F1TENTH Boot Camp

Lecture 3: Mapping and Localization

Mechanical Engineering

Cheolhyeon Kwon

2024.09.27

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

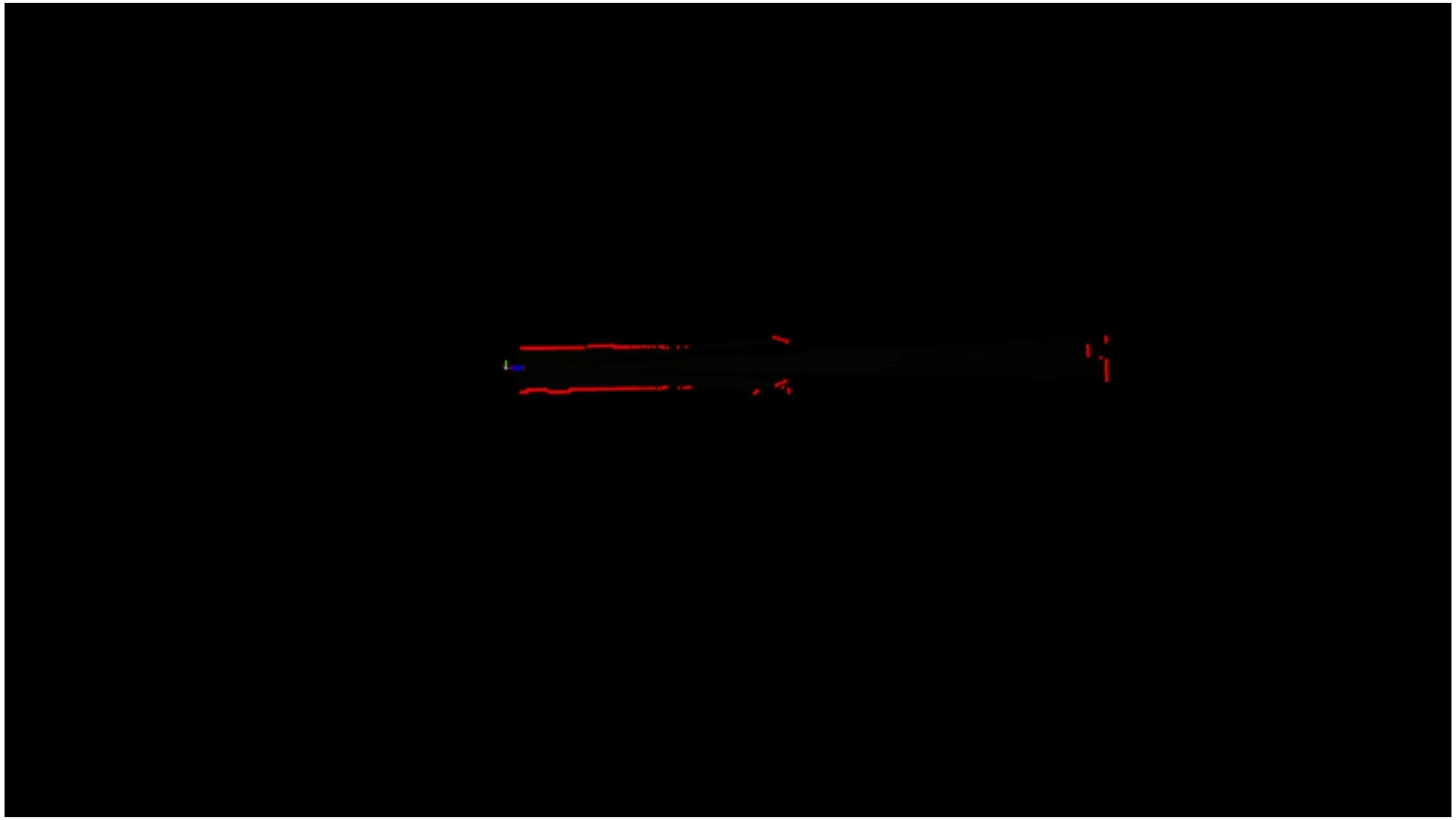
Scan Matching

Autonomous Vehicle Localization Problem

- Determine the state(position and orientation) of a vehicle with respect to the environment
- Localization can be within a global frame of a map, or relative to one's starting point
- Key component of all methods & techniques covered in this course!

Localization within a given Map

Localization and Mapping (SLAM)



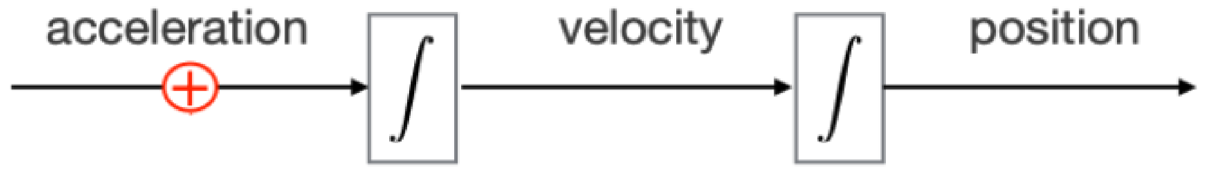
First attempt: Motion Integration

- Odometry: Start at known pose and integrate control and motion measurements to estimate the current pose
  - Integrate dynamics using information from VESC, wheel encoders, IMU, etc

(credit : Luca Carlone, MIT 6.141/16.405)

First attempt: Motion Integration

- Odometry: Start at known pose and integrate control and motion measurements to estimate the current pose
  - Integrate dynamics using information from VESC, wheel encoders, IMU, etc.

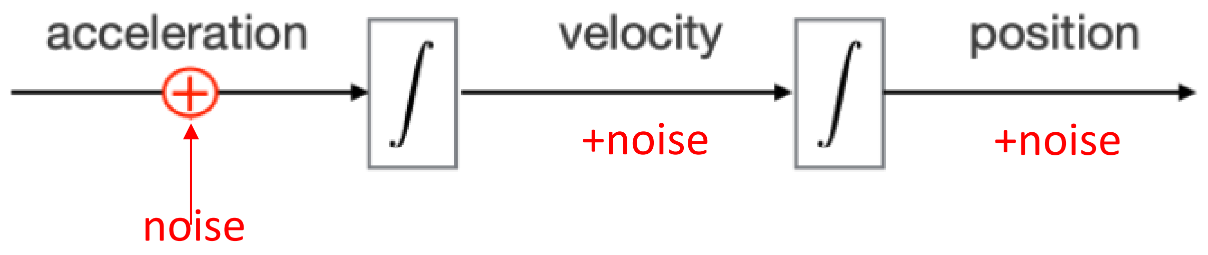


$$\begin{tikzcd} \arrow[r, "\text{acceleration}"] & \oplus \arrow[r] & \boxed{\int} \arrow[r, "\text{velocity}"] & \boxed{\int} \arrow[r, "\text{position}"] & {} \end{tikzcd}$$

(credit : Luca Carlone, MIT 6.141/16.405)

First attempt: Motion Integration

- Odometry: Start at known pose and integrate control and motion measurements to estimate the current pose
    - Integrate dynamics using information from VESC, wheel encoders, IMU, etc



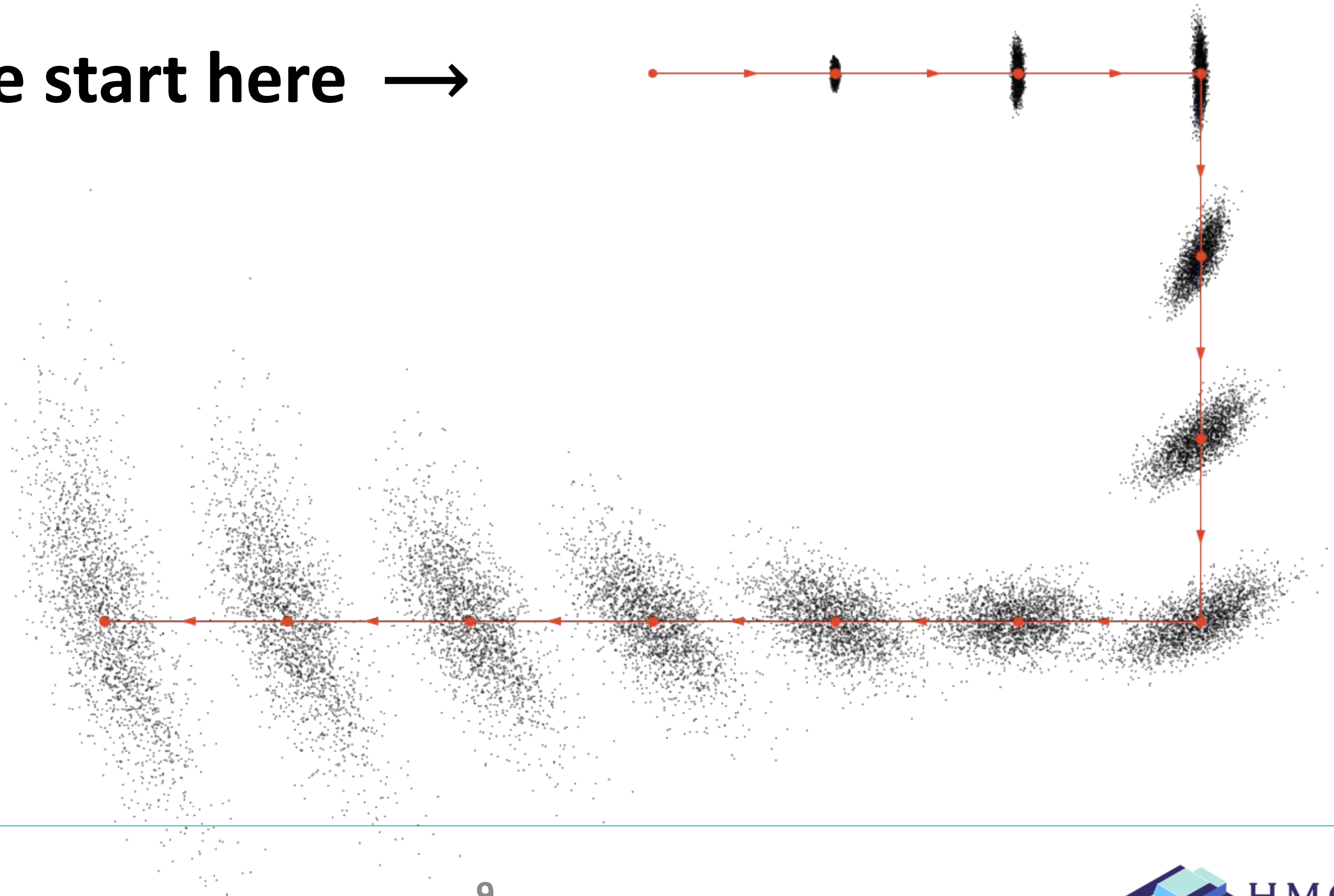
- Odometry = open loop estimation → error increases over time

