value of the point in 3D

Preliminaries

- Plane model in *x-y-z* domain

$$a_0x + a_1y + a_2z + a_3 = 0$$

Projection

$$u = f_u\frac{x}{z} \qquad v = f_v\frac{y}{z}$$

- Can use image (u-v) and inverse depth ($d := z^{-1}$) domain

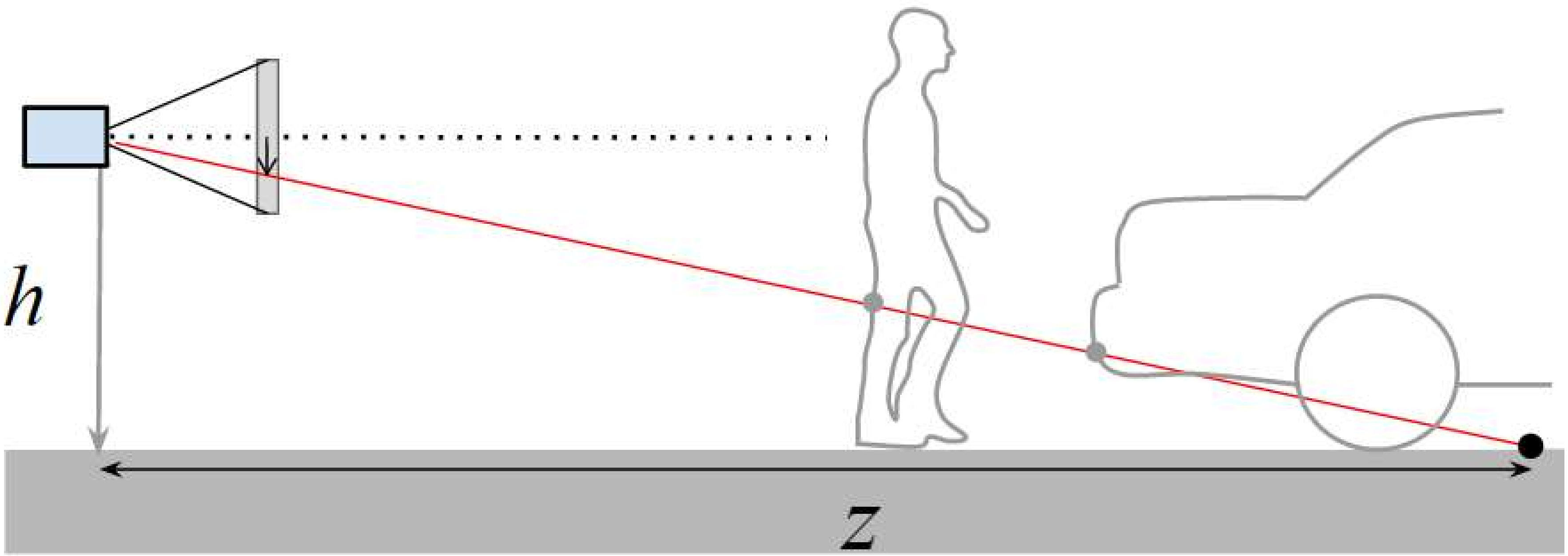
$$a_0'u + a_1'v + a_2' + a_3'd = 0$$

I. Ground Detection
   1) Aligned Camera
   2) Rotated Camera

II. RGB-Depth Image Alignment

III. Texture Mapping

Ground Detection *(1) Aligned Camera*



Ground Plane model

$$y = h$$

Projection

$$\frac{v}{f_v} = \frac{y}{z}$$

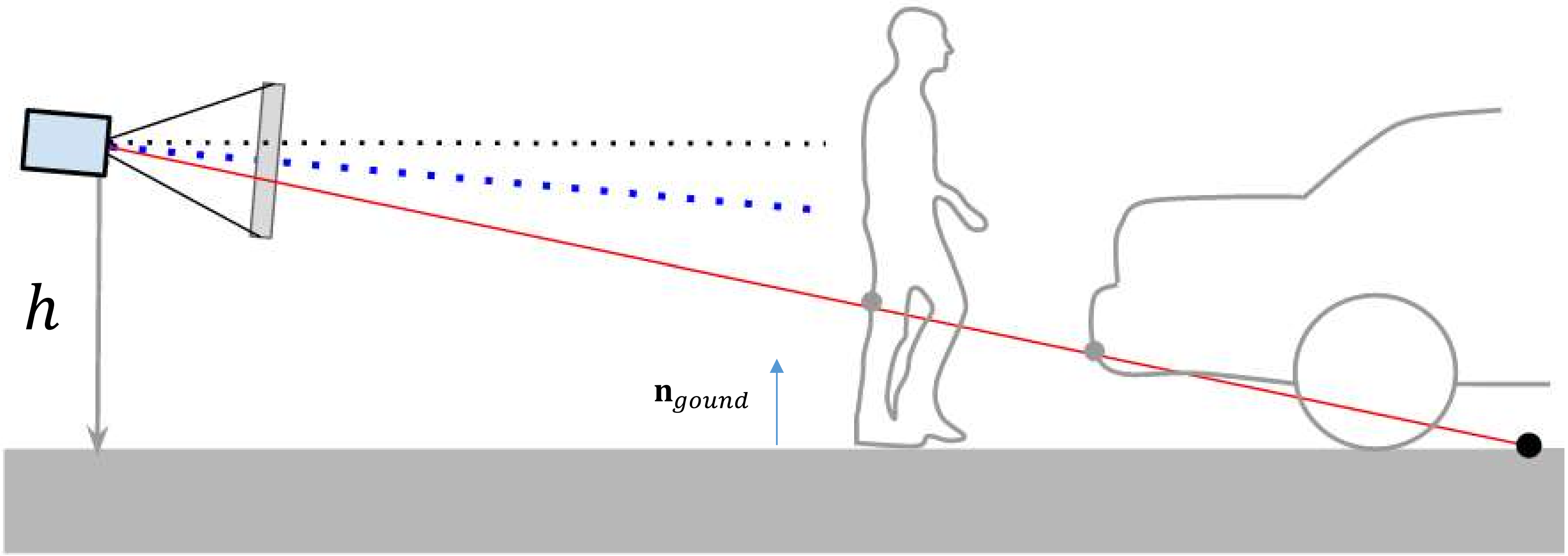
A ground point should satisfy this:

$$\left|\frac{v}{f_v} - \frac{h}{z}\right| < \varepsilon$$

*h,f are constants!

Ground Detection *(2) Rotated Camera*

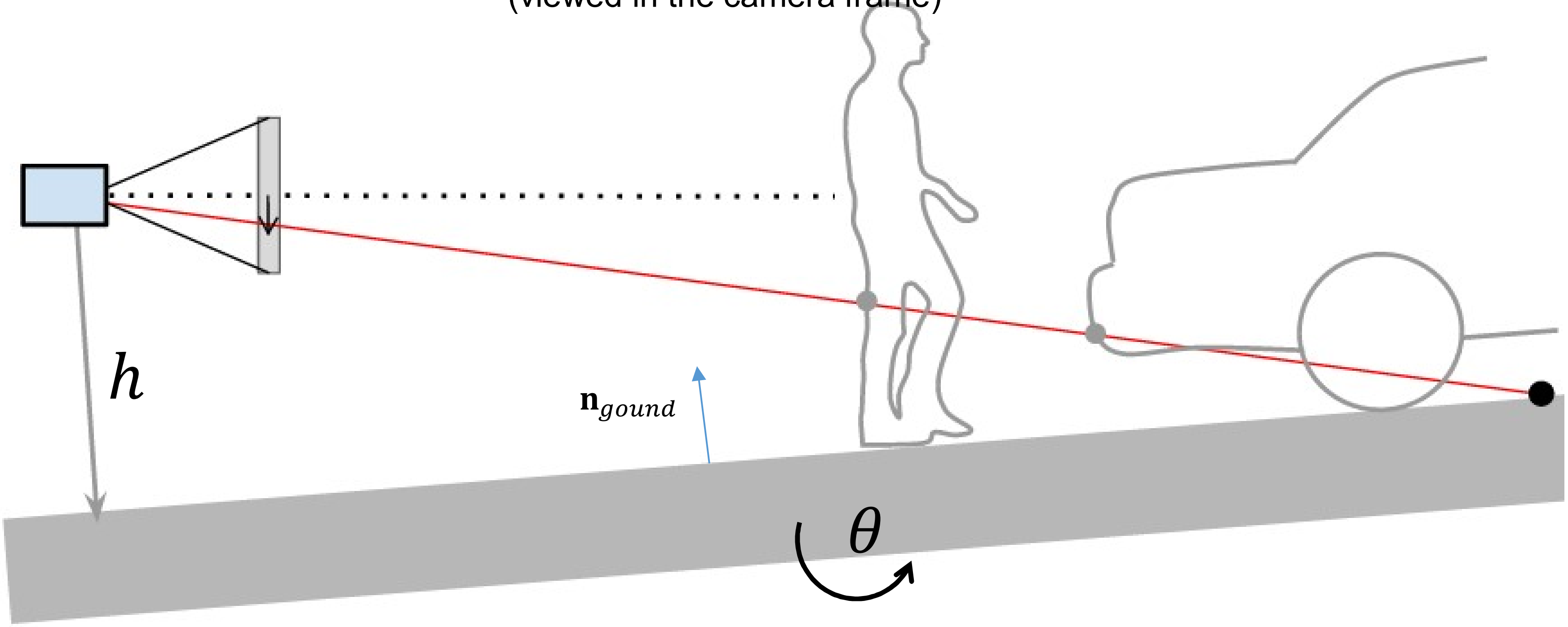
(viewed in the global frame)



$$\mathbf{n}_{gound} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Ground Detection *(2) Rotated Camera*

(viewed in the camera frame)



Ground Plane model

$$a_0 x + a_1 y + a_2 z + a_3 = 0$$

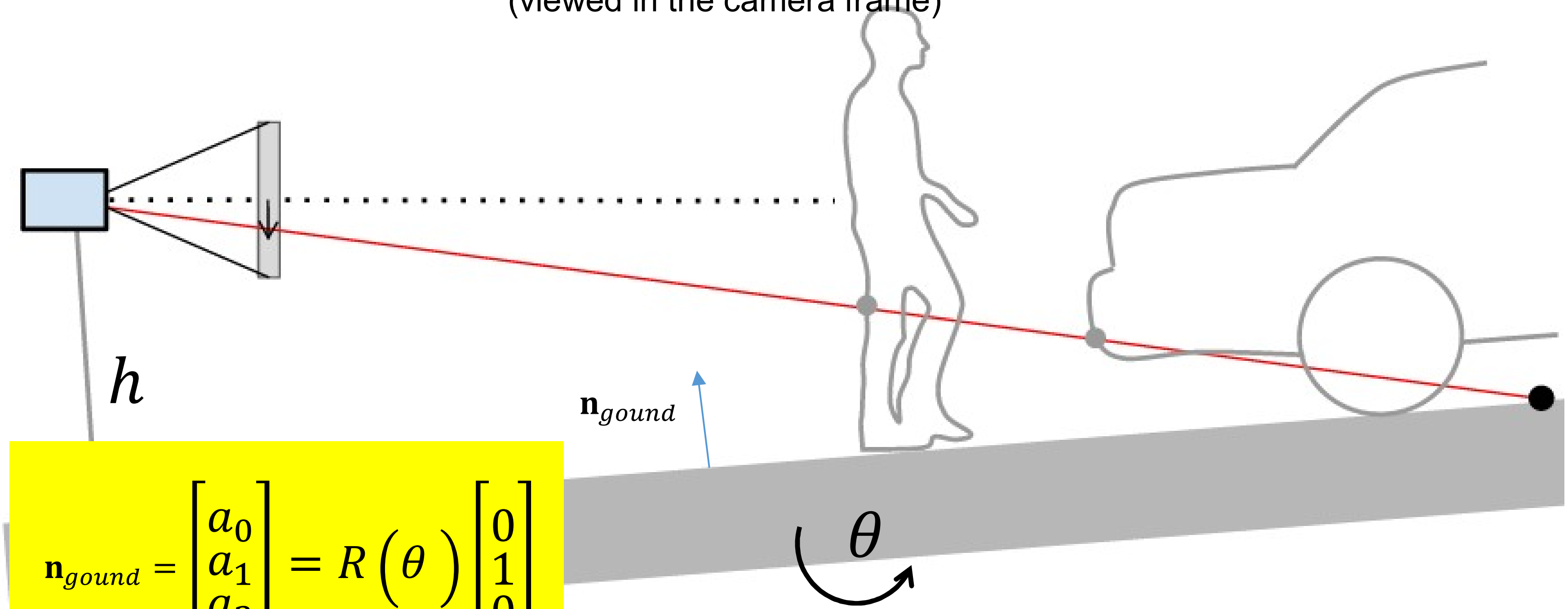
Projectio
n

A ground point should satisfy this:

$$|a_0' u + a_1' v + a_2' + a_3' d| < \varepsilon$$

Ground Detection *(2) Rotated Camera*

(viewed in the camera frame)



$$\mathbf{n}_{gound} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = R(\theta) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$a_3 = -h$$

Ground Plane model

$$a_0 x + a_1 y + a_2 z + a_3 = 0$$

Projection

A ground point should satisfy this:

$$|a_0' u + a_1' v + a_2' + a_3' d \,| < \varepsilon$$

Coefficients are computed from this, where θ is the camera pose.

I. Ground Detection

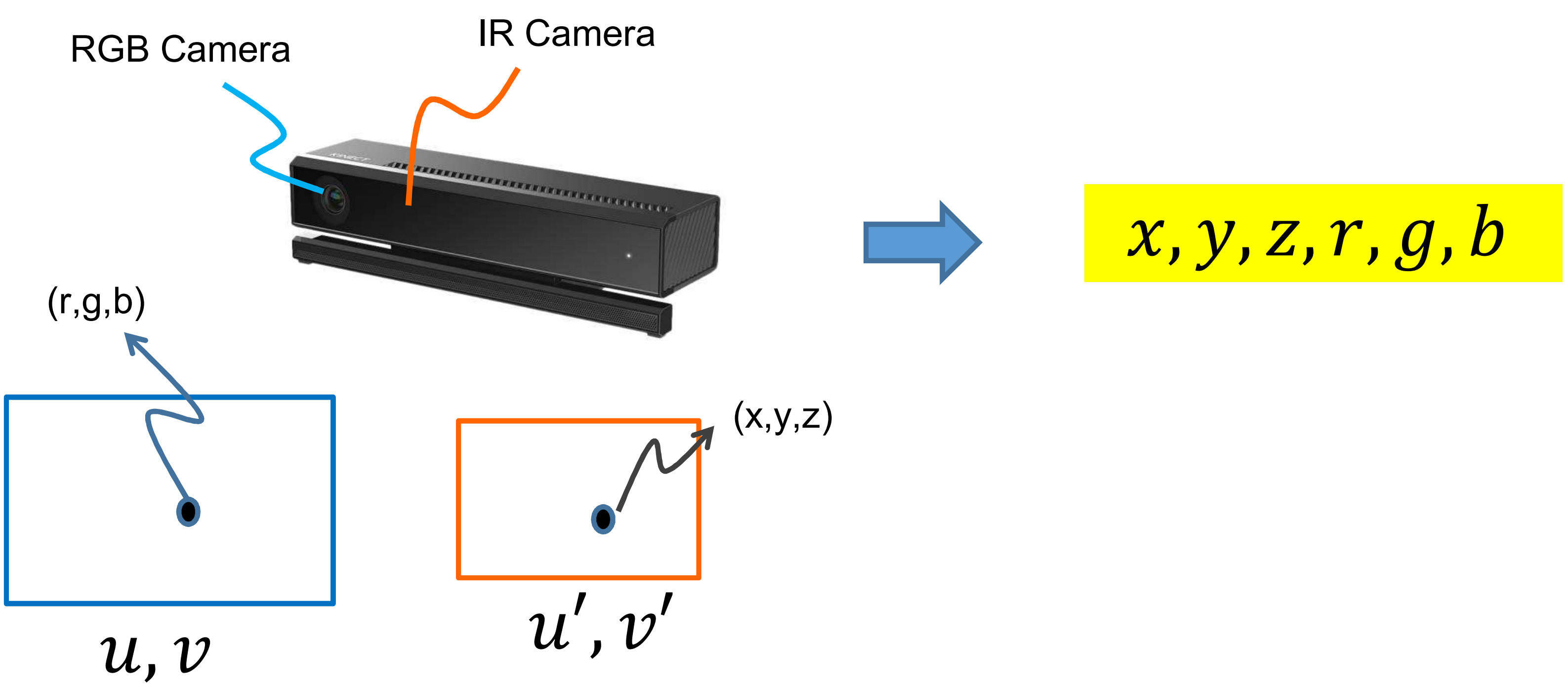
II. RGB-Depth Image Alignment

III. Texture Mapping

RGB-Depth Image Alignment

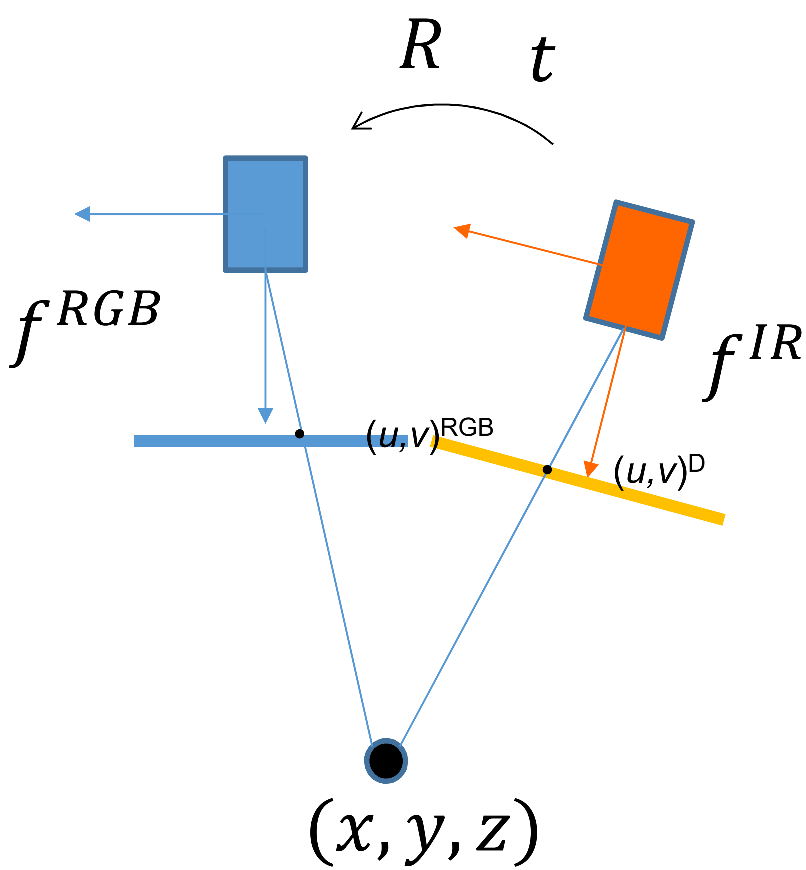
Aim: we want to obtain the tuple of 3d point and its color.

Issue: RGB images are obtained from the RGB camera, Depth images are from the IR camera. They are physically separated, and have different camera parameters.



RGB-Depth Image Alignment

Essential Problem: Given all camera parameters(R,t,f), find the corresponding points of a RGB and a depth image.



RGB-Depth Image Alignment

Recall this: $$\frac{u}{f_u} = \frac{x}{z}$$

1) Compute the 3d coordinate X^{IR} in IR camera frame

$$x^{IR} = uz/f^{IR} \quad y^{IR} = vz/f^{IR}$$

$$X^{IR} = [x^{IR} \quad y^{IR} \quad z^{IR}]$$

2) Transform into the RGB camera frame

$$X^{RGB} = RX^{IR} + t$$

3) Reproject them on to the image plane

$$u^{RGB} = f^{RGB}\frac{x^{RGB}}{z^{RGB}} \quad v^{RGB} = f^{RGB}\frac{x^{RGB}}{z^{RGB}}$$

4) Read (r,g,b) at $(u,v)^{\mathrm{RGB}}$

RGB-Depth Image Alignment

Recall this: $$\frac{u}{f_u} = \frac{x}{z}$$

1) Compute the 3d coordinate X^{IR} in IR camera frame

$$x^{IR} = uz/f^{IR} \quad y^{IR} = vz/f^{IR}$$

$$X^{IR} = [x^{IR} \quad y^{IR} \quad z^{IR}]$$

2) Transform into the RGB camera frame

$$X^{RGB} = RX^{IR} + t$$

3) Reproject them on to the image plane

$$u^{RGB} = f^{RGB} \frac{x^{RGB}}{z^{RGB}} \quad v^{RGB} = f^{RGB} \frac{x^{RGB}}{z^{RGB}}$$

4) Read (r,g,b) at $(u,v)^{RGB}$

This is the color of **this** point.

I. Ground Detection

II. RGB-Depth Image Alignment

III. Texture Mapping

Camera Parameters

- fci : IR camera focal length
- cci : IR camera principal point
- fcc : RGB camera focal length
- ccc : RGB camera principal point

- IR Image : 512 X 424
- RGB image : 1920 X 1080

Procedure

- Find **homogeneous transformation** between global frame and camera frame (based on the best particle)
- **Find 3D points** from depth image w.r.t. camera frame

$$x^{IR} = uz/f^{IR} \quad y^{IR} = vz/f^{IR} \qquad X^{IR} = [x^{IR} \quad y^{IR} \quad z^{IR}]$$

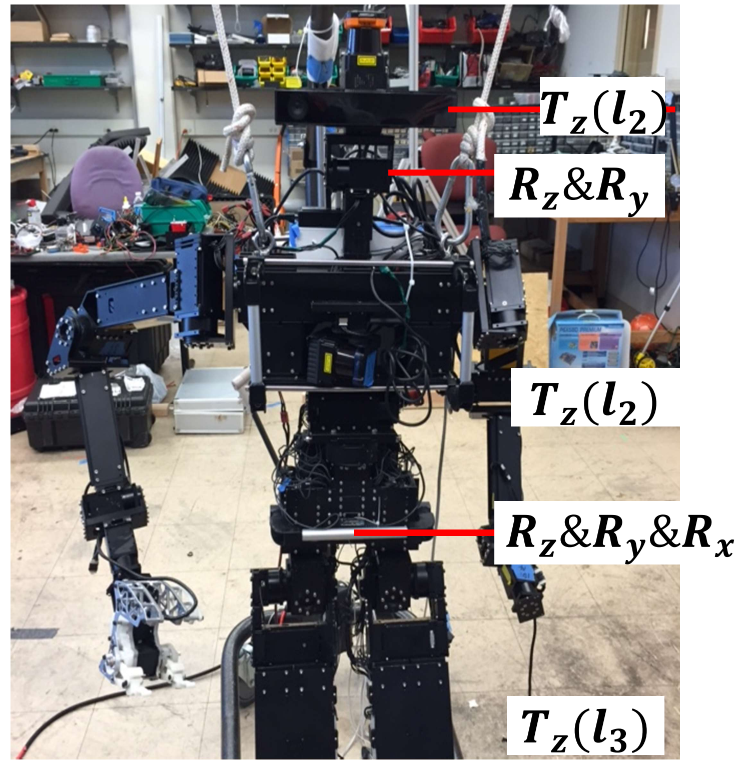
For 'fc' (focal length), the first element corresponds to f_u and the second element corresponds to f_v

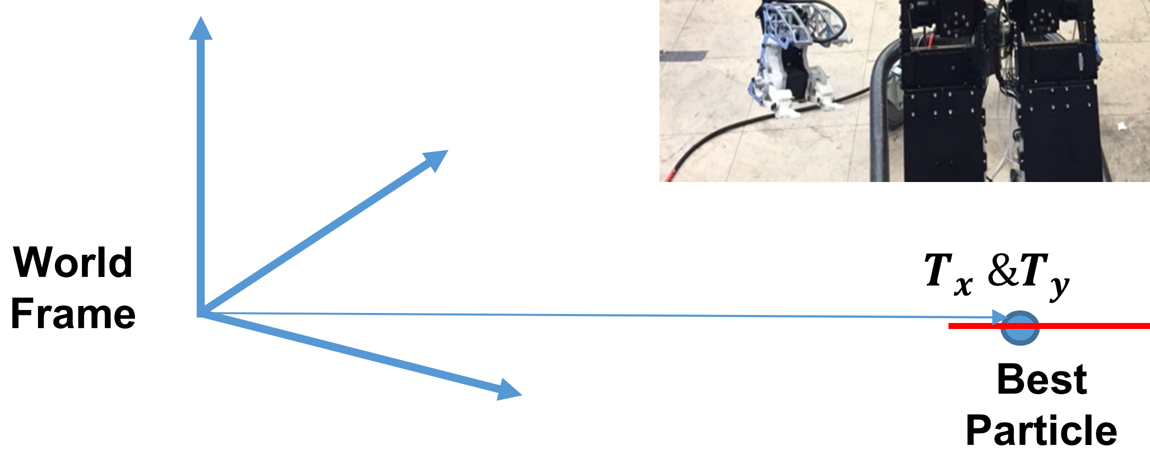
compute appropriate pixel positions using the principal point values before applying the equations

- **Find colors** corresponding to 3D points
- **Transform** 3D points from camera frame **to global frame**
- **Find the ground plane** in the transformed data via RANSAC or **simple thresholding on the height**
- Project color points in the the occupancy grid map

Transformation

- Input
  - $\boldsymbol{pf(x, y, \theta)}$
  - Body roll and pitch: $\boldsymbol{r, p}$
  - Head yaw and pitch

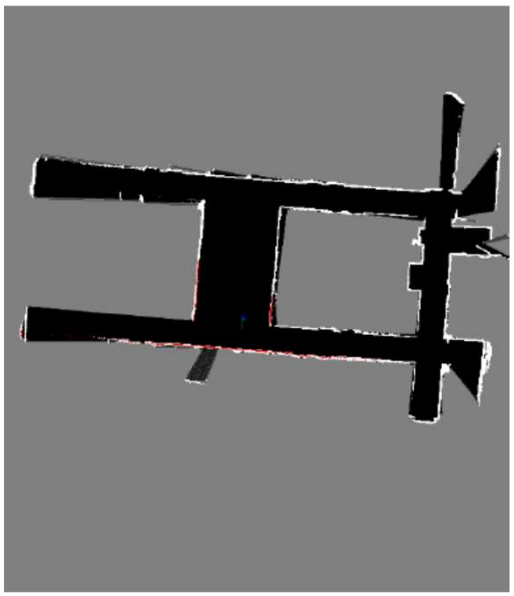


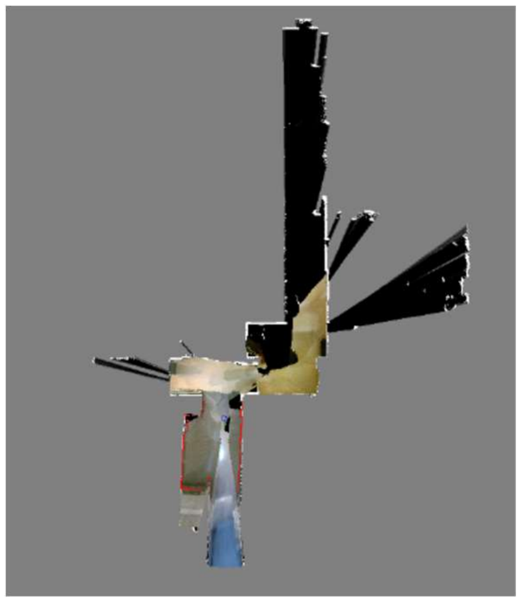


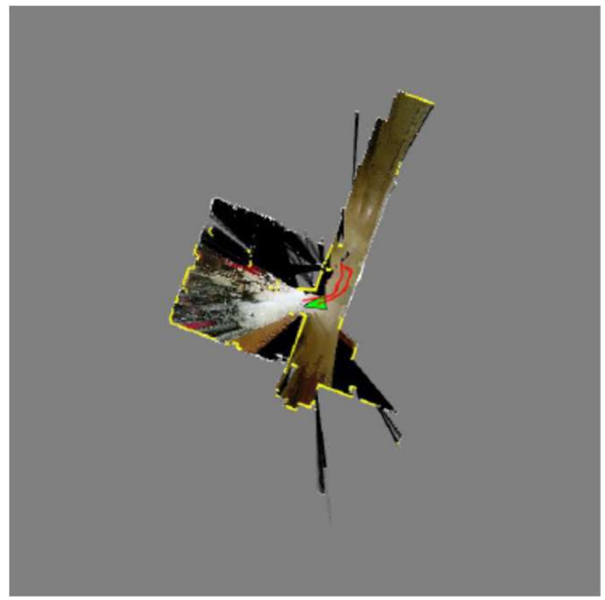
$$\boldsymbol{T = T_{xyz}(pf(x), pf(y), l_3) R_z(pf(\theta)) R_y(p) R_x(r) T_z(l_2) R_z(head\_yaw) R_y(head\_pitch) T_z(l_1)}$$