(credit : Luca Carlone, MIT 6.141/16.405)

First attempt: Motion Integration

- Odometry = open loop estimation → error increases over time

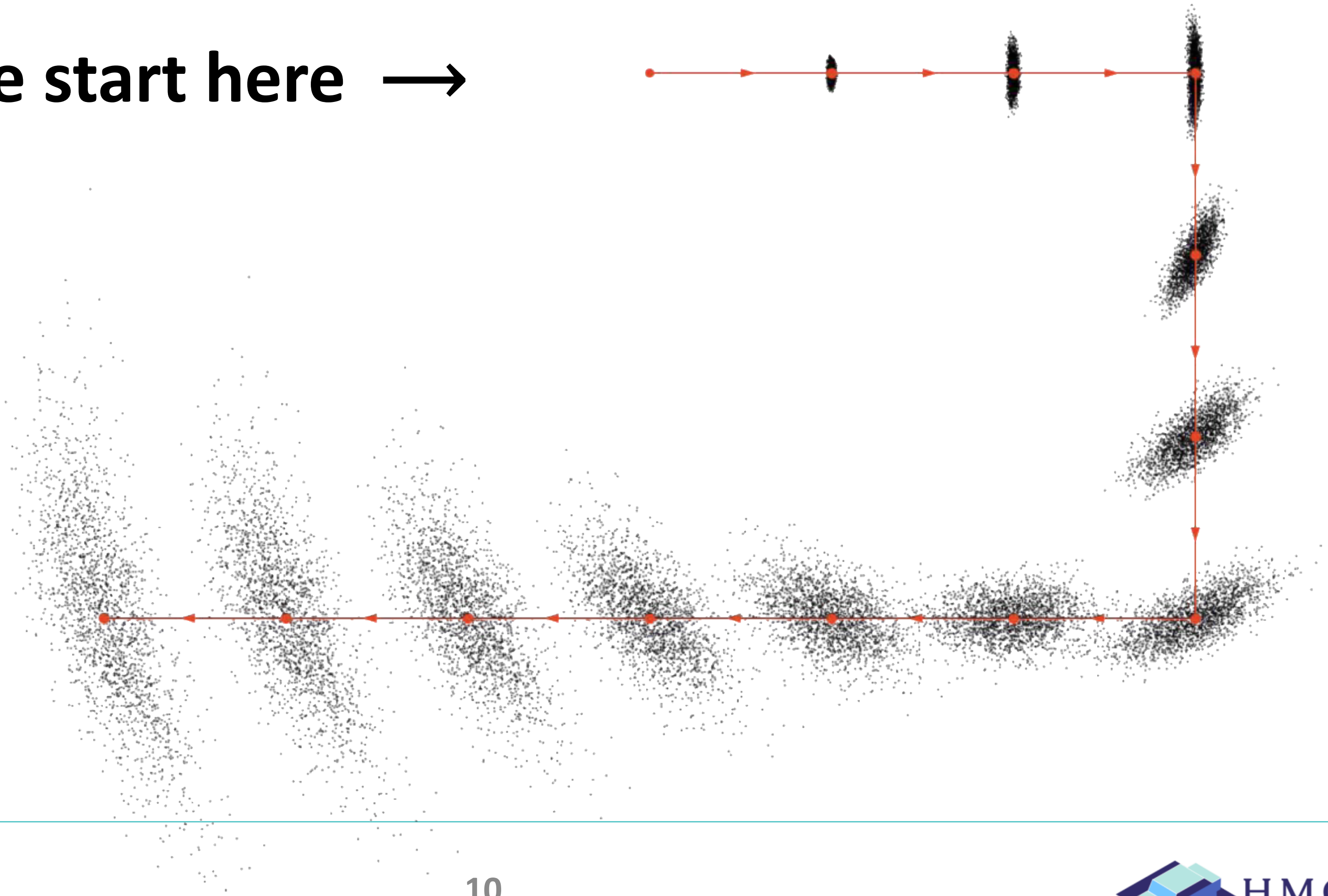
Let's say we start here ⟶



First attempt: Motion Integration

- Odometry = open loop estimation → error increases over time

Let's say we start here ⟶



Notice how
Uncertainty
Increases ⟶

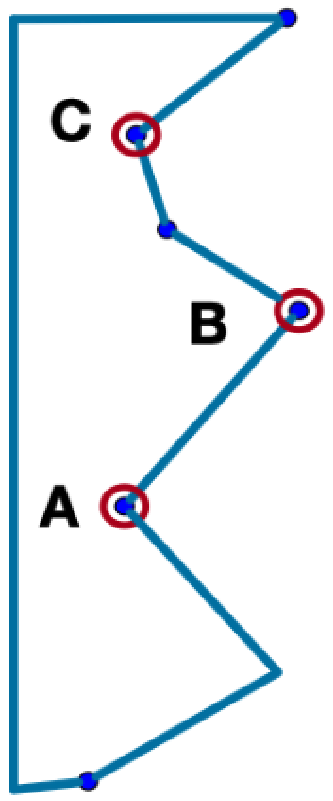
Beyond Odometry: Scan Matching

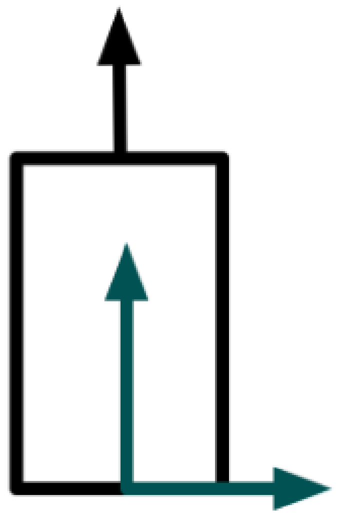
- Using odometry for localization is known as “dead reckoning” in aircraft and maritime navigation
- How do we utilize that information, such as range sensor measurements to localize the vehicle?

- One approach to localization using sets of range measurements collected across time – E.g. Lidar scans
- Note: Odometry information is not used, but the output of scan matching can be combined with odometry info to improve localization

Problem Setup

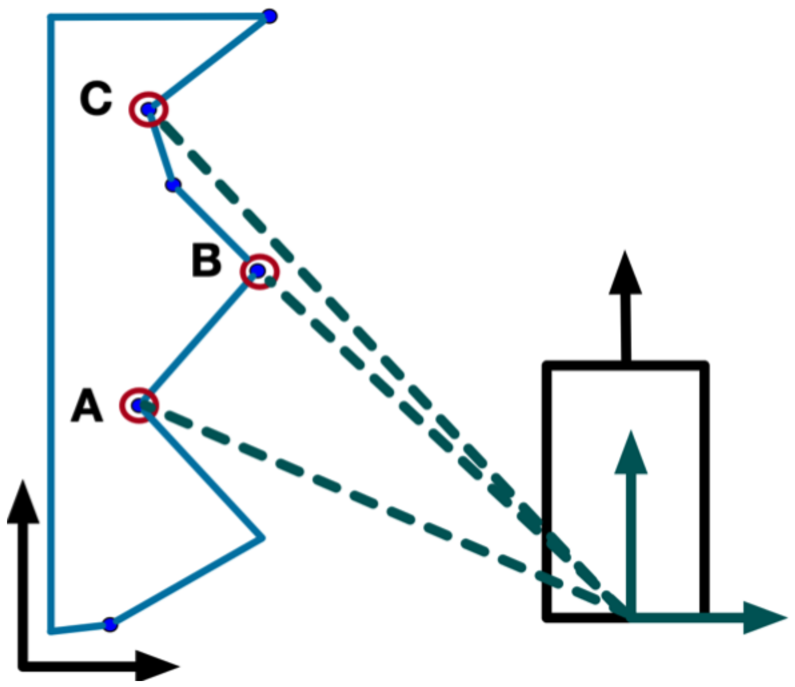
- Let's say my vehicle is in some environment with landmarks A, B, C