Project #4 SLAM-PF

- Good examples

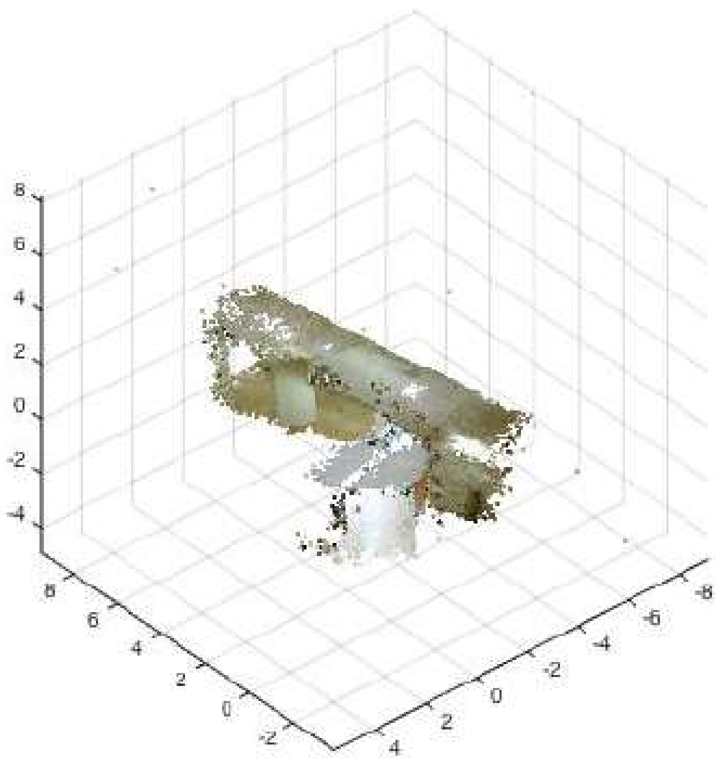


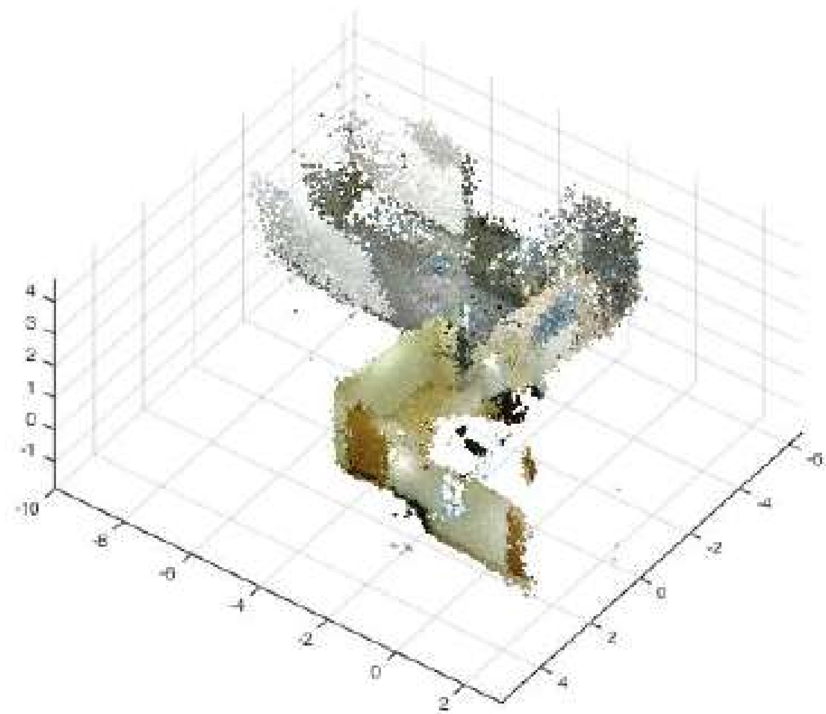




Project #4 SLAM-PF

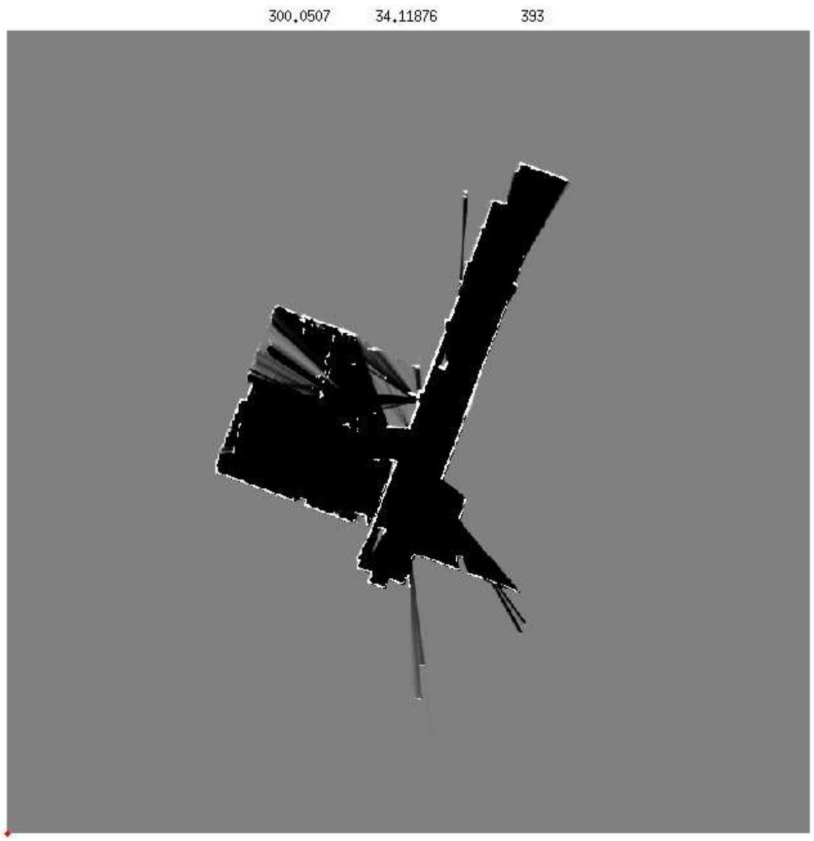
- Previous good examples

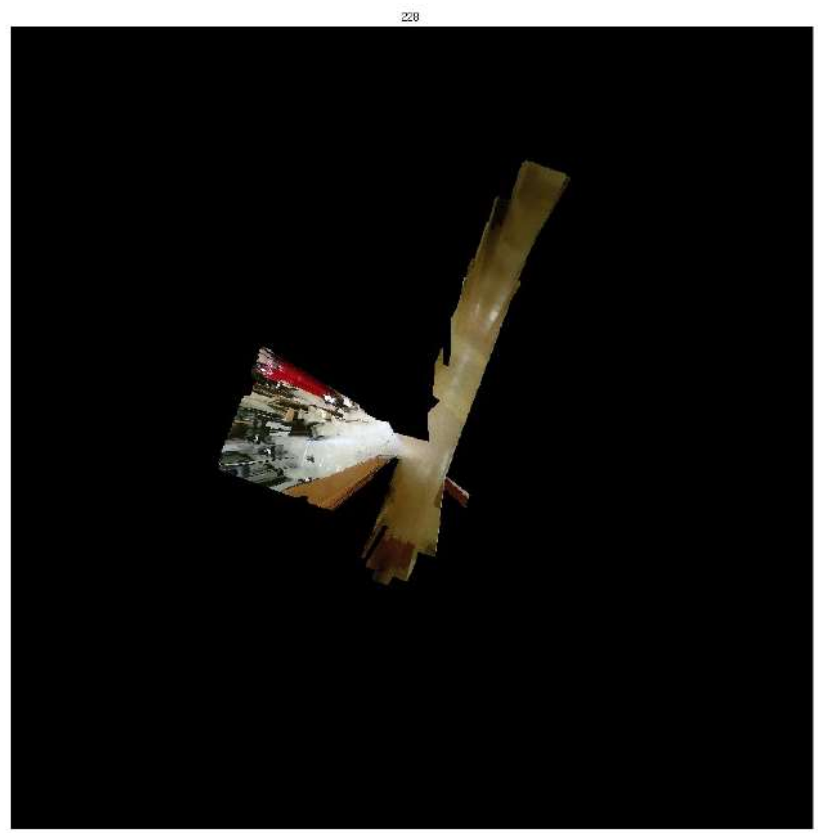




Project #4 SLAM-PF

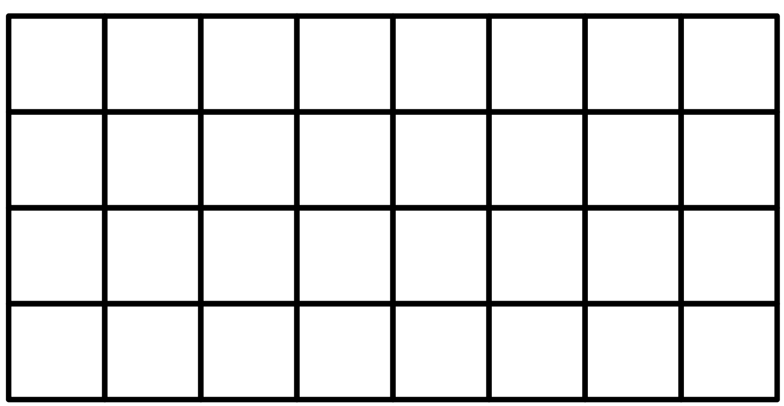
- Previous good examples



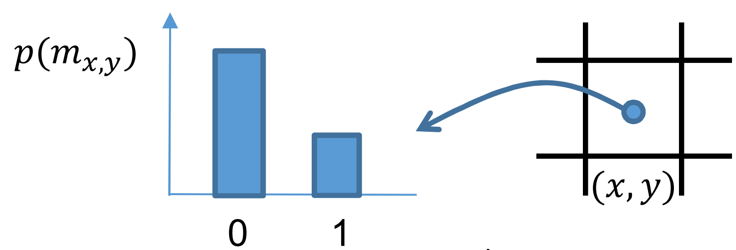


Occupancy Grid Mapping

- Occupancy: binary R.V. $m_{x,y}: \{free, occupied\} \rightarrow \{\,0\,,1\,\}$
- Occupancy grid map
  : fine-grained grid with occupancy variable associated with cell
- Bayesian filtering $p(m_{x,y}|z) \propto p(z|m_{x,y})p(m_{x,y})$
- Usually based on a range sensor



For each cell,
we update $p(m_{x.y}|z)$



Occupancy Grid Mapping

Free

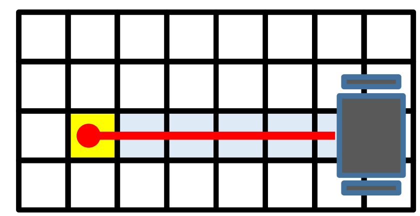
Occupied

- Measurement

$$z \sim \{-1, 1\}$$

- Measurement model

$$p(z|m_{x,y})$$



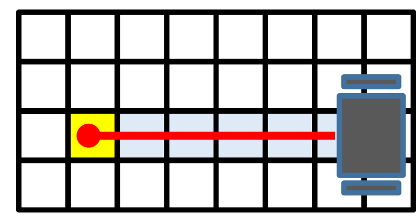
Occupancy Grid Mapping

- Measurement

$$z \sim \{-1, 1\}$$

Free

Occupied



- Measurement model

$$p(z \mid m_{x,y})$$

$$p(z = \ \ 1 \mid m_{x,y} = 1)$$

$$p(z = -1 \mid m_{x,y} = 1) = \ \ 1 - p(z = \ \ 1 \mid m_{x,y} = 1)$$

$$p(z = \ \ 1 \mid m_{x,y} = 0)$$

$$p(z = -1 \mid m_{x,y} = 0) = \ \ 1 - p(z = \ \ 1 \mid m_{x,y} = 0)$$

Occupancy Grid Mapping

- Odd

$$\frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)} = \frac{p(m_{x,y}=1|z)}{1-p(m_{x,y}=1|z)} = \frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}$$

Occupancy Grid Mapping

- Log-odd

$$\log \frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)} = \log \frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}$$

$$= \log \frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)} + \log \frac{p(m_{x,y}=1)}{p(m_{x,y}=0)}$$

$\Rightarrow$ (Log odd) = (Log LH ratio) + (Log prior ratio)

(log odd) ← (log odd) + (log Likelihood model ratio)

Occupancy Grid Mapping

- Example

$$\log odd \mathrel{+}= \log \frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)}$$

Initially,

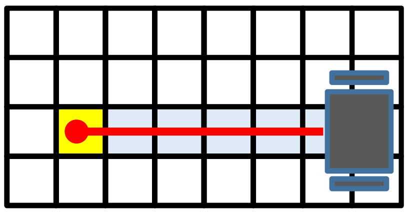
$\log odd = 0$ for all (x,y)

$$p(m_{x,y}=1) = p(m_{x,y}=0) = 0.5$$

Measurement Model

$$\log odd\_occ := \log \frac{p(z=1|m_{x,y}=1)}{p(z=1|m_{x,y}=0)} = \log \frac{0.7}{0.2}$$

$$\log odd\_free := \log \frac{p(z=-1|m_{x,y}=0)}{p(z=-1|m_{x,y}=1)} = \log \frac{0.8}{0.3}$$



Occupancy Grid Mapping

- Example (continued)

<u>Case I : cells with z=1</u>

$$\log odd \leftarrow \log odd + \log odd\_occ$$

$$\log odd \leftarrow \log odd + \log(\frac{0.7}{0.2})$$

<u>Case II : cells with z=-1</u>

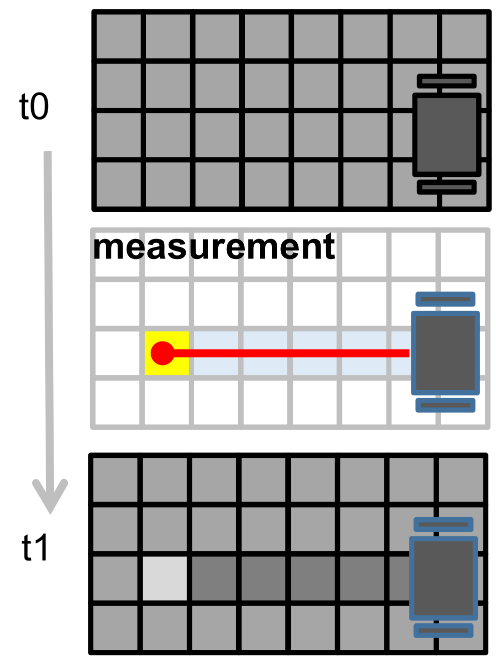
$$\log odd \leftarrow \log odd - \log odd\_free$$

$$\log odd \leftarrow \log odd - \log(\frac{0.8}{0.3})$$

<u>Case III : cells with no z</u>

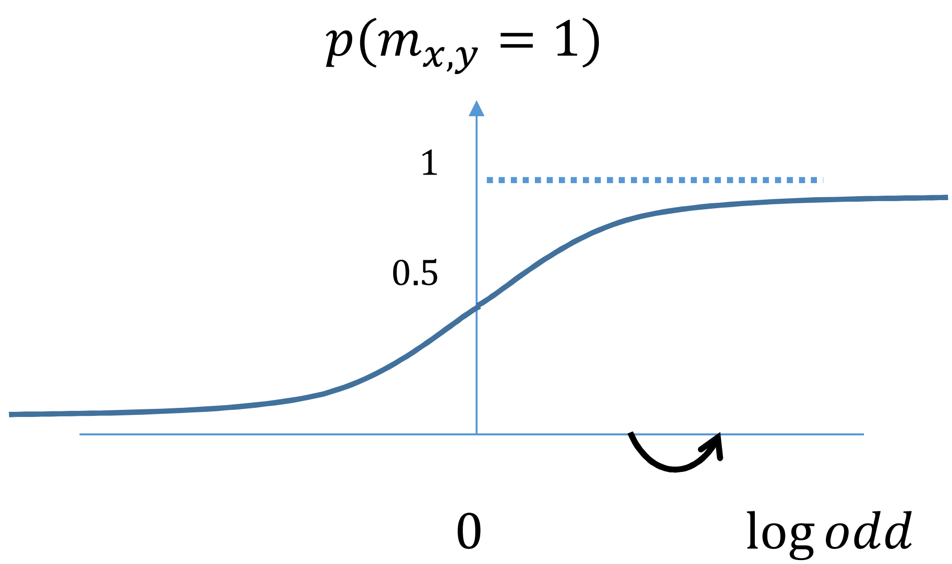
$$\log odd = 0$$

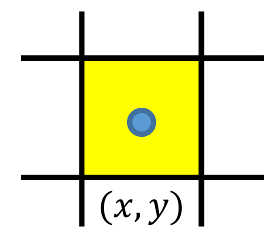
$$\log odd += \log \frac{p(z|m_{x,y} = 1)}{p(z|m_{x,y} = 0)}$$



Occupancy Grid Mapping

- In summary… for cells with z = 1

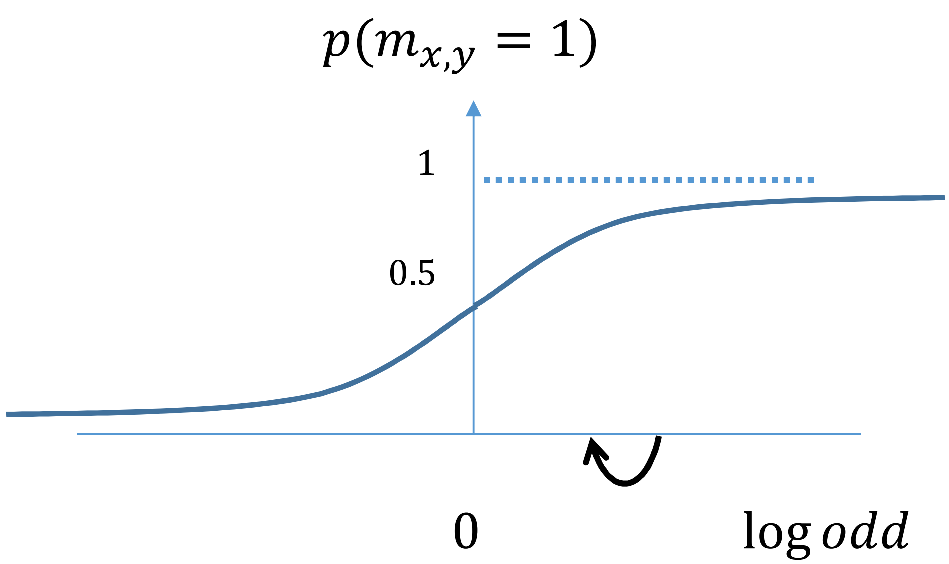


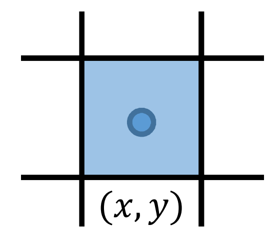


$$\log odd \mathrel{+}= \log odd\_occ$$

Occupancy Grid Mapping

- In summary... for cells with z = -1

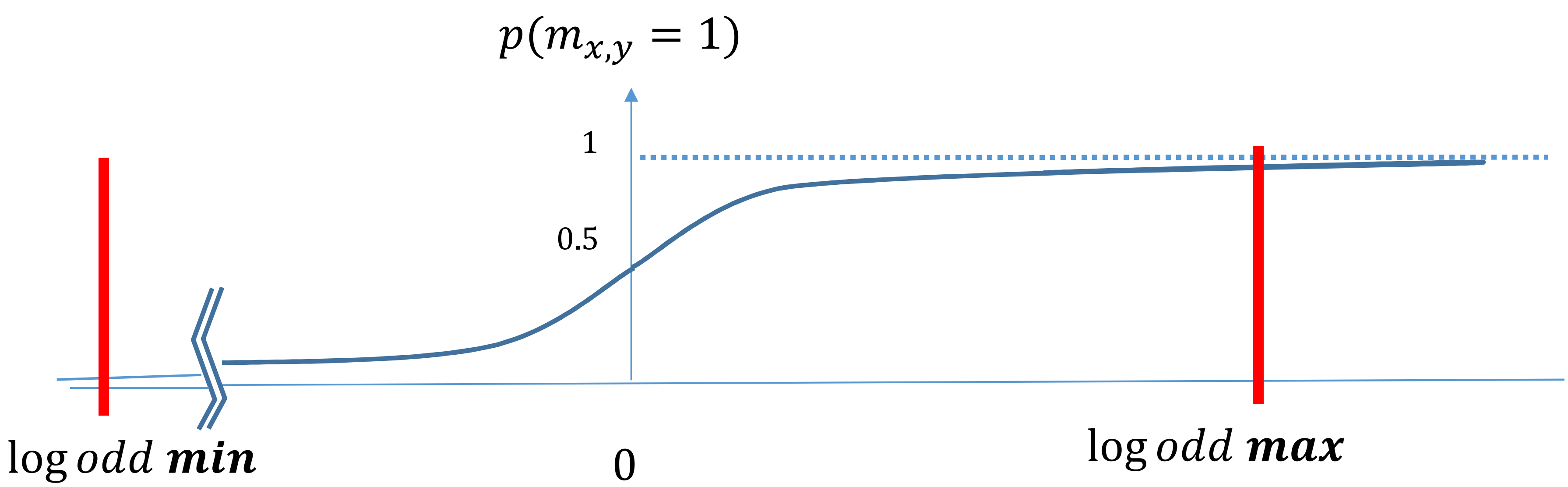




$$\log odd -= \log odd\_free$$

Occupancy Grid Mapping

- Tips : Never make anything certain! Saturate log-odd.



Map Registration

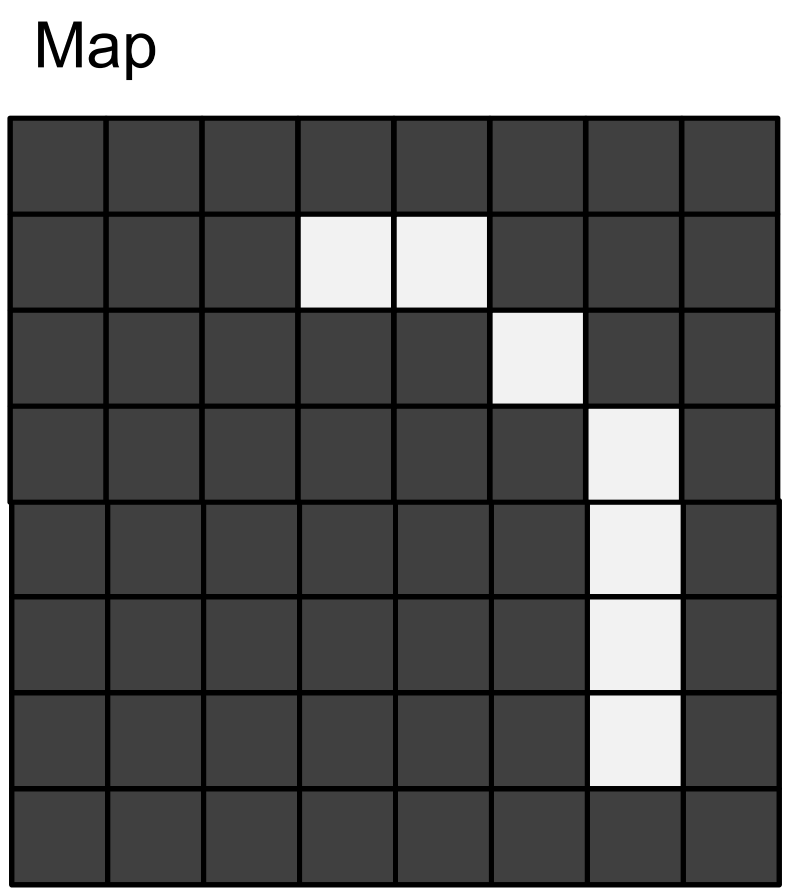
- Correlation-based Matching

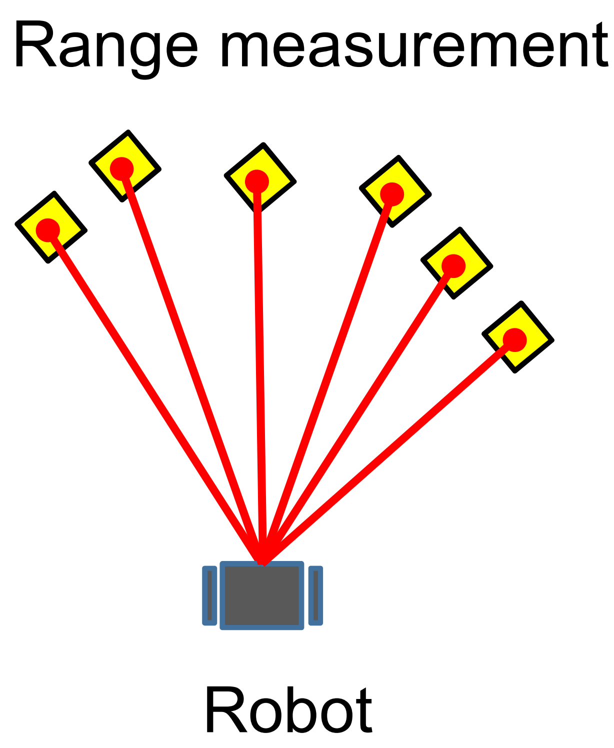
    1) General hypotheses

    2) Evaluate hypotheses

    3) Pick the best

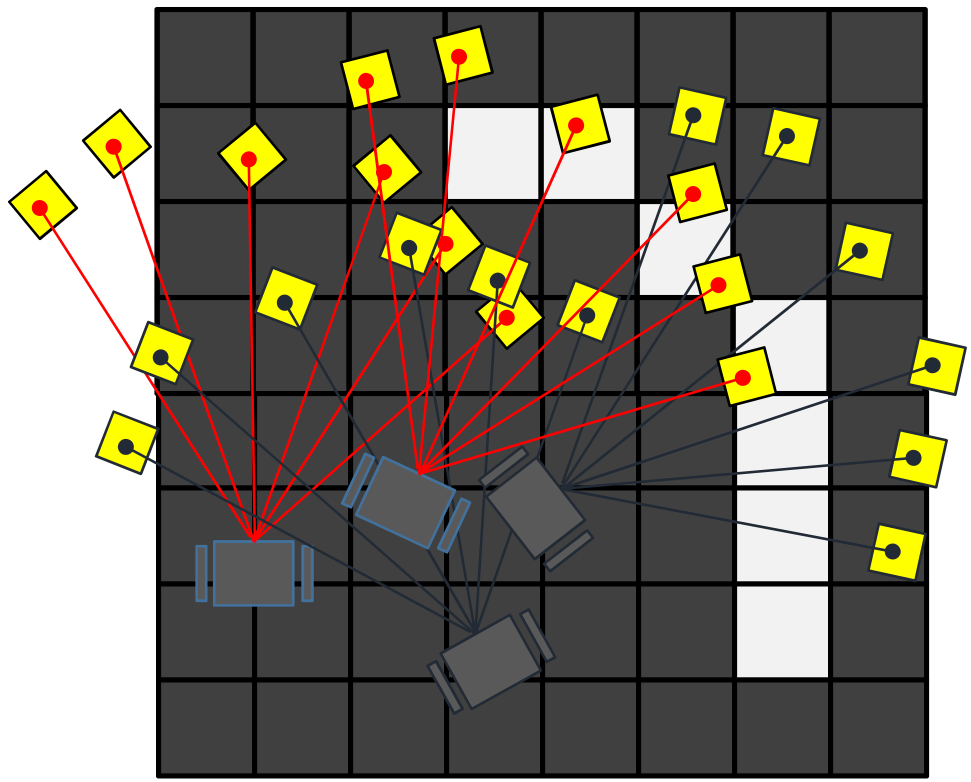
Map Registration





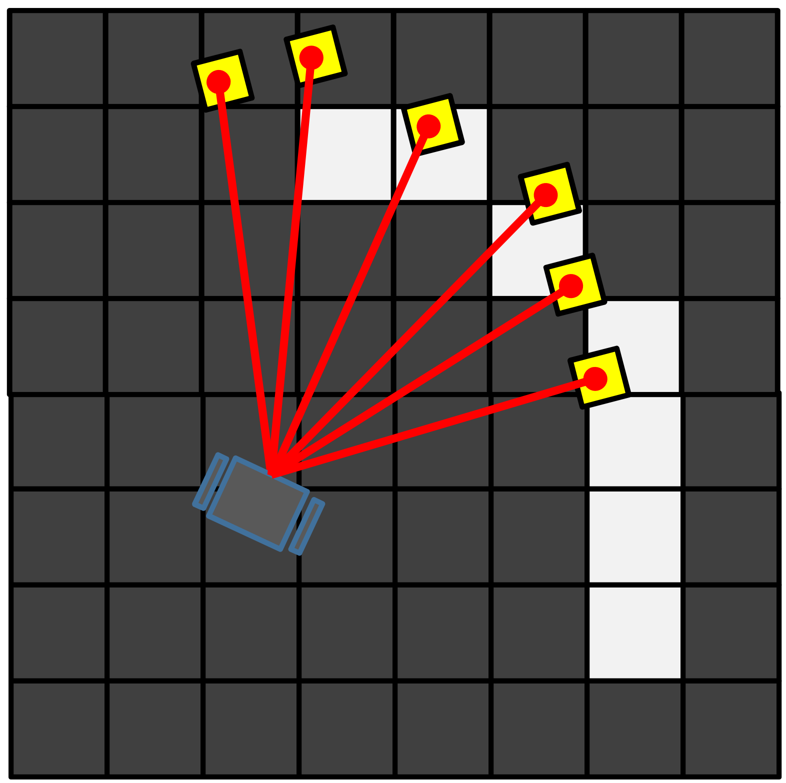
Map Registration

- Correlation-based Matching
  - Generate hypotheses (particles)



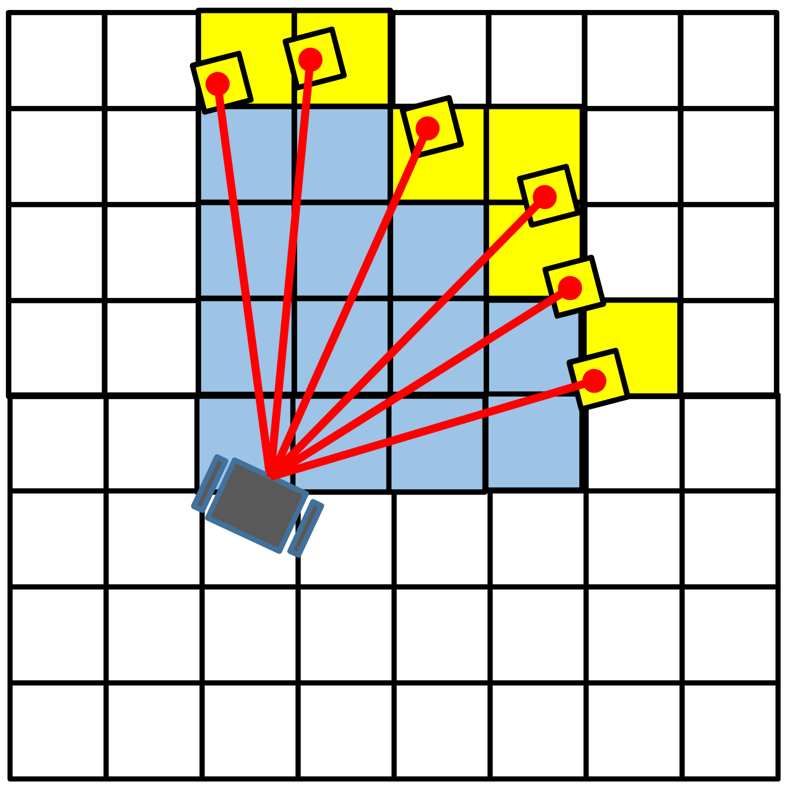
Map Registration

- Correlation-based Matching
    - For each hypothesis



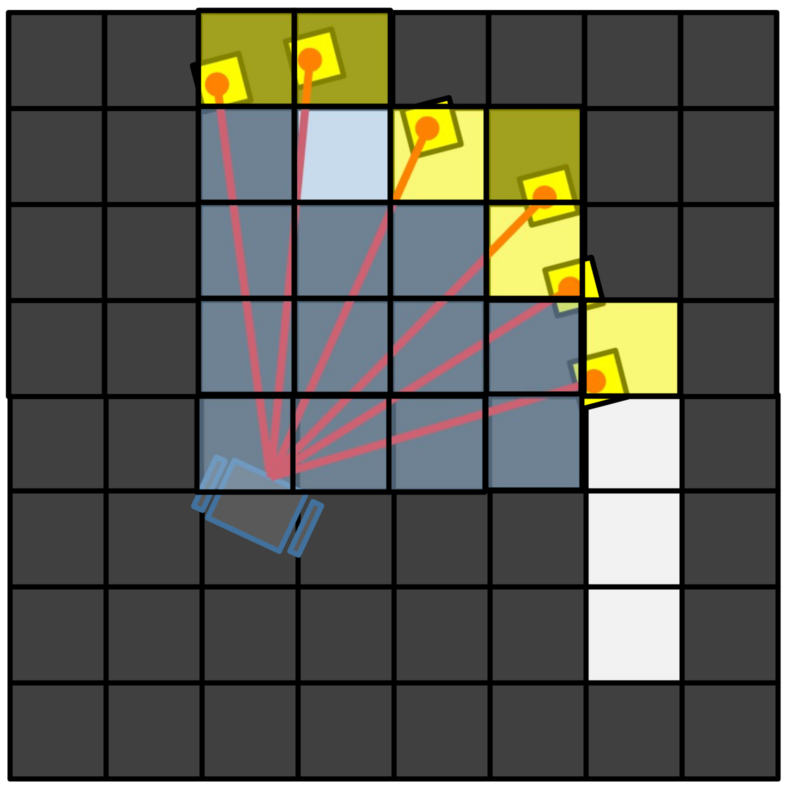
Map Registration

- Correlation-based Matching
  - Build a local map from the measurement in a form that can be compared with the global map



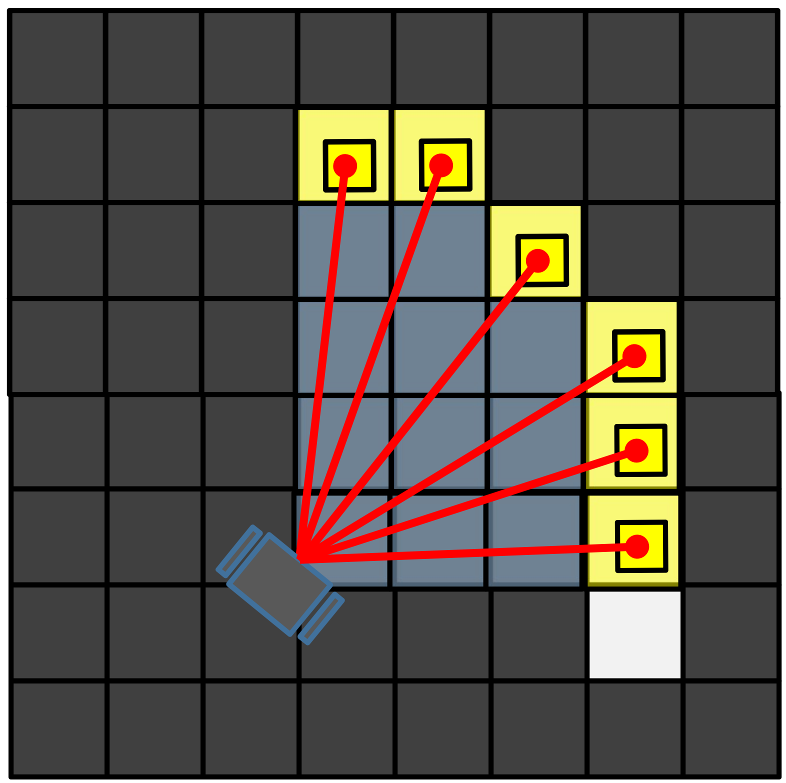
Map Registration

- Correlation-based Matching
  - Evaluate hypotheses
    - — score the hypothesis

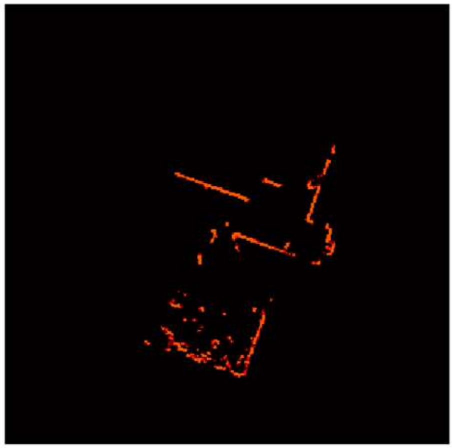


Map Registration

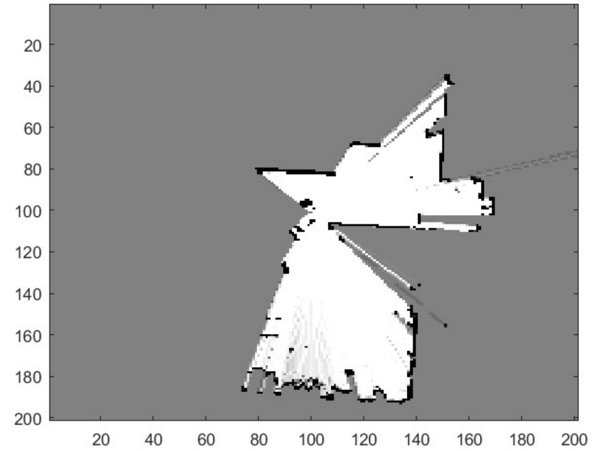
- Correlation-based Matching (Find the best*)
  - Among all the hypotheses, choose the one that has the largest score in order to represent your current location



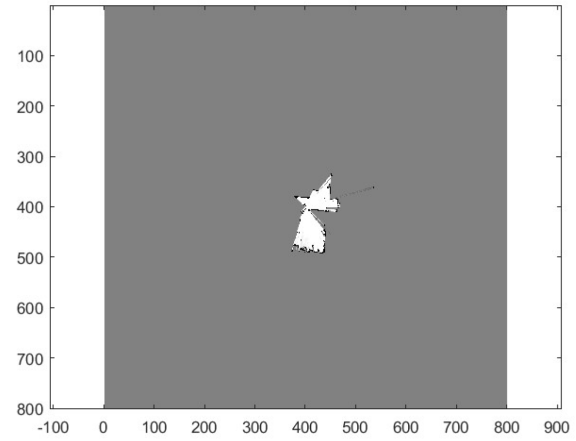
Map Registration



Lidar



Local Map



Local map in global frame

Update Global Map

- Global map
  - Uniform grid map/Quad-tree map
  - Accuracy depends on grid resolution

Map correlation

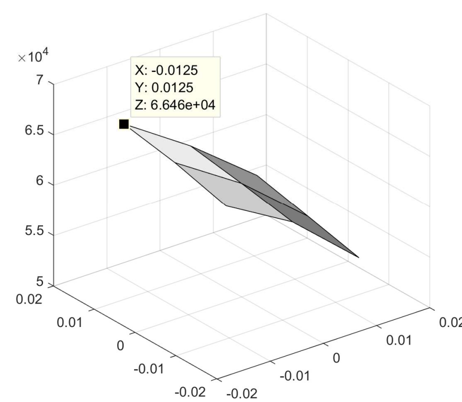
- MU.mapCorrelation(MAP['map'],x_im,y_im, pReading_world[0:3,:], x_range, y_range)
  - Evaluate the correlation of a particle in the patch (X_range, yrange)
- MAP['map'] : Global Map (int8)
- x_im, y_im : physical x, y positions of the grid map cells (unit: m)
- pReading_world: occupied x, y positions from lidar based on a particle w.r.t. world frame (unit: m)
  - (np.concatenate([np.concatenate([xs0,ys0],axis=0),np.zeros(xs0.shape)],axis=0)