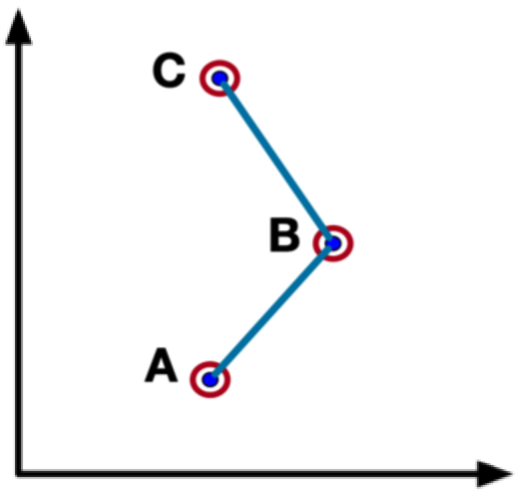


Problem Setup

- At t=0, measurements of distances to A, B, C are taken:

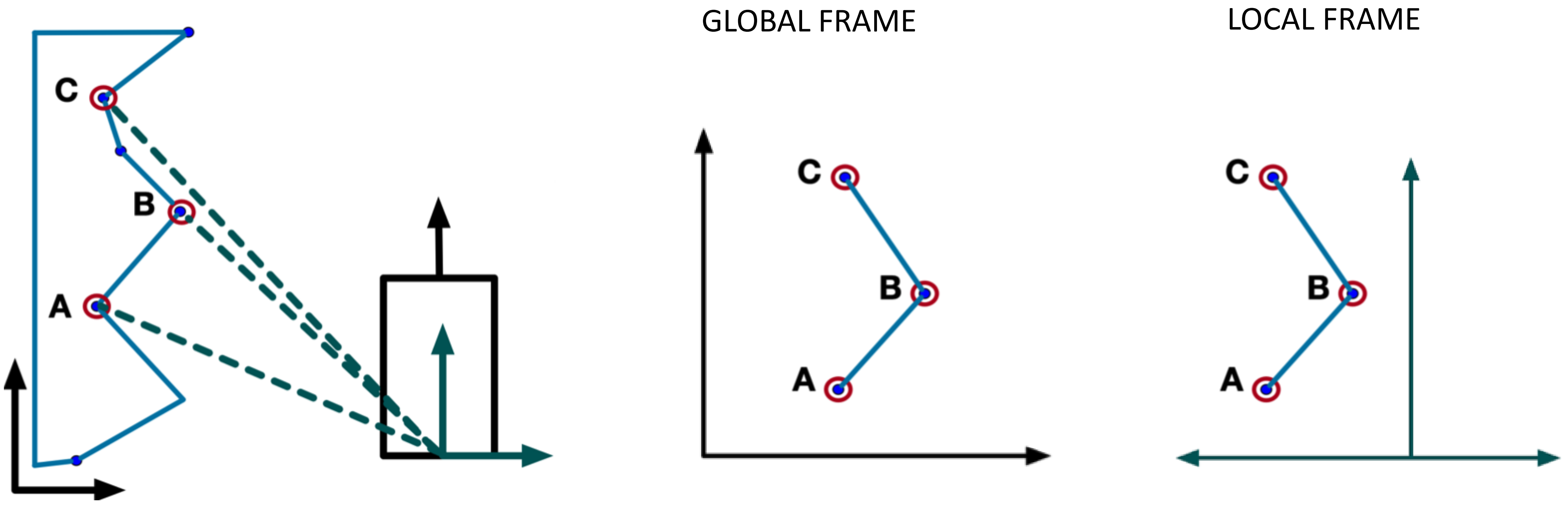


GLOBAL FRAME



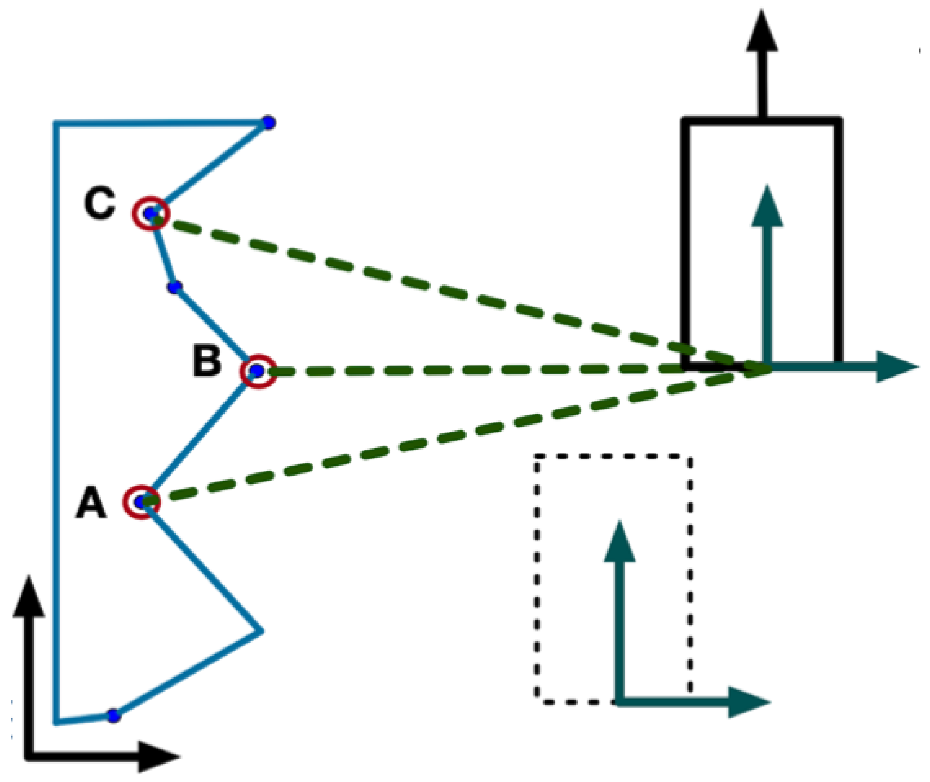
Problem Setup

- At t=0, measurements of distances to A, B, C are taken:

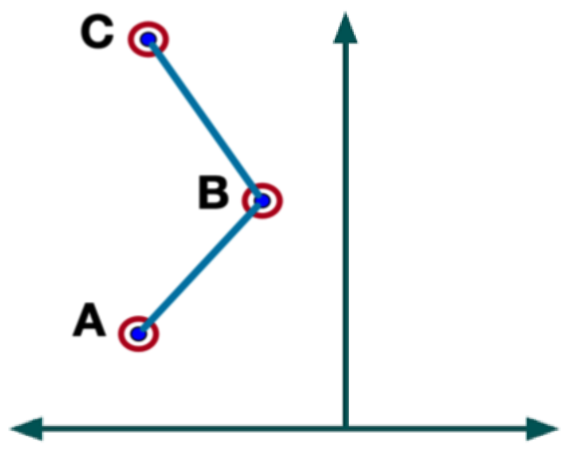


Problem Setup

- At t=1, moving in some unknown direction, the distances to the landmarks will change:

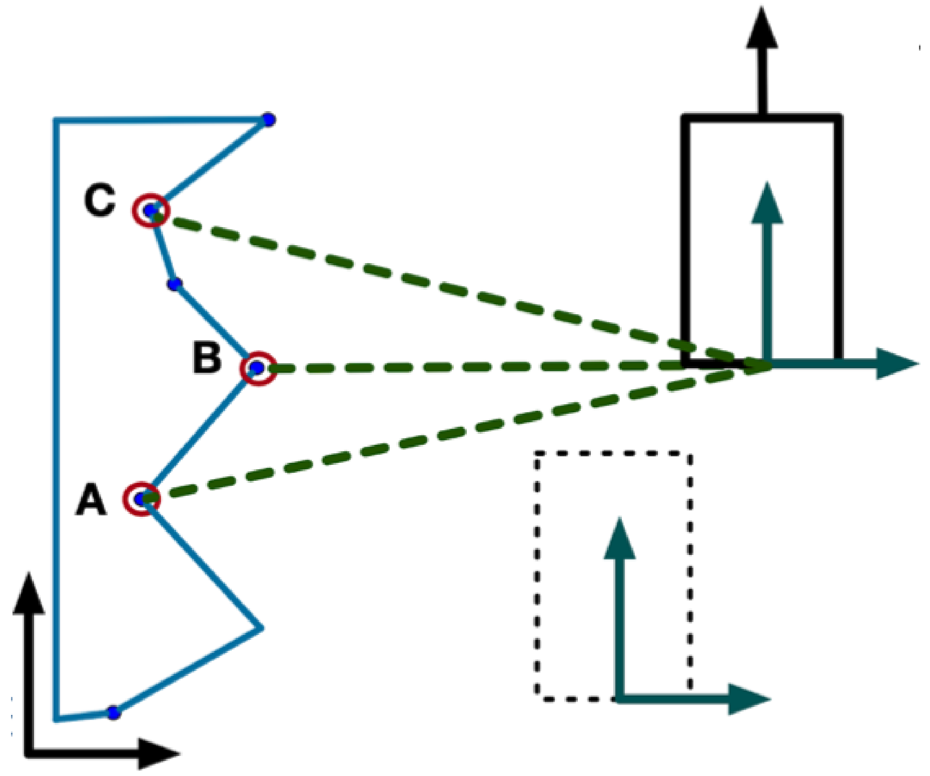


<t=0>



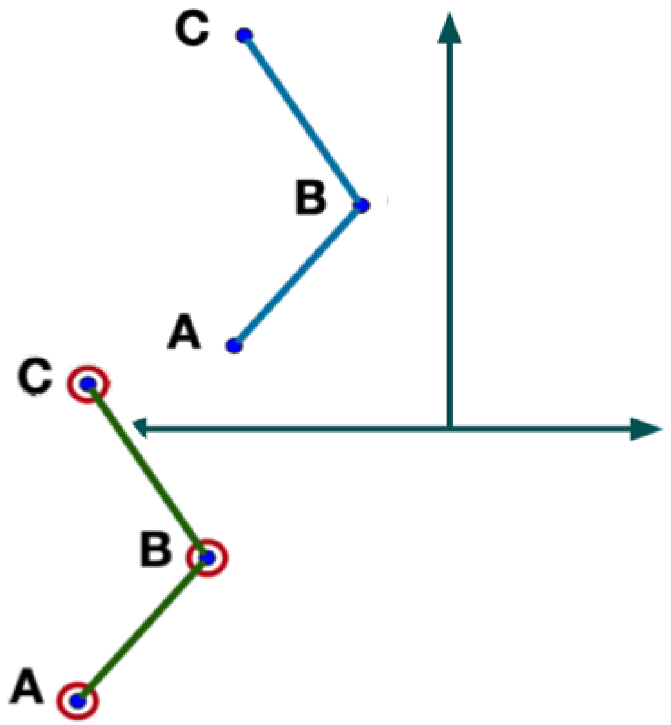
Problem Setup

- We want to find transform R, that transforms the two set of points to be the closest:



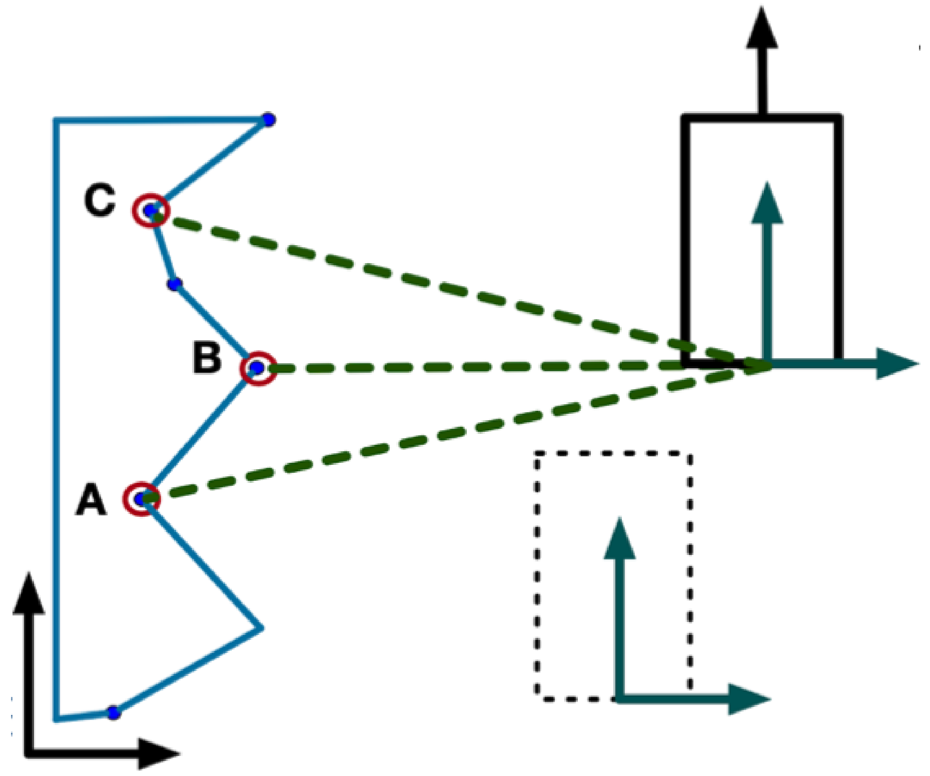
<t=0>

<t=1>



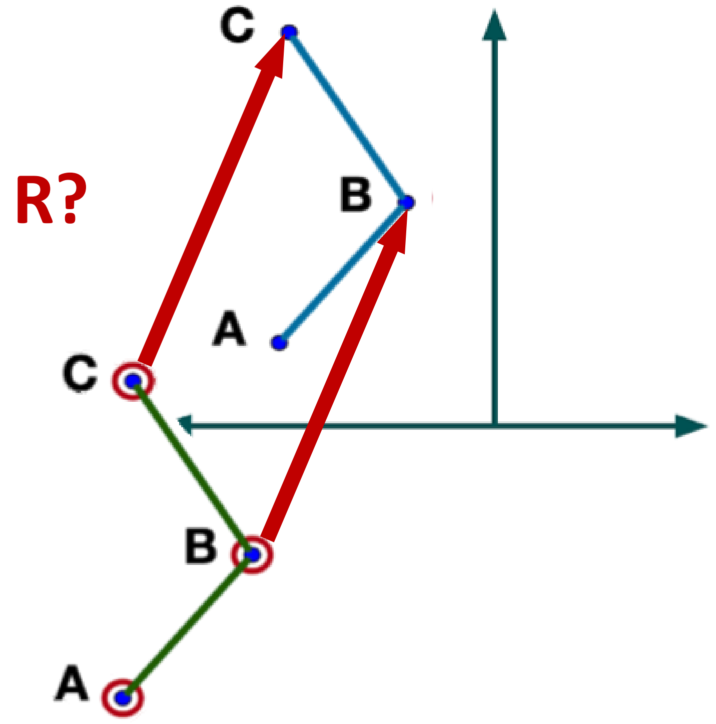
Problem Setup

- We want to find transform R, that transforms the two set of points to be the closest:



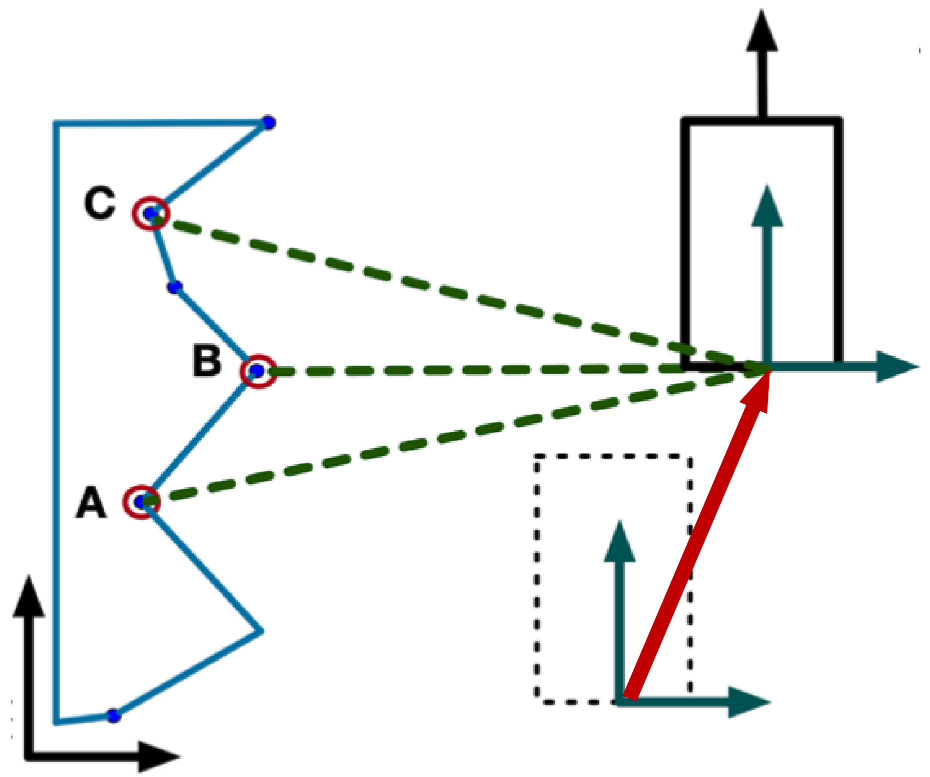
<t=0>

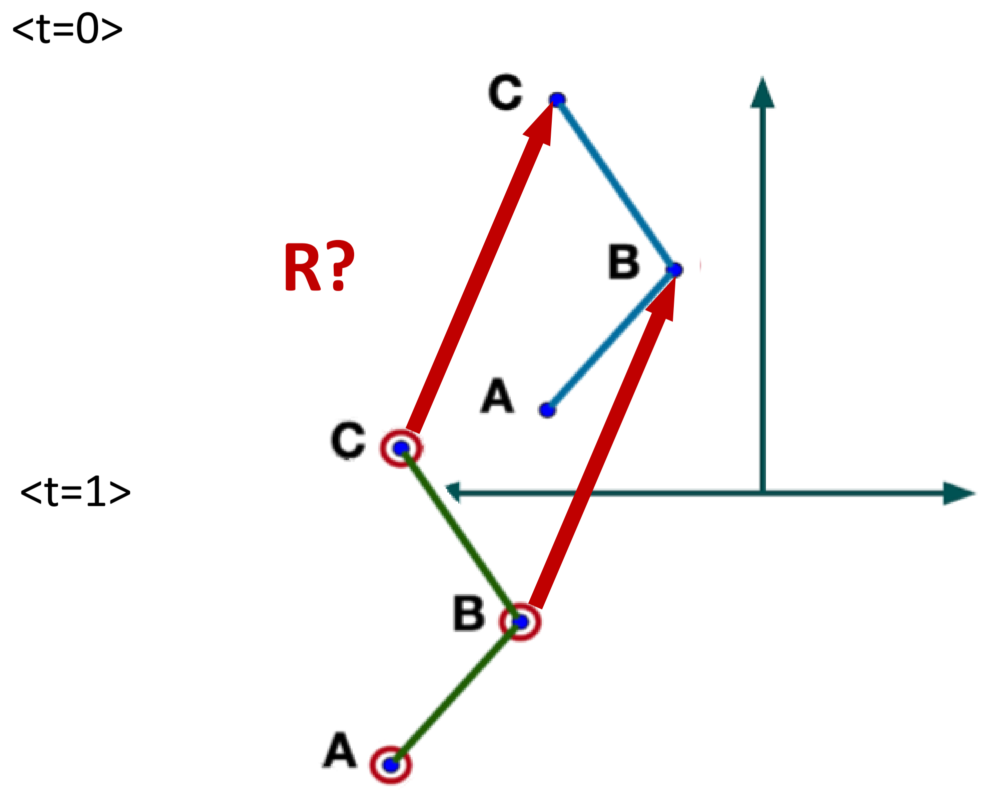
<t=1>



Problem Setup

- We want to find transform R, that transforms the two set of points to be the closest:

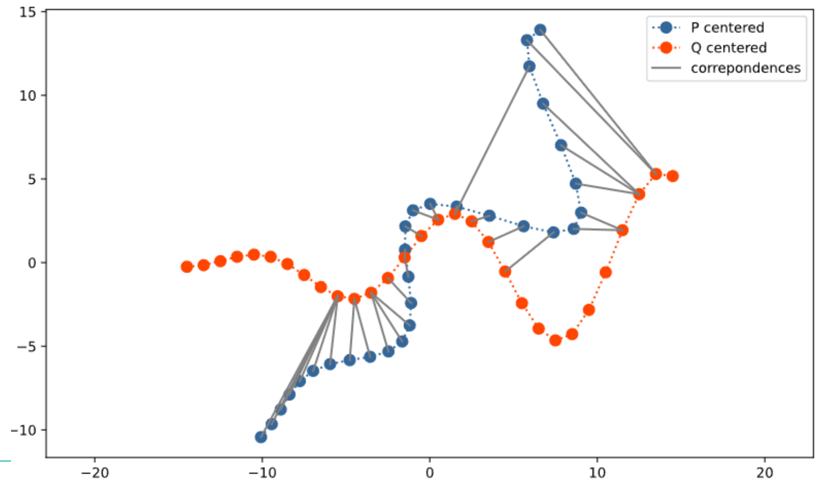




How do we find R?

- Challenge :
  - If we knew A, B, C (landmarks) exactly, then this would be easy
  - We don’t know which measurement is for which landmark

- Solution :
  - Assume closest points correspond to each other ⟶ “correspondence match”
  - Iteratively find the best transform R



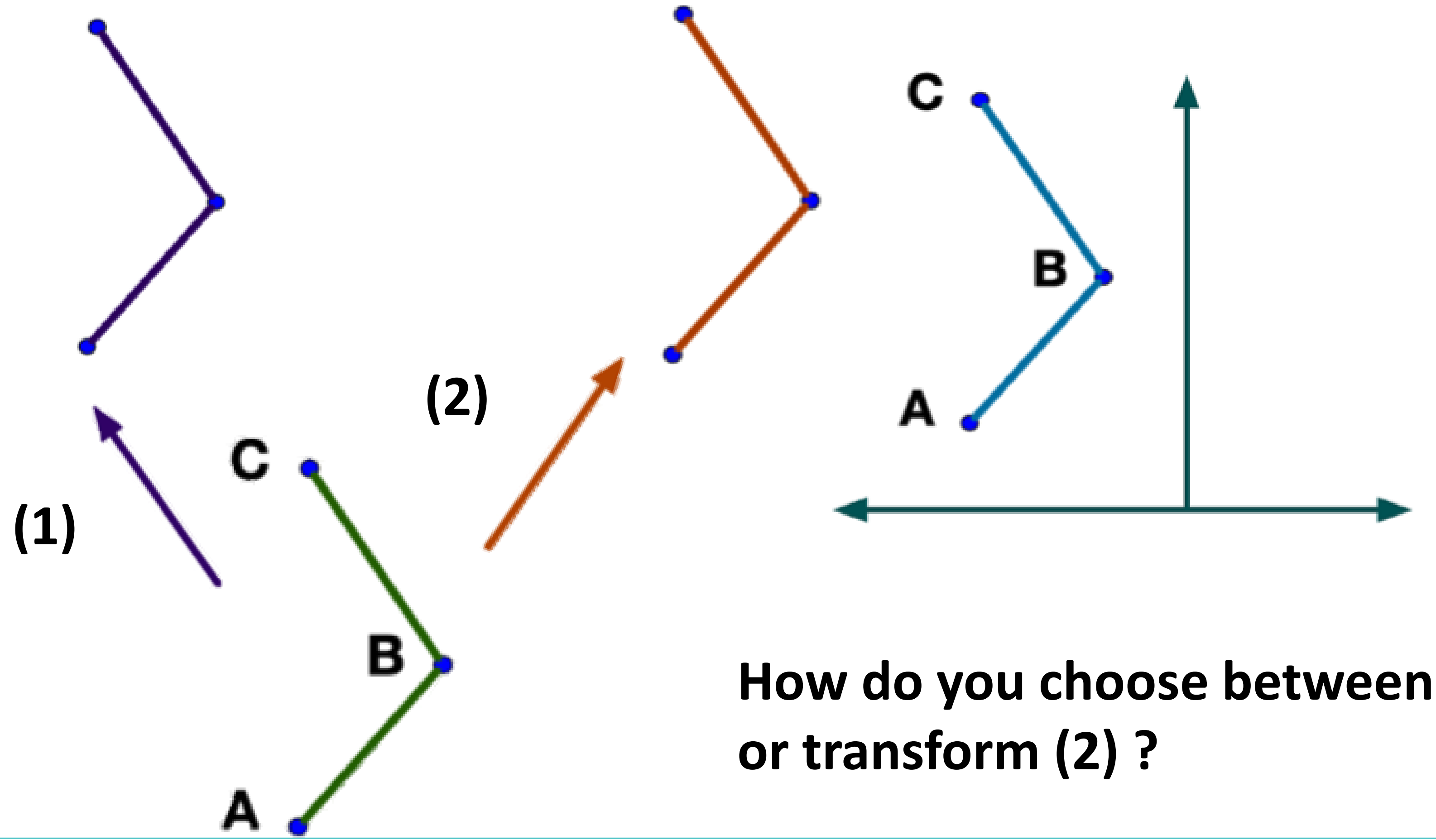
Iterative search for best transform

1. Make some initial "guess" of R (current guess)
2. For each point in new scan (t=k+1), find closest point in previous set (t=k) (**correspondence search**)
3. Make another "better guess" of R (next guess)
4. Set next guess to current guess, repeats step 2-4 until converges

What makes a "better guess"? ⟶ Need a metric

How do you choose between (1) or (2)?

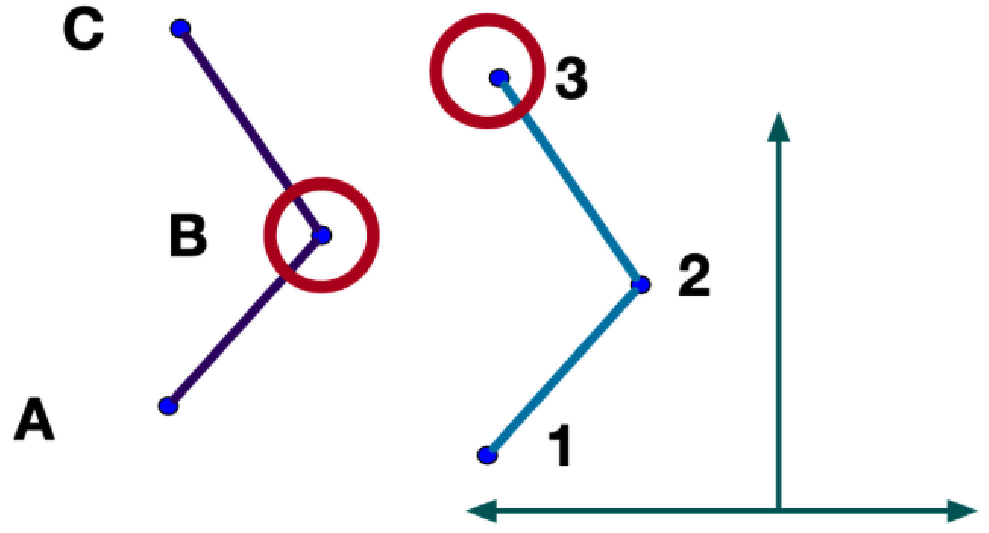
- Assume correspondences have been found



How do you choose between transform (1) or transform (2) ?