Map correlation

- Compute the weight based on the map correlation function
  - C = MU.mapCorrelation(MAP['map'],x_im,y_im, pReading_world[0:3,:], x_range, y_range)
- x_range,y_range : physical x, y, positions you want to evaluate "correlation“
- pReading_world : lidar scan data based on a particle frame w.r.t global frame
- c = map_correlation(pp, GM.grid, GM.grid, [pReading_world; zeros(1, numReading)], x_range, y_range);
- weight of each particle : max(c(:));

MAP['map']

Lidar scan base on a particle



c

example

- 25m X 25m, resolution 0.05m map
- c = MU.mapCorrelation(MAP['map'],x_im,y_im,Y[0:3,:],x_range,y_range)
- MAP['map']: 1000 X 1000 int8
- X_im : 1X1001 vector (1:0.05:25)
- Y : 3X1081 scan data w.r.t. world frame (based on the particle)
  - Note that elements in last row of Y (Z values) are zero
- x_range, y_range : [-0.05 0 0.05] or [-0.025 -0.0125 0 0.0125 0.025]

  : the range you want to evaluate

weight of each particle : max(c(:))

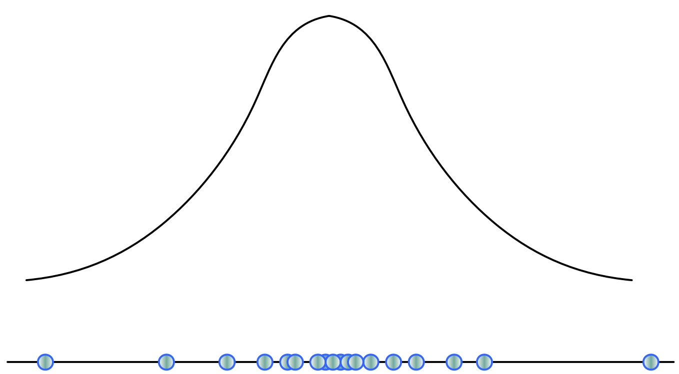
Update weights/Best particle

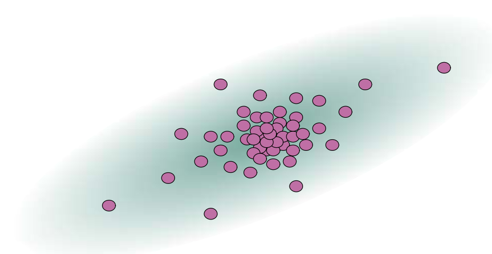
- Multiplication
  - pfWt(i) = pfWt(i)*maxC(i);
- normalization
  - pfWt = pfWt / sum(pfWt);
- Choose the best particle based on the weight
  - [~, as] = max(pfWt);
  - pfBest = pf(as, :);

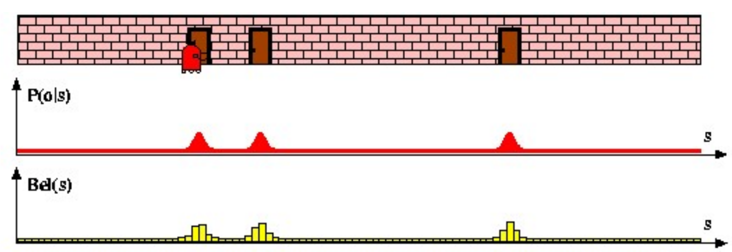
3. PF-based SLAM

Particle Filter

- Non-parametric model (multimodal)
  - Mixtures of Gaussians, multi-hypothesis Kalman Filter
- Uses particles instead of probability distribution
- Fast and efficient