Better Candidate Match Function

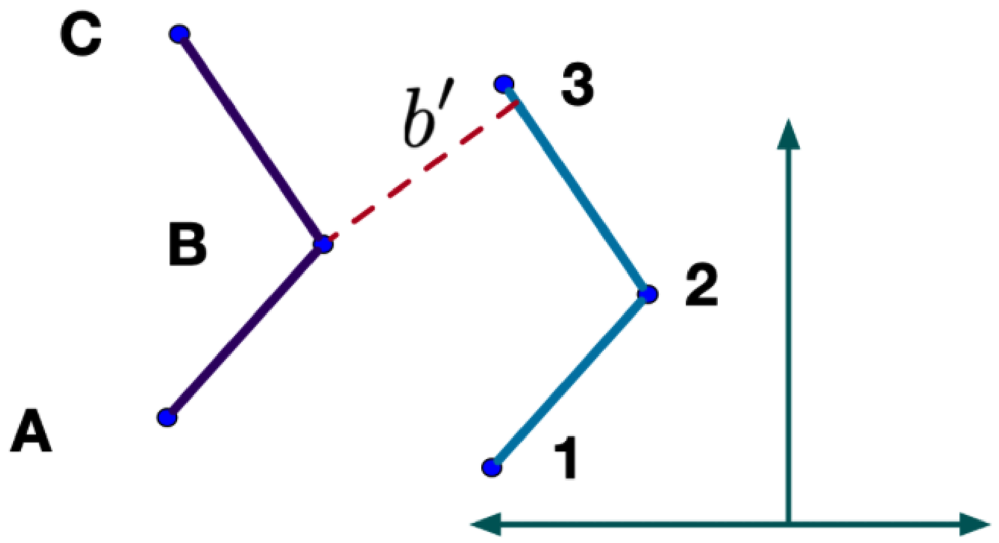
- Point to projected point (point-to-point) metric:



Closest point to B is 3

Better Candidate Match Function

- Point to projected point (point-to-point) metric:

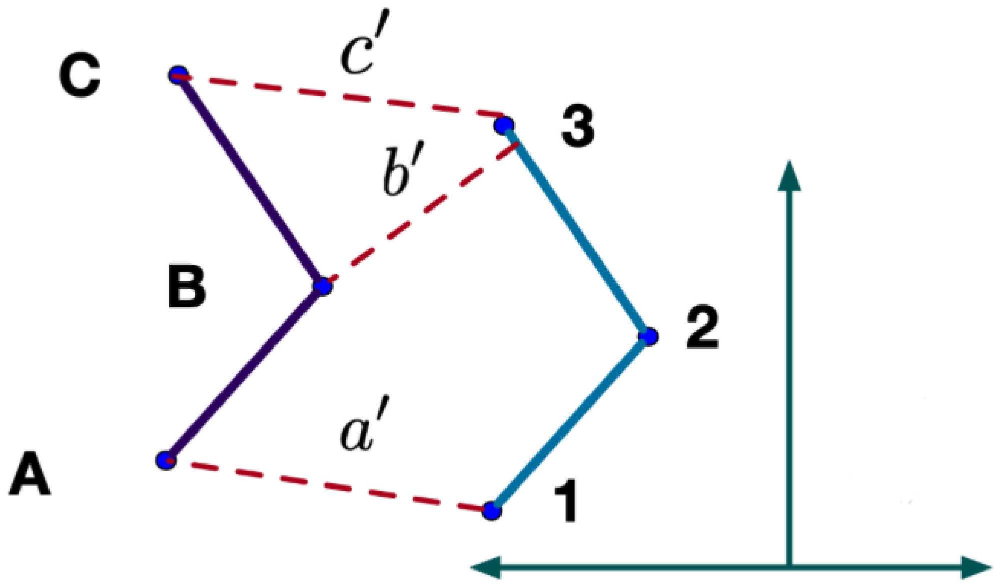


Closest point to B is 3

Find projected **point** on line segment 2-3, b'

Better Candidate Match Function

- Point to projected point (point-to-point) metric:



Closest point to B is 3

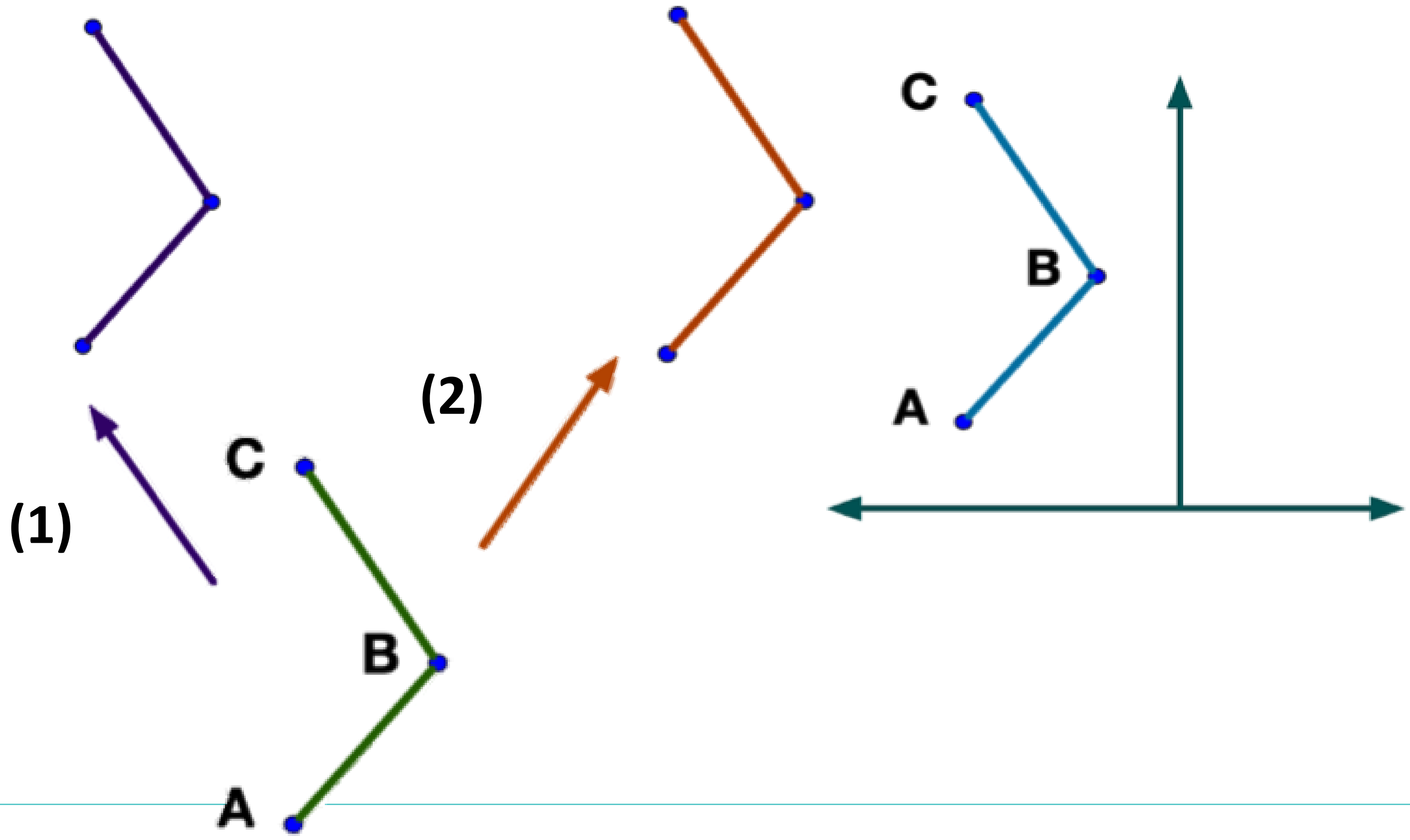
Find projected **point** on line segment 2-3, b'