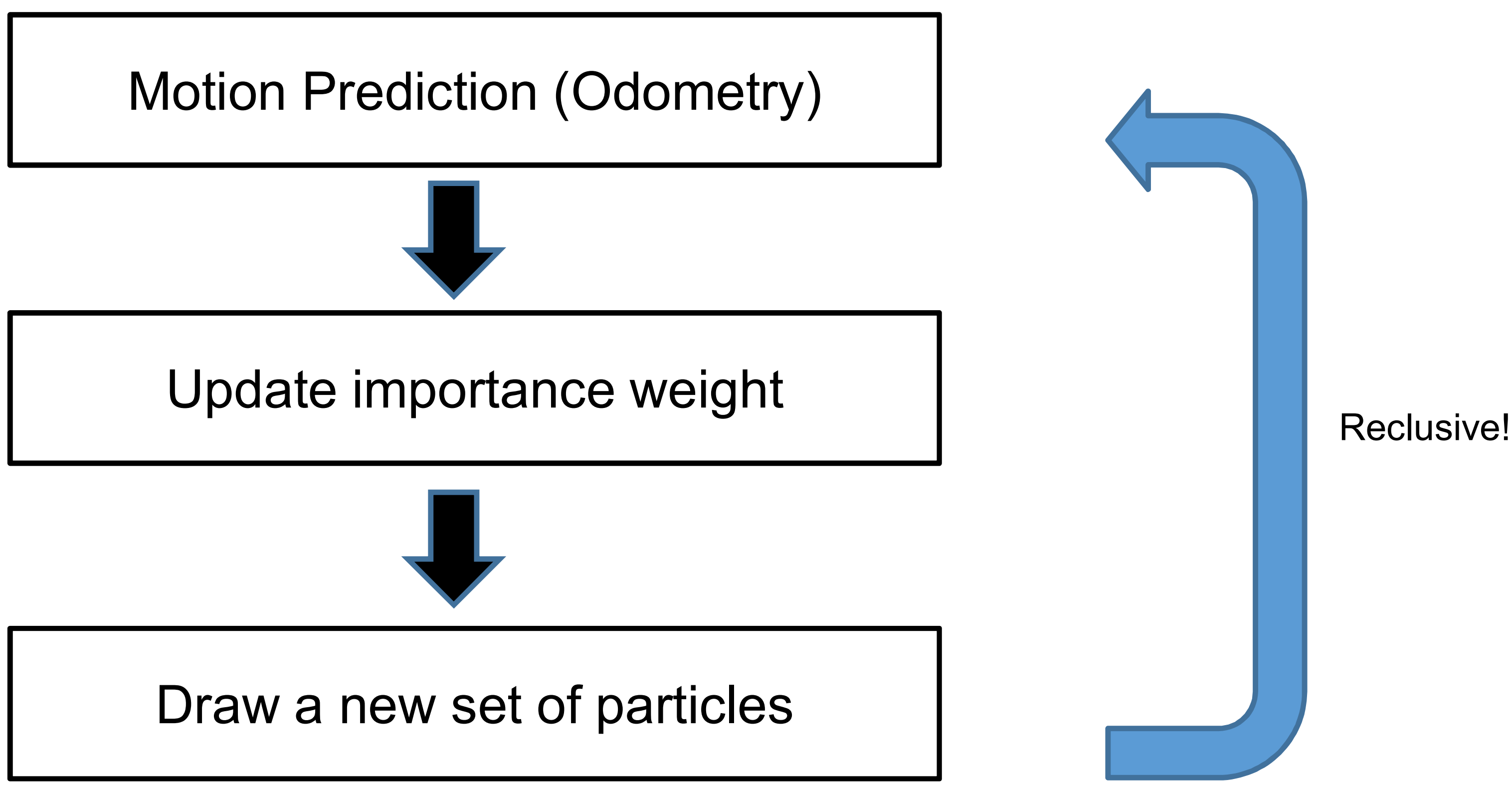






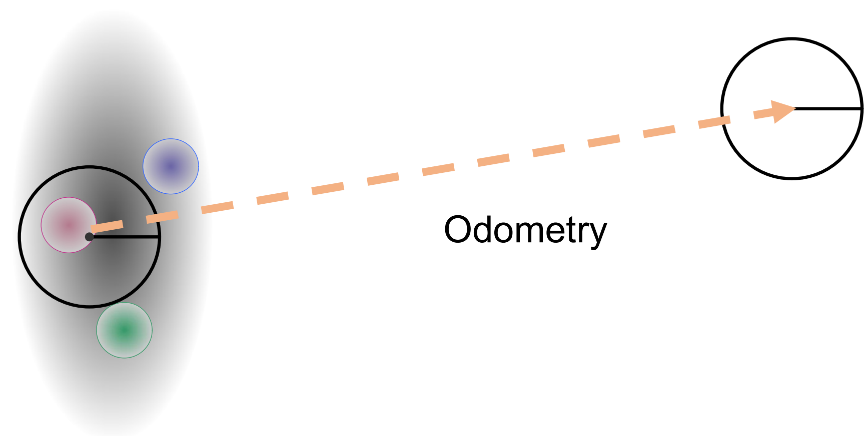
Particle Filter

- Flowchart



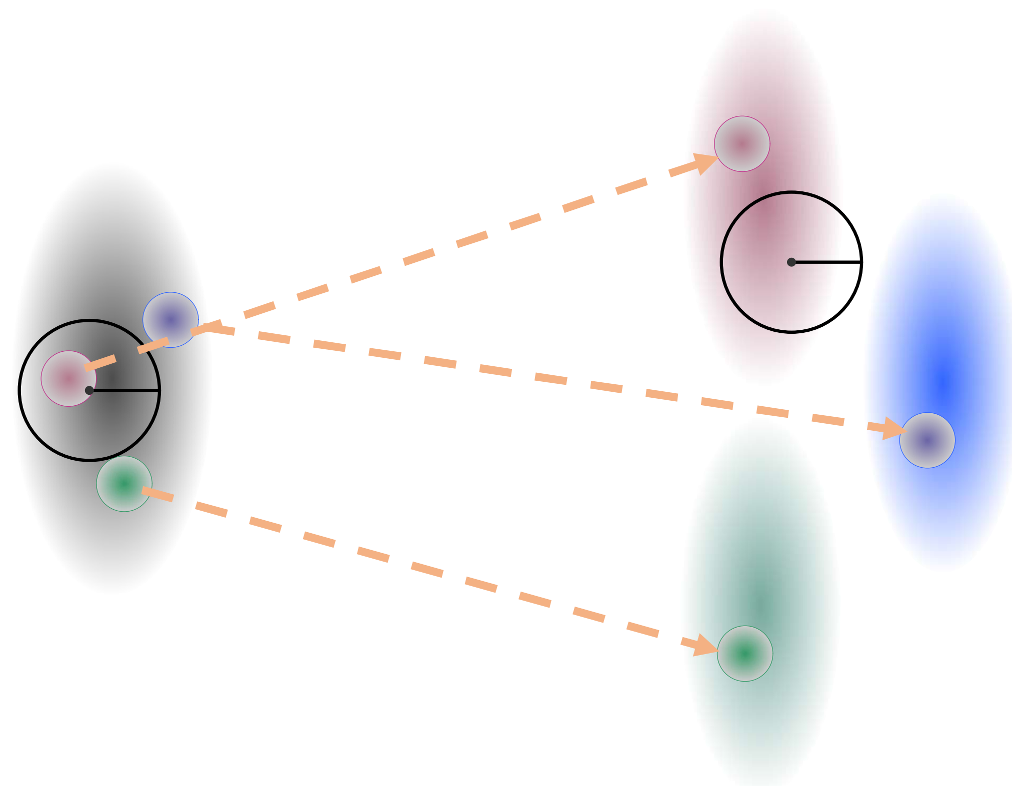
1st step: New pose from motion

New pose given new control



1st step: New pose from motion

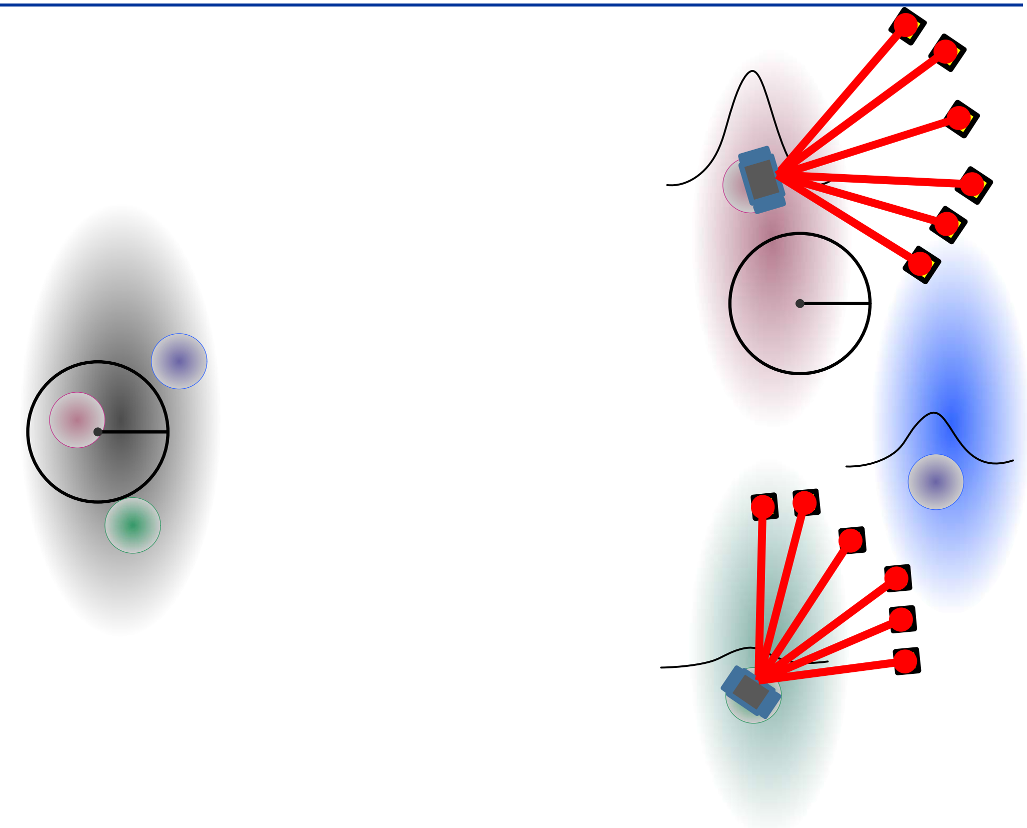
New pose given new control



2nd step: Importance weight

New pose given new control

Importance weight



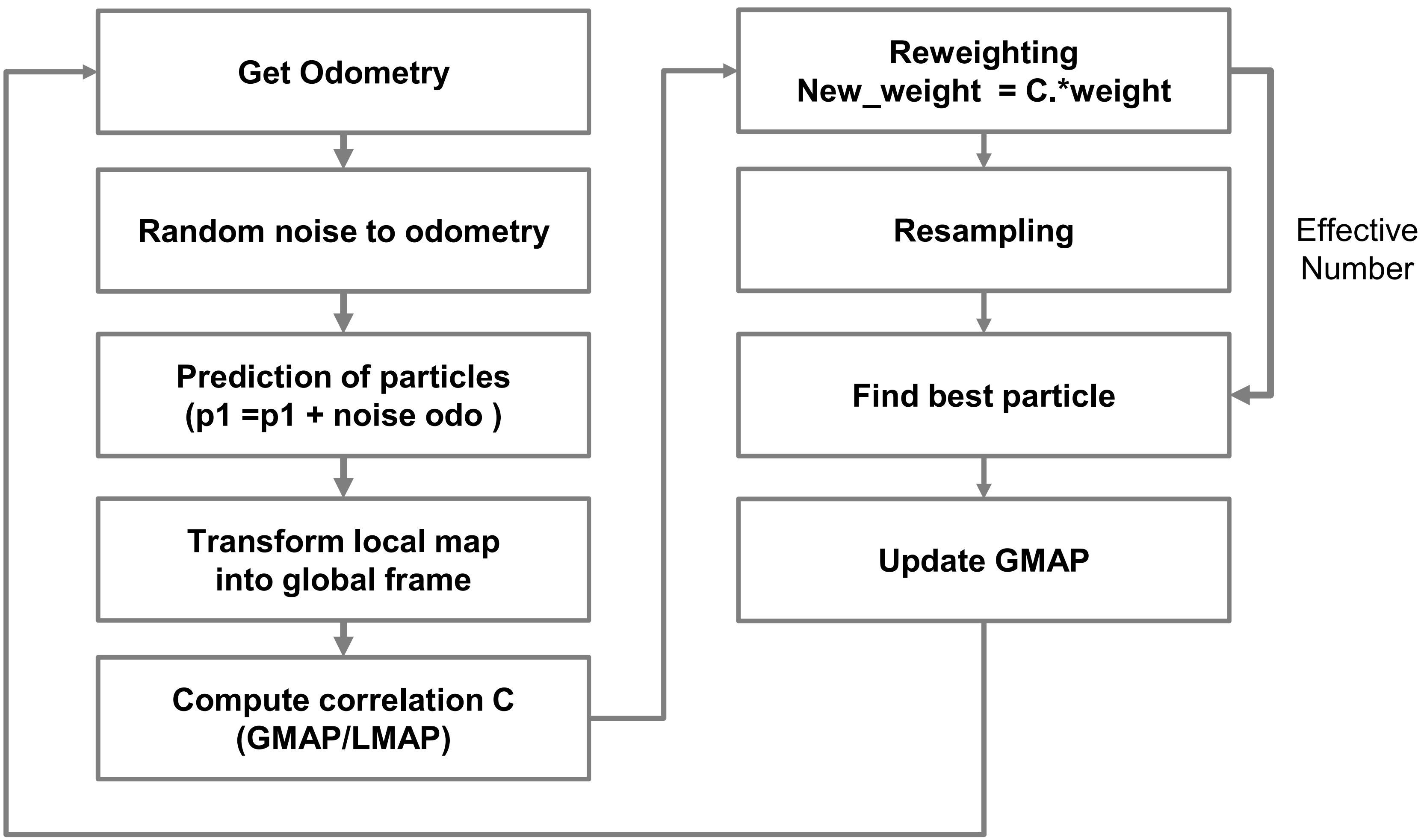
3rd step: Resampling

New pose given new control

Importance weight

Draw a new set of particles

PF-Based SLAM

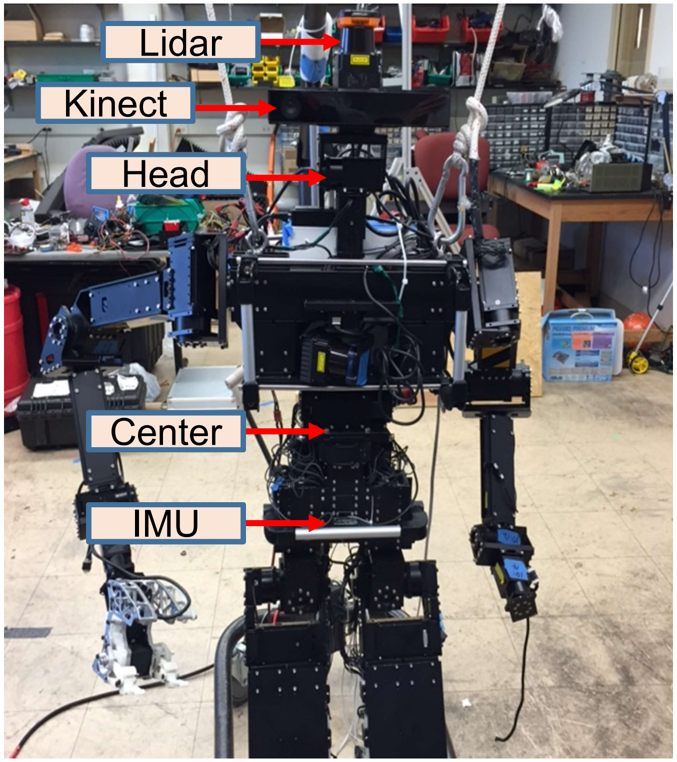


Project #4 SLAM-PF

- Humanoid SLAM
  - 2D SLAM based on several sensors
  - Odometry, Lidar (Head), IMU, Vision (Kinect)

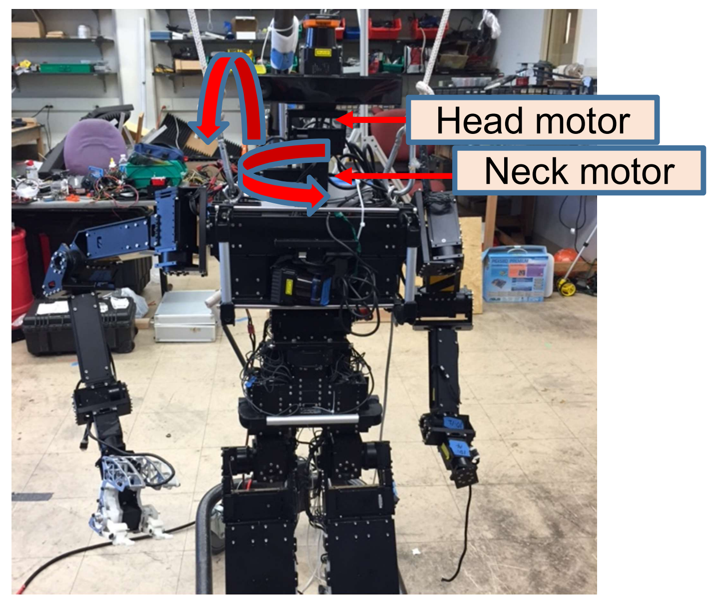
- Ground Detection
  - Visualization of the detected ground

- Difficulties
  - 3D jerky motion of the THOR
  - Roll & Pitch motion
  - Head motion
  - Moving obstacles



Project #4 SLAM-PF

- 3D jerky motion of the THOR
  - Roll and Pitch motion
  - Poor odometry
- Head motion while moving
  - Compensate roll and yaw motion
  - Remove the lidar scan of the ground
- Relative coordinate
  - IMU, Lidar, Kinect coordinates
- Etc.
  - Moving obstacles



Project #4 SLAM-PF

- Training Data set(#0 ~ #3)
  - Train_lidar.mat
  - Train_joint.mat
  - RGB.mat
  - Depth.mat

- Maputils
  - Map_correlation
  - GetMapCellsFromRay

Project #4 SLAM-PF

- Joint.mat
  - Joint angles
  - pos: Matrix of positions (Maybe you don't need)
  - ts: timestamps (absolute time)
  - gyro: angular velocity of body
  - acc: acceleration of body
  - rpy: roll, pitch and yaw angles of body
  - Head_angles = [Neck angle(yaw), head angle(Pitch)];
  - Ft_l, ft_r : Torque/Force sensor of left and right foot (Maybe you don't need)

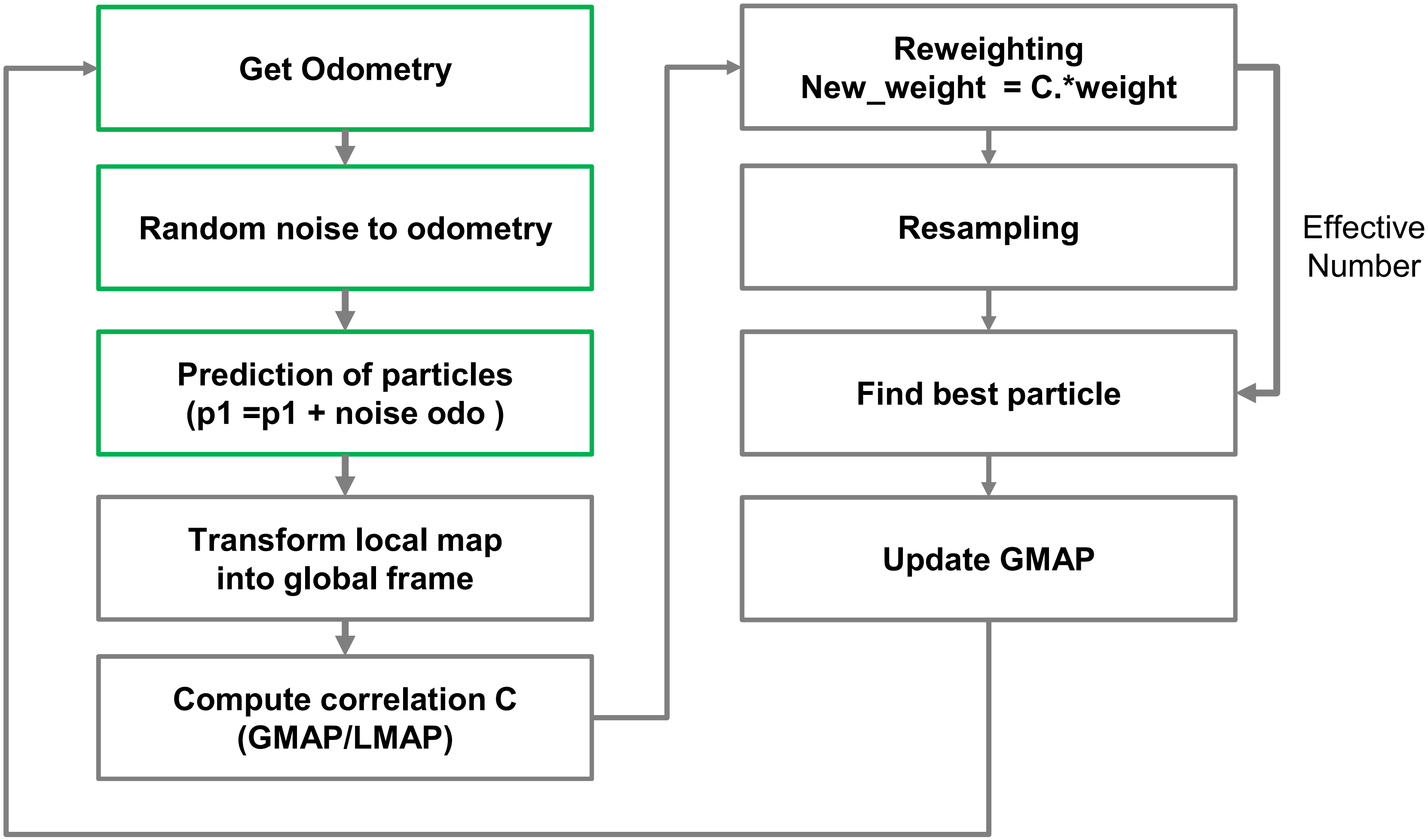
Project #4 SLAM-PF

- lidar.mat
  - t: 1.4268e+09(absolute time)
  - ~~rsz: 4324~~ (You don't need it)
  - pose: [0 0 0] (global odometry)
  - res: 0.0044 (radian, resolution) (theta = -135:0.25:135)
  - rpy: [-0.0120 -0.0164 -0.1107] (IMU roll pitch yaw)
  - scan: [1x1081 single] (Scan data, range -135deg to 135 deg)
- lidar.rpy and joint.rpy are identical l for the same time stamps

Project #4 SLAM-PF

- Odometry
  - lidar{i}.pose: [x, y, theta]
  - +x: forward from robot
  - +y: left from robot
  - +z: up from robot
  - theta: rotation around +z

PF-Based SLAM

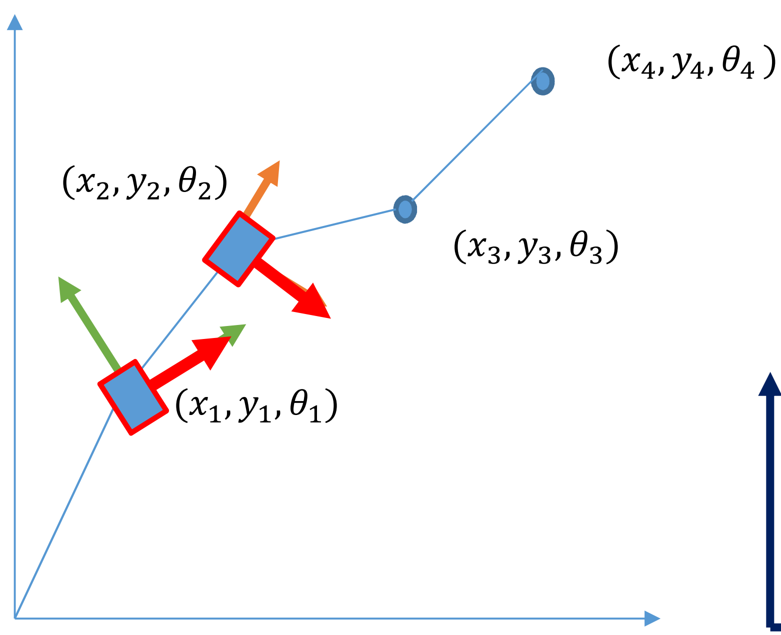


Project #4 SLAM-PF

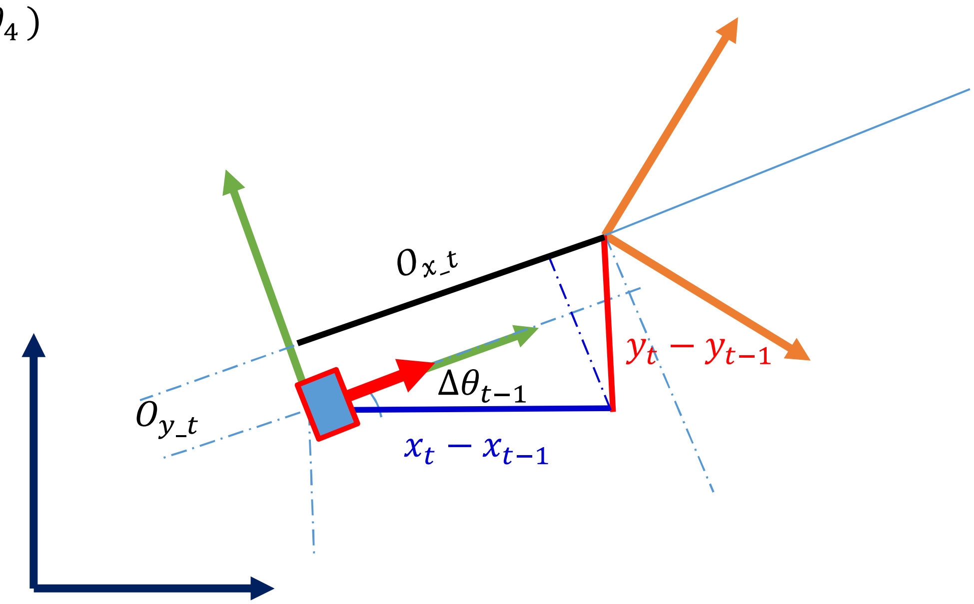
- Relative pose based on the odometry $(o_{t+1} \ominus o_t)$
  - Given global odometry
  - Find delta x, delta y and delta theta

$$\begin{bmatrix} O_{x\_t} \\ O_{y\_t} \end{bmatrix} = \begin{bmatrix} \cos\theta_{t-1} & \sin\theta_{t-1} \\ -\sin\theta_{t-1} & \cos\theta_{t-1} \end{bmatrix} \times \begin{bmatrix} x_t - x_{t-1} \\ y_t - y_{t-1} \end{bmatrix}$$

$$O_{\theta\_t} = \theta_t - \theta_{t-1}$$



Transform to local coordinate frame!!!!



Implementation

$$p_{t+1} = p_t \oplus (o_{t+1} \ominus o_t)$$

- Adding random noise
  - Random noises (mu = 0, sigma = σ) to odometry
  - For N particles, generate N random noise values

- Prediction$[P_{x_t}, P_{y_t}, P_{\theta_t}]$
  - $\begin{bmatrix} P_{x\_t} \\ P_{y\_t} \end{bmatrix} = \begin{bmatrix} P_{x_{t-1}} \\ P_{y_{t-1}} \end{bmatrix} + \begin{bmatrix} \cos P_{\theta_{t-1}} & -\sin P_{\theta_{t-1}} \\ \sin P_{\theta_{t-1}} & \cos P_{\theta_{t-1}} \end{bmatrix} \times \begin{bmatrix} O_{x\_t} \\ O_{y\_t} \end{bmatrix}$
  - $P_{\theta\_t} = P_{\theta_t - 1} + O_{\theta\_t}$
  - Note : $P_{\theta_{t-1}}$ is the heading of a particle

- Example
  - for i =1:N(=100)

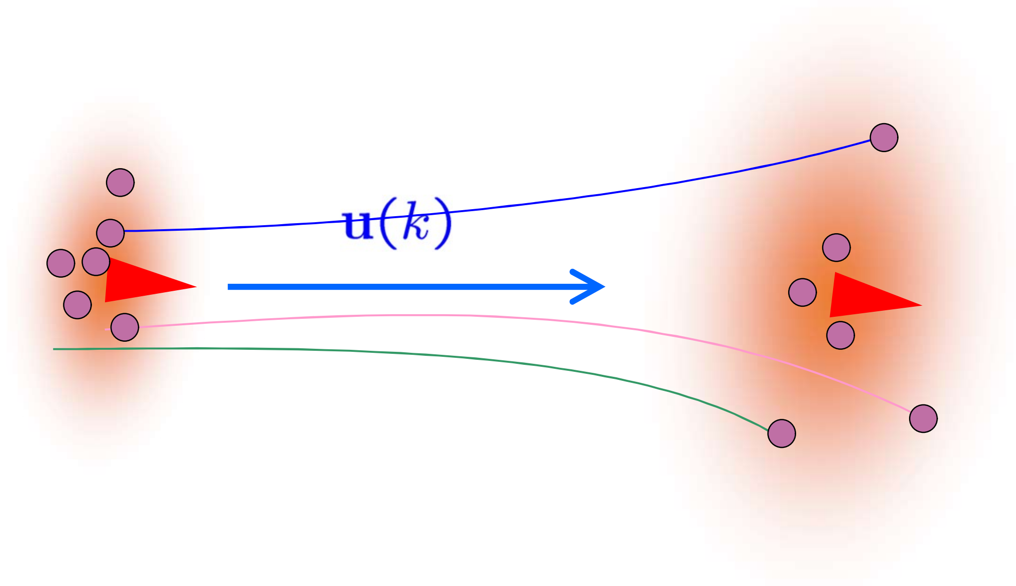
```
noise_odo(i) = odo + normrnd(0, sigma)
pf(jj,:) = [pf(i,1:2) + (R* noise_odo(i,1:2)')', pf(i,3)+ noise_odo(i, 3)]
```