Square the sum of projected distances

$$Score = |a'|^2 + |b'|^2 + |c'|^2$$

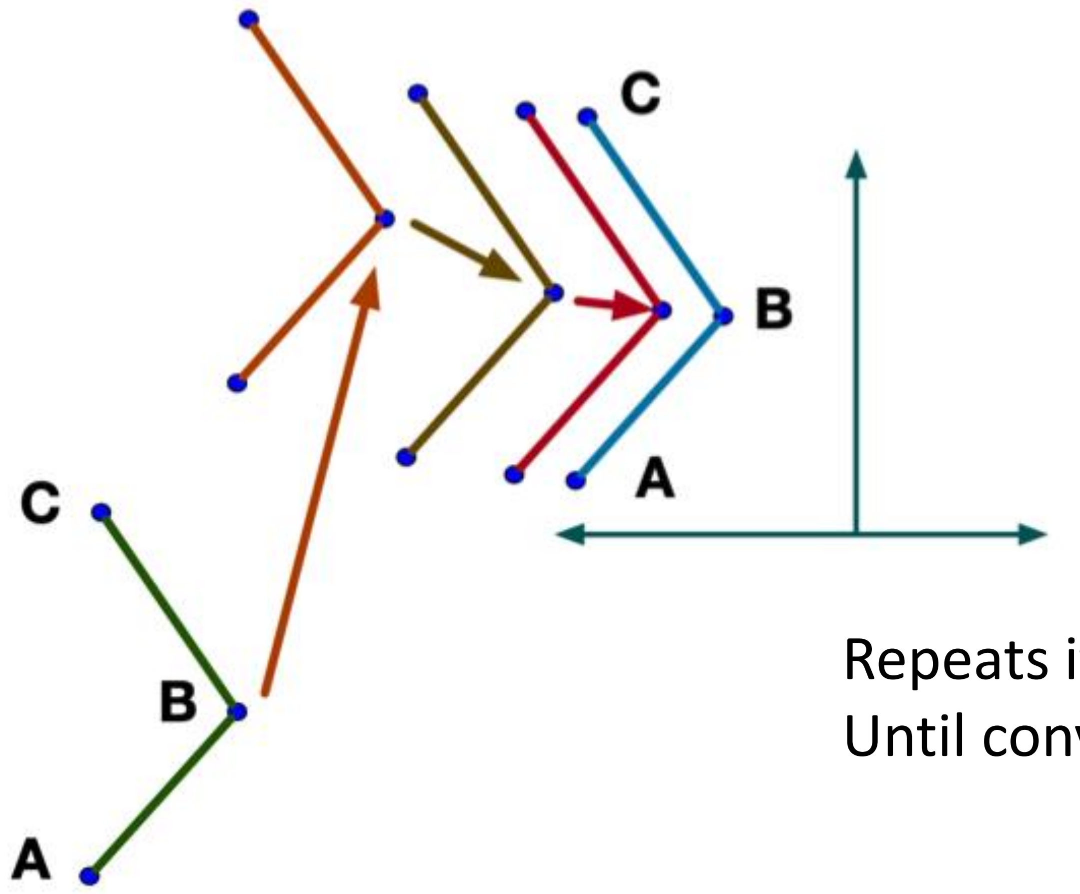
How do you choose between (1) or (2)?

Score (1) > **Score (2)** ⟶ Choose transform (2)



Vanilla Iterative Closest Point (ICP)

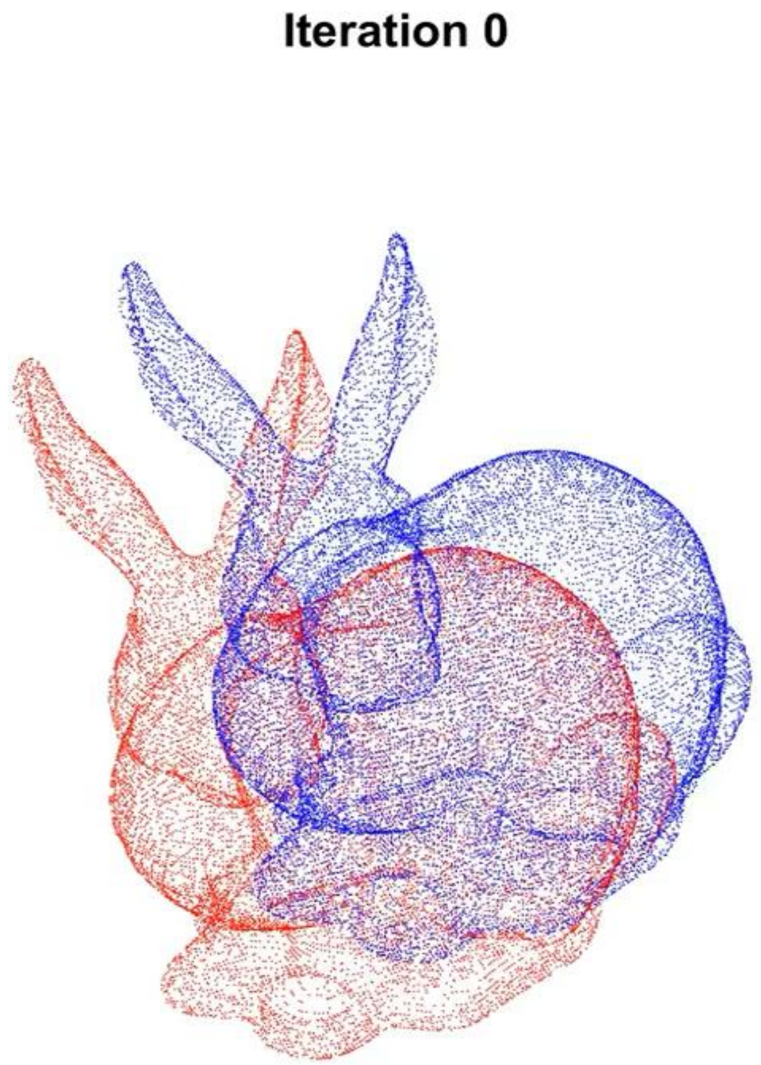
- Uses point-to-point metric



Repeats iterations of algorithm
Until convergence to solution

Vanilla Iterative Closest Point (ICP)

- ICP Example

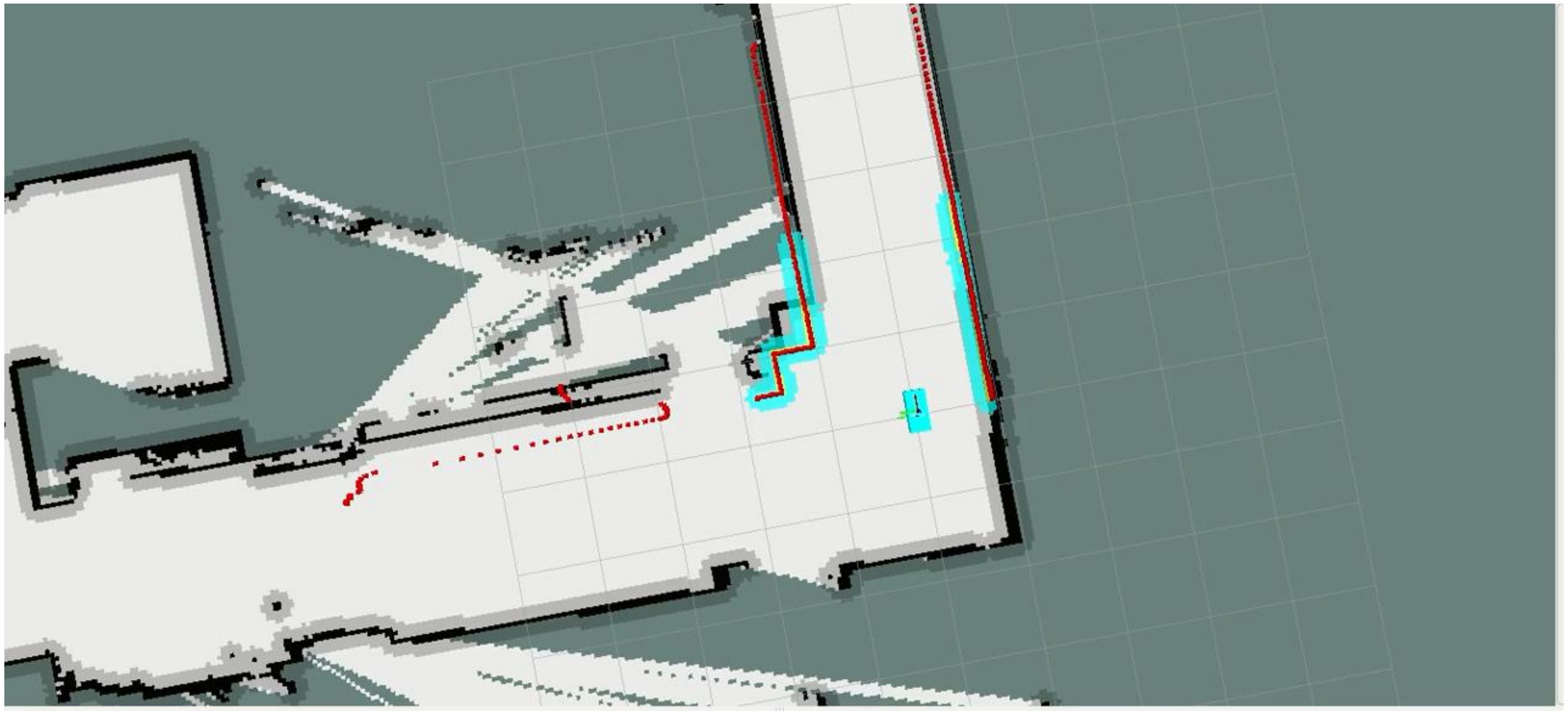


ICP Assumptions and Limitations

- Assumes a sufficiently fast scan rate
  - Significant overlap will exist between sequential scans
  - Points with high correspondence will exist