- TIP: Sigma = sigma * norm(odo)

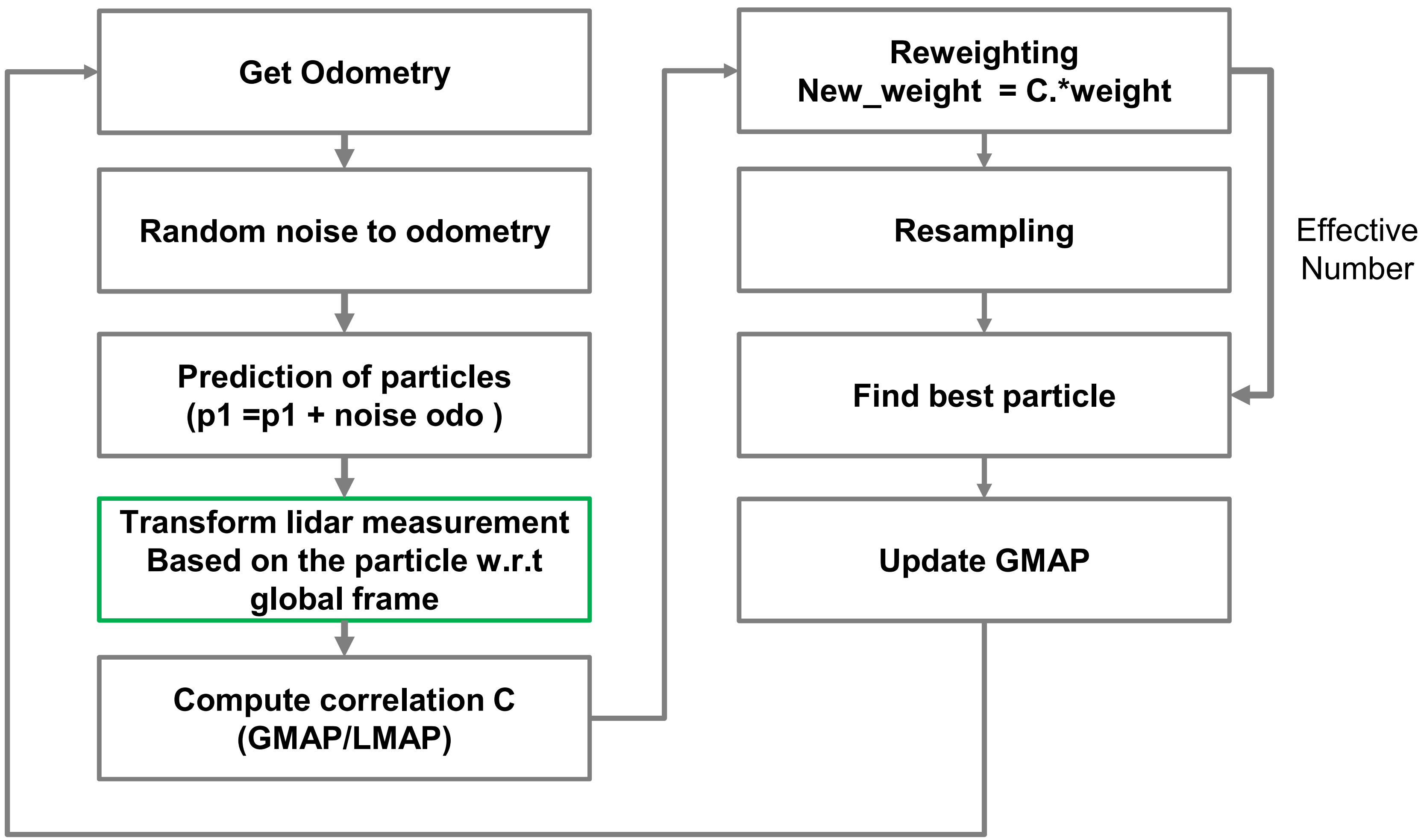
(when the robot doesn't move, norm(odo = 0))

Implementation

- Motion noise
  - Gaussian Random Noise (mu = 0, sigma = σ)

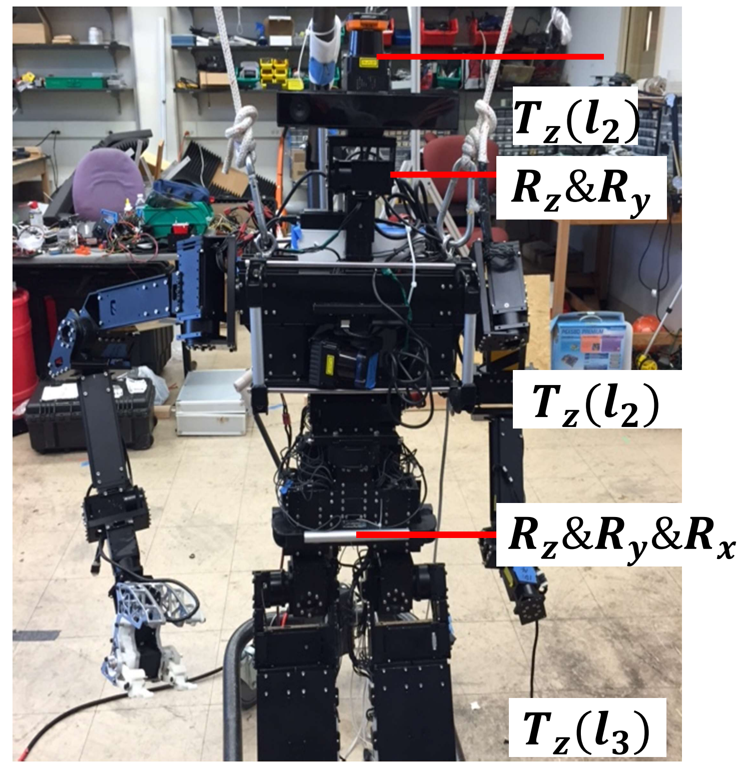


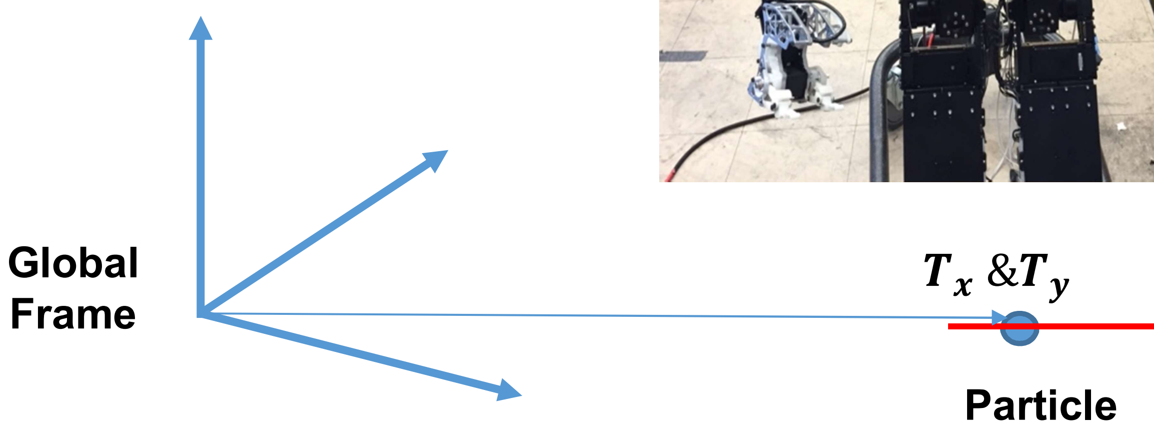
PF-Based SLAM



Transformation

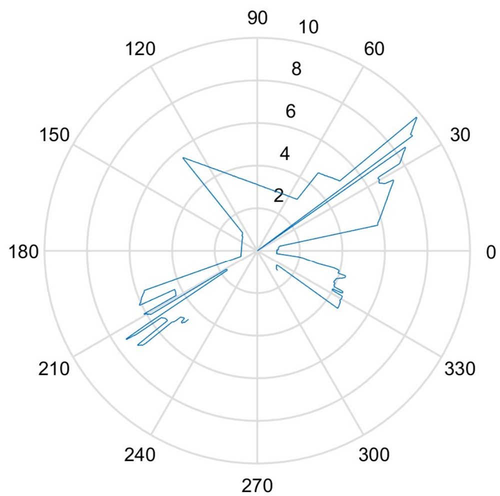
- Input
  - $\boldsymbol{pf(x, y, \theta)}$
  - Body roll and pitch: $\boldsymbol{r, p}$
  - Head yaw and pitch





$$T = T_{xyz}(pf(x), pf(y), l_3)R_z(pf(\theta))R_y(p)R_x(r)T_z(l_2)R_z(head\_yaw)R_y(head\_pitch)T_z(l_1)$$

Transformation

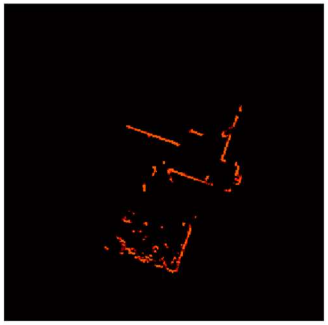


$$\text{scan} = \begin{bmatrix} x_1, y_1, z_1; \\ \cdots \\ x_{1081}, y_{1081}, z_{1081} \end{bmatrix}$$

Polar Coordinate

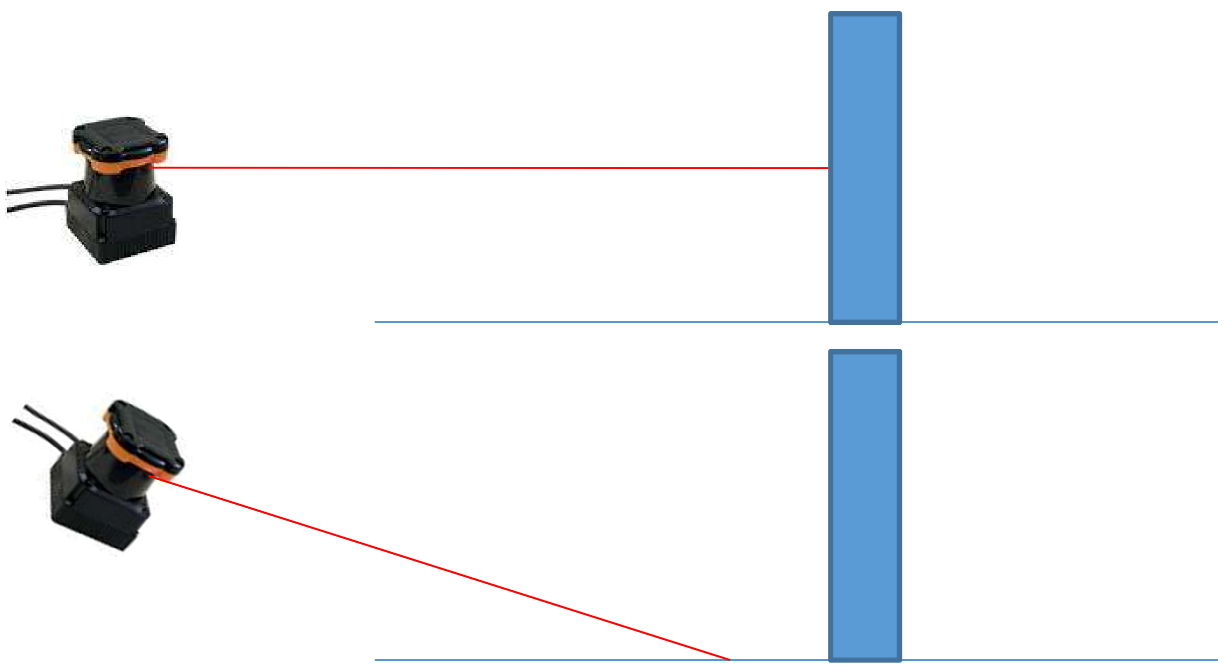
Cartesian Coordinate w. r. t lidar frame

$$\text{scan\_world} = \mathrm{T} \times scan$$

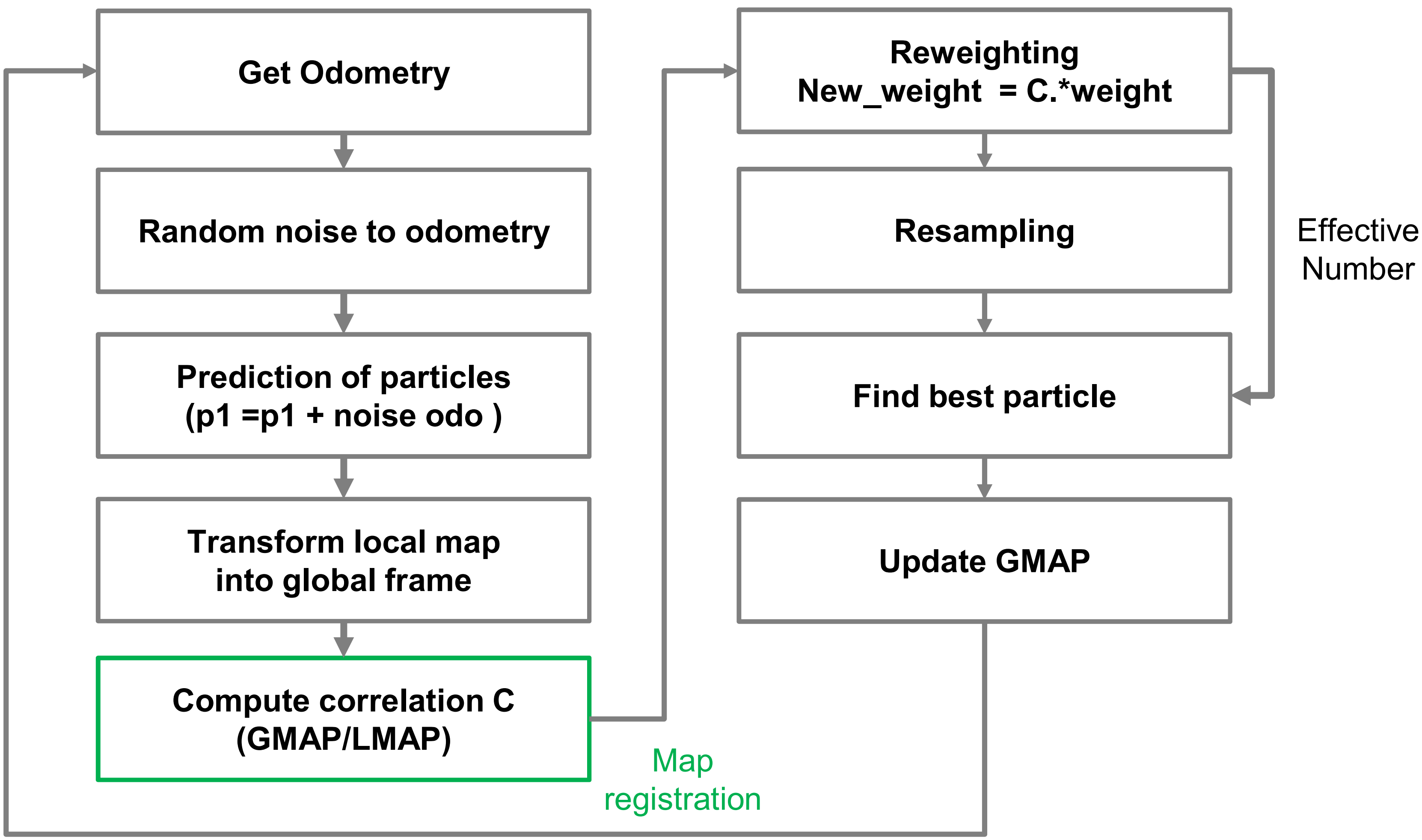


Implementation

- Lidar Data
  - Remove the hits on the ground
  - Compensate the roll and yaw of head pose
  - Possible to compensate the motion of the robot

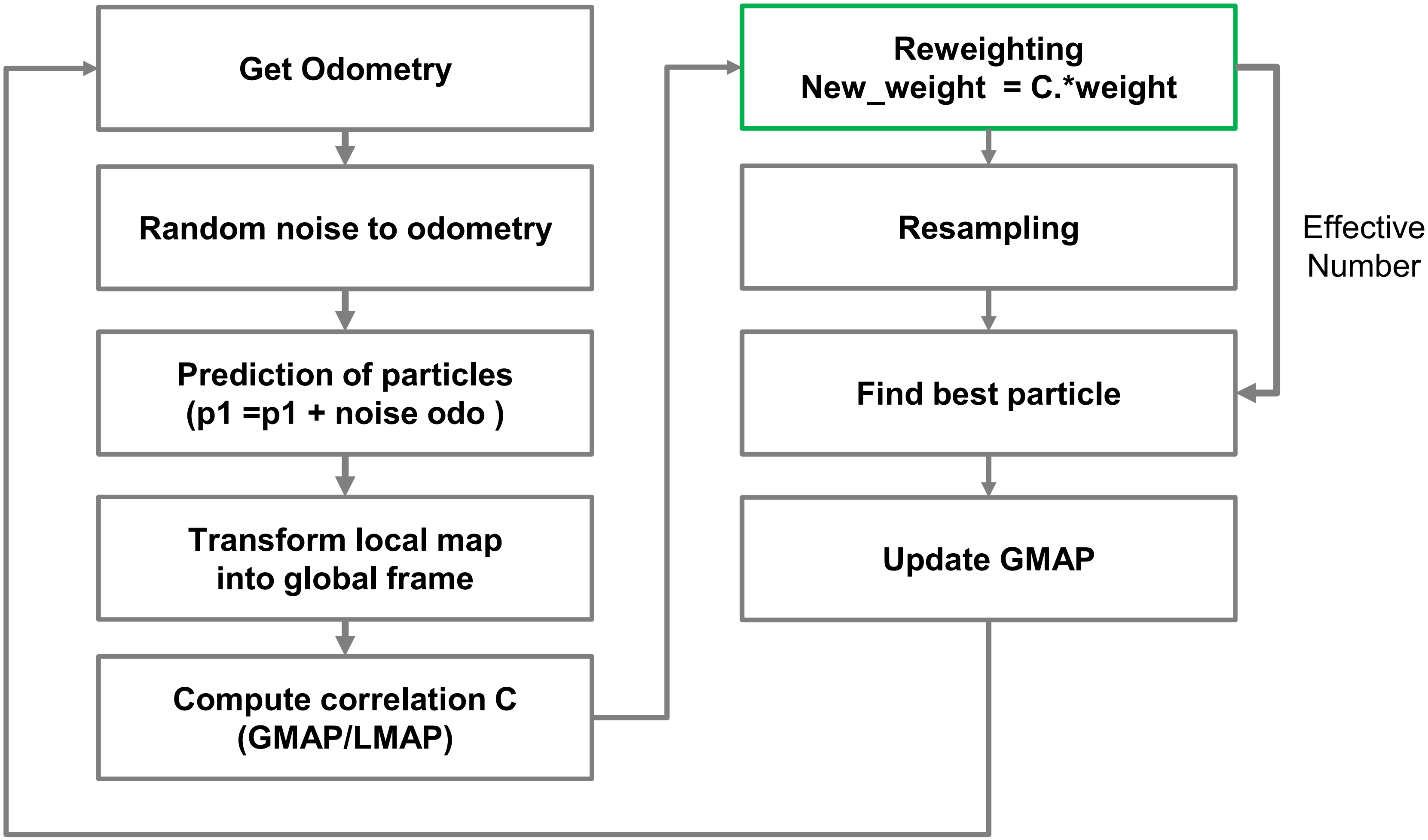


PF-Based SLAM



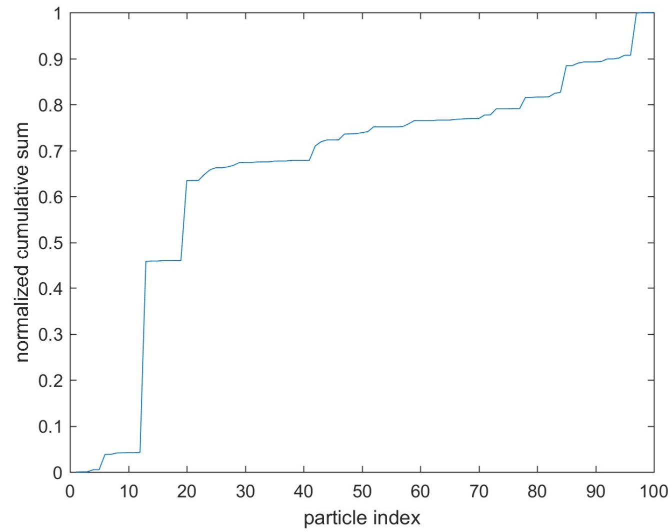
PF-Based SLAM

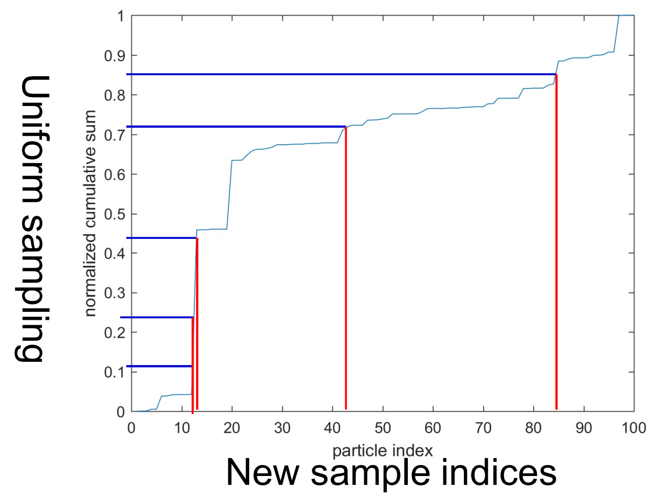
```
pfWt = pfWt.* C;
pfWt = pfWt / sum(pfWt);
```



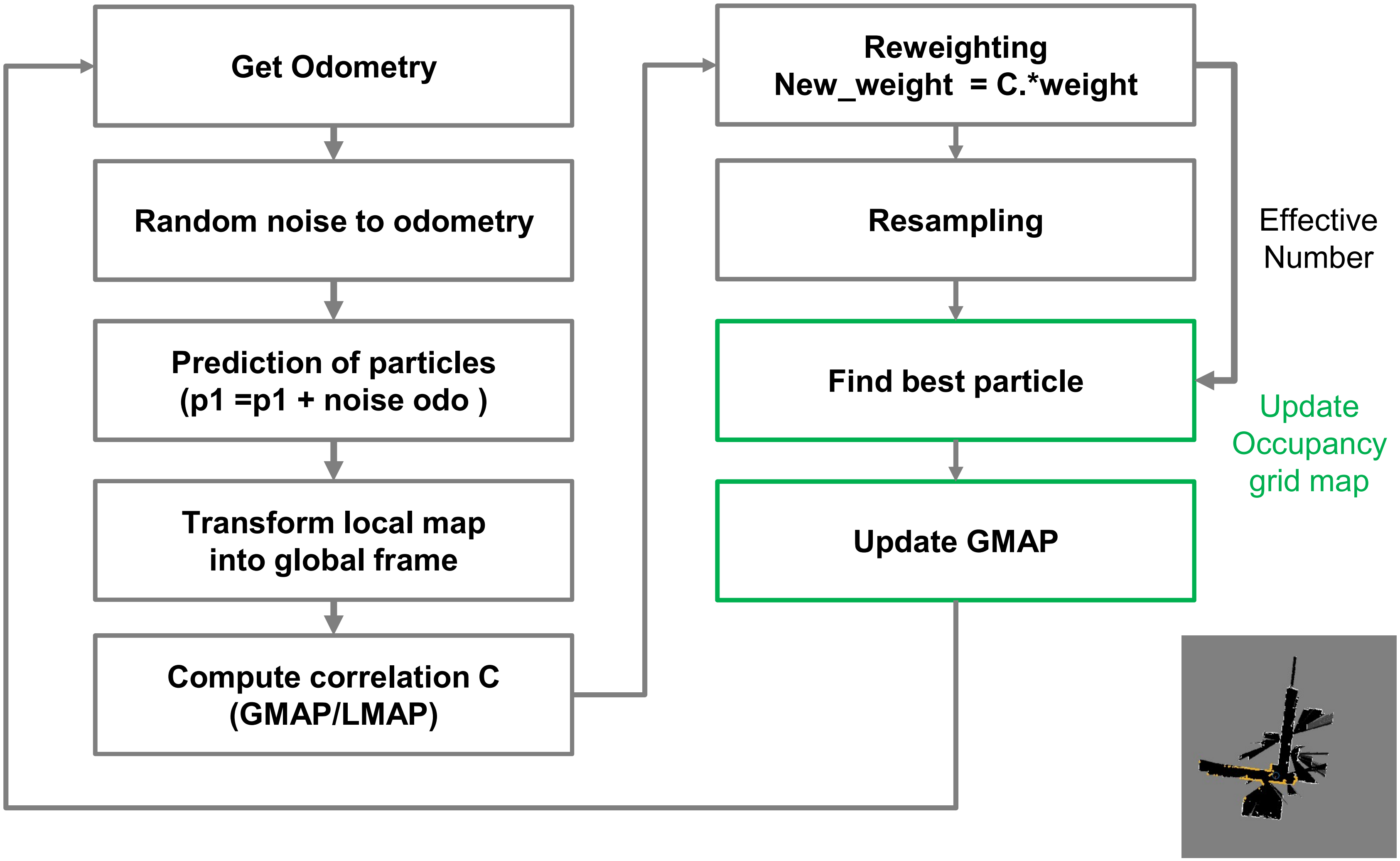
Resampling

- `wsum = cumsum(w(:)');`
- `wsum = wsum./wsum(end);`
- `r = rand(1, n);`
- `[~, isort] = sort([wsum r]);`
- `s = [ones(1,nw) zeros(1,n)];`
- `s = s(isort);`
- `ssum = cumsum(s);`
- `index = ssum(s == 0) + 1;`



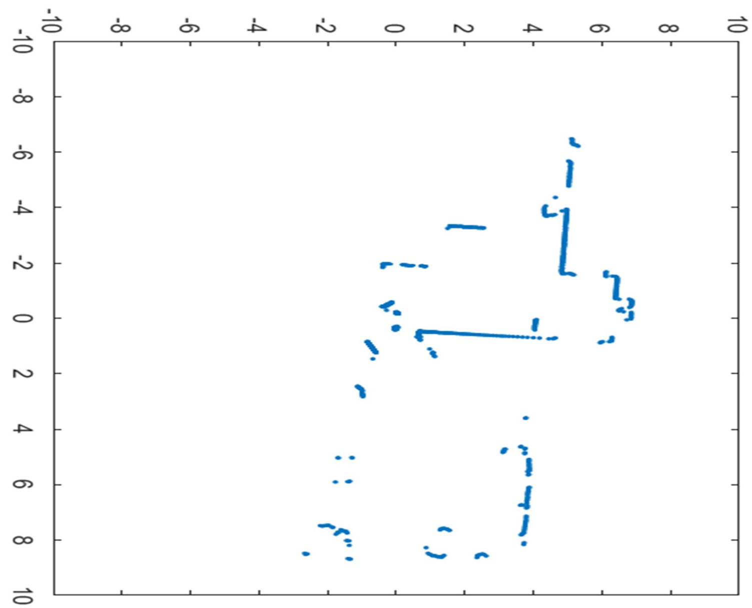


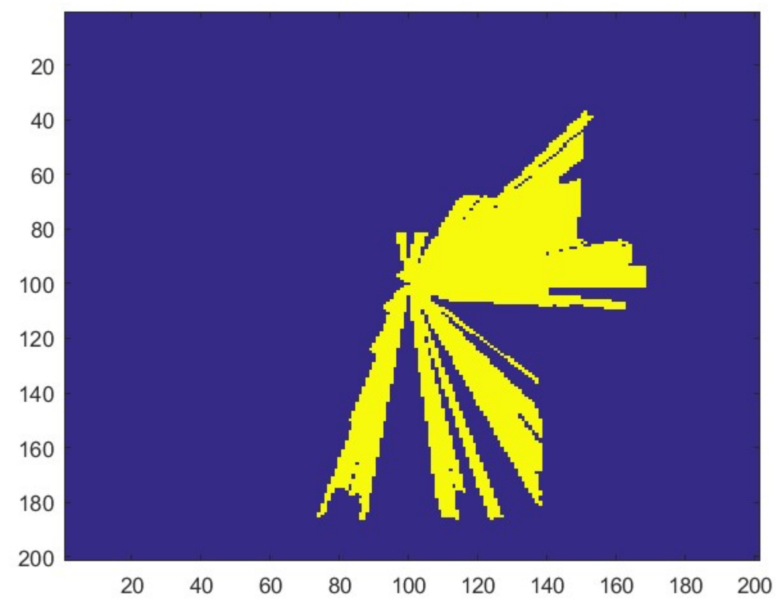
PF-Based SLAM



Occupancy grid map

- getMapCellsFromRay
  - [x_between, y_between] = getMapCellsFromRay(xori, yori, xis(i), yis(i));
  - returns empty indices in the global map between robot position and scan point
  - xori, yori : pose of robot in the global map (indices)
  - xis, yis : scan points in the global map (indices)





Tips

- Skip static part at the beginning part
- MapUtils2 with bresenham2D.py for getMapCellsFromRay
- Fast Debugging
  - Start with a small number of particles
  - Small size of grid map for debugging
- Interval between scan data : 10
  - First, start with large interval value
- Check Transformation!
  - From lidar to global
  - Odometry

https://piazza.com/class/jc7rz6mr2ol1ys?cid=260

Tips: Parameters (baseline)

- The number of particles : 50 ~ 150
- Map resolution : 0.05
- Noise (when interval is 10)
  - Normal random([0.01, 0.01, 0.5*pi/180])
  - If norm(odo) < ε, noise is also zero
- Log odds Parameter
  - logOddOcc = 3, logOddFree = -1
  - Maximum : 120
  - Minimum :-120

Tips: Parameters (baseline)

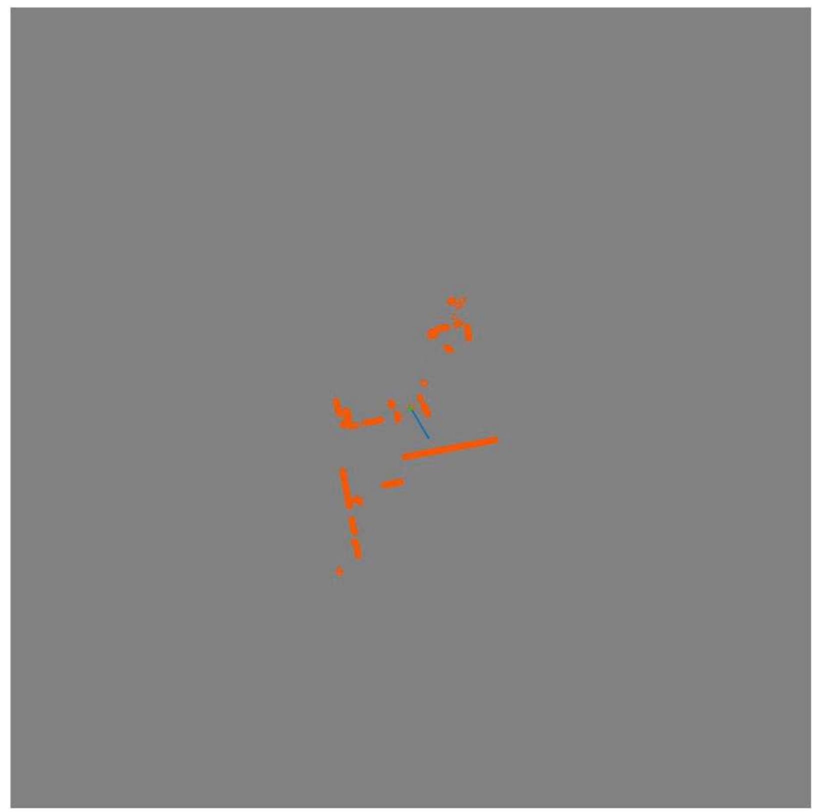
- Resampling (100 particles)
  - Effective number (1/sum(weights*weights)) < 20

Project #4 SLAM-PF

- Train#0

Mapping based on odometry

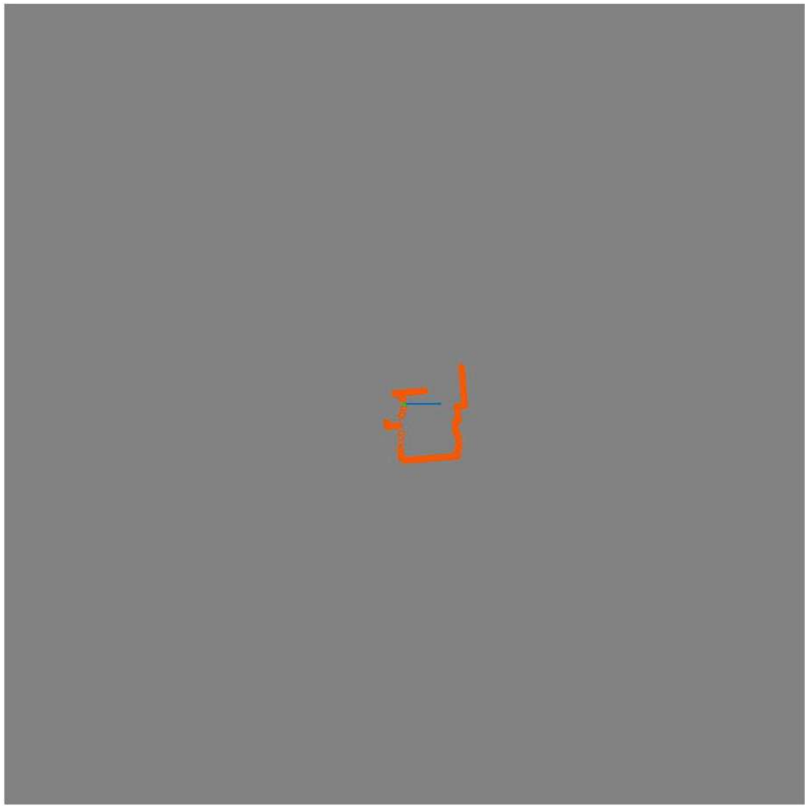
Particle-SLAM



Project #4 SLAM-PF

- Train#3

Mapping based on odometry



Particle-SLAM