- Assumes surface will be non-smooth and heterogeneous
  - Smooth surface will look the same to the algorithm

Failed scan matching



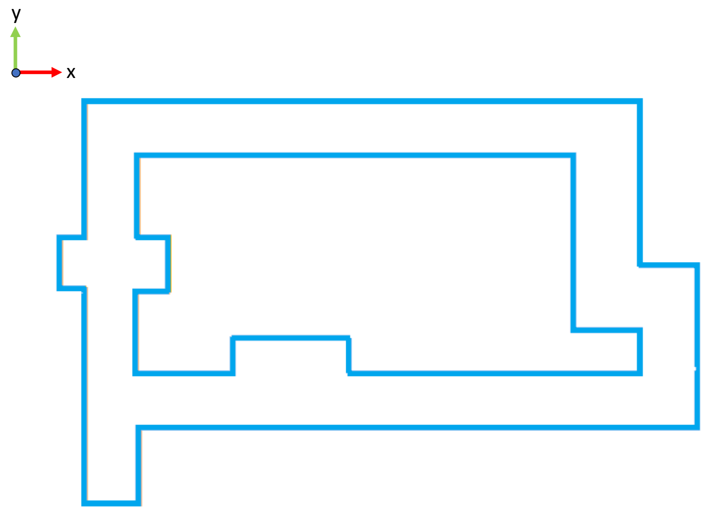
ICP Assumptions and Limitations

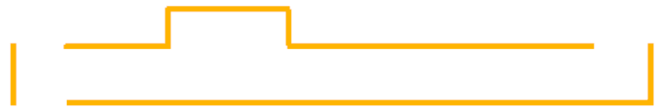
- Assumes a sufficiently fast scan rate
  - - Significant overlap will exist between sequential scans
  - - Points with high correspondence will exist

- Assumes surface will be non-smooth and heterogeneous
  - - Smooth surface will look the same to the algorithm

- Limitations
  - - Initial guess is important for ICP to converge
  - - Vanilla ICP can be slow

Practice: Scan Matching





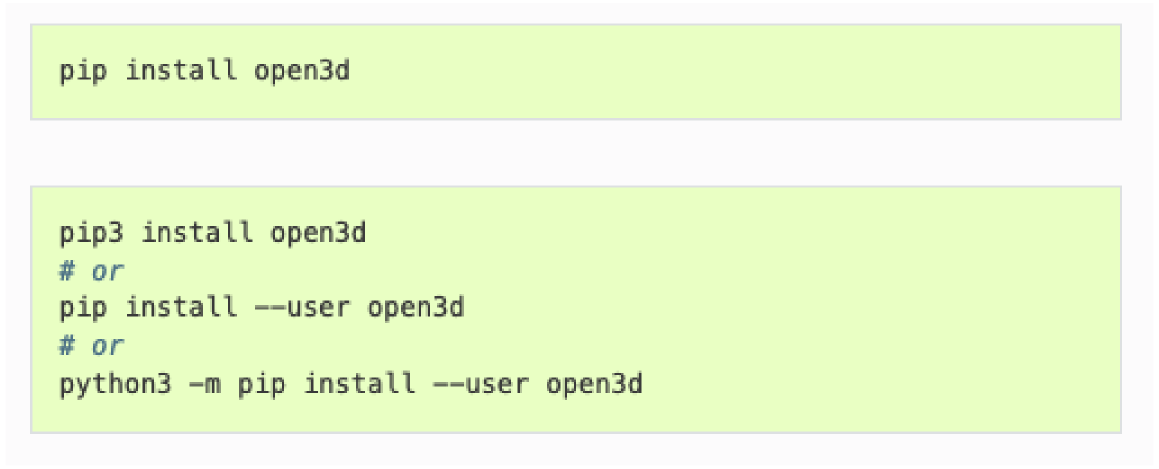
- The goal of this lab is to implement the Scan Matching algorithm.
- In this lab, we will implement Scan Matching, which uses scan data and map data to estimate positions.

Overview

- For this lab, we will implement a ‘**Scan matching**’ in the simulation environment.
- We need to implement the Scan Matching algorithm to determine the vehicle's position.

Environment Setup

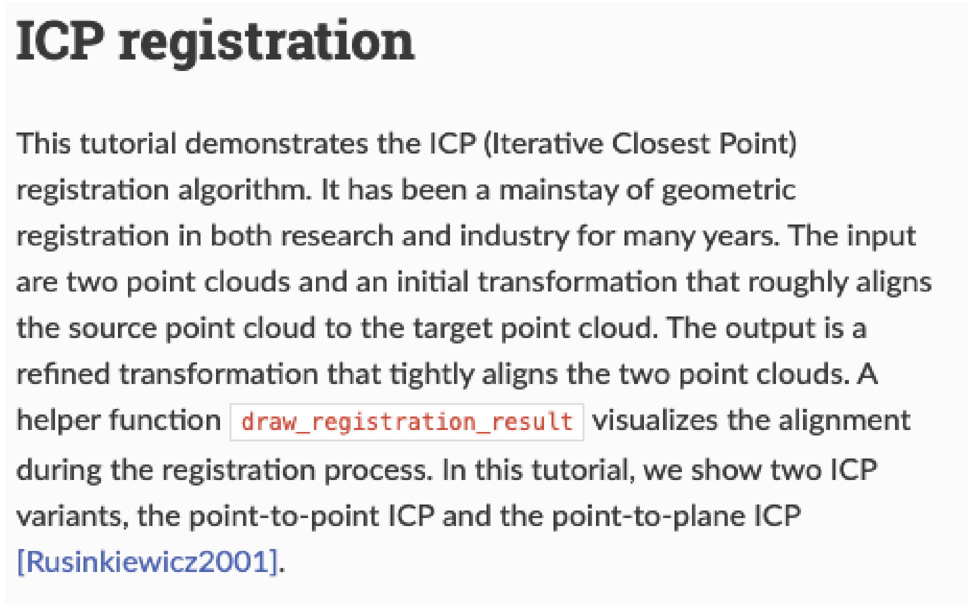
- In the file, you can find 5 pcd file and 1 python file, (map, scan1, scan2, prev_scan, cur_scan, scan_match.py).
- Install Open3D using the pip



```
pip install open3d
```

```
pip3 install open3d
# or
pip install --user open3d
# or
python3 -m pip install --user open3d
```

https://www.open3d.org/docs/0.9.0/getting_started.html



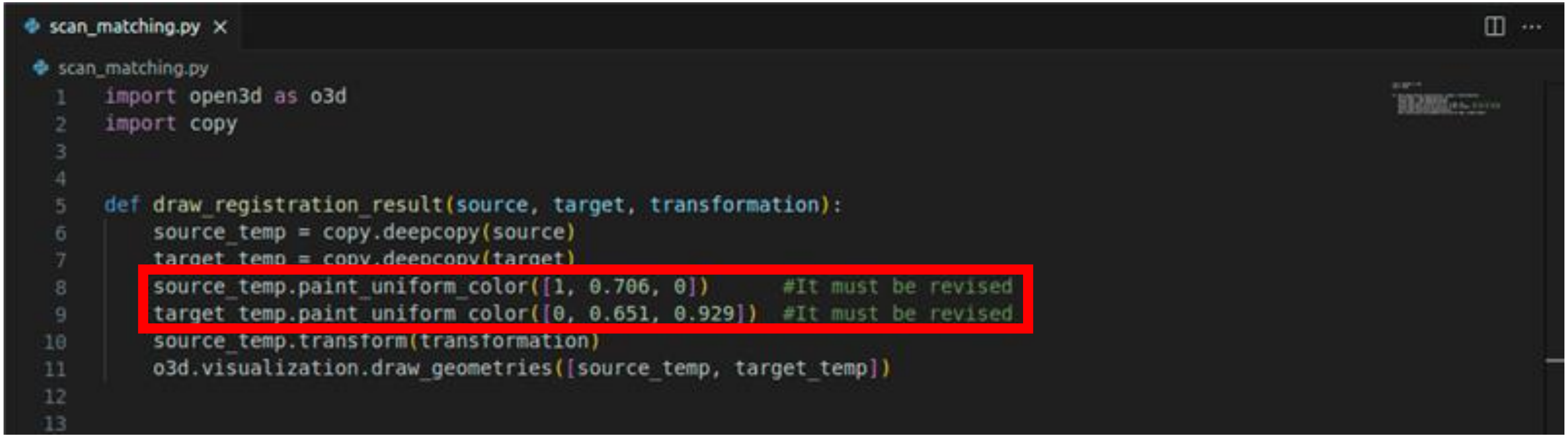
ICP registration

This tutorial demonstrates the ICP (Iterative Closest Point) registration algorithm. It has been a mainstay of geometric registration in both research and industry for many years. The input are two point clouds and an initial transformation that roughly aligns the source point cloud to the target point cloud. The output is a refined transformation that tightly aligns the two point clouds. A helper function `draw_registration_result` visualizes the alignment during the registration process. In this tutorial, we show two ICP variants, the point-to-point ICP and the point-to-plane ICP [Rusinkiewicz2001].

https://www.open3d.org/docs/0.9.0/tutorial/Basic/icp_registration.html

scan_matching.py

- scan_matching.py contains the skeleton code for your scan matching.
- Must change the color of the source and target.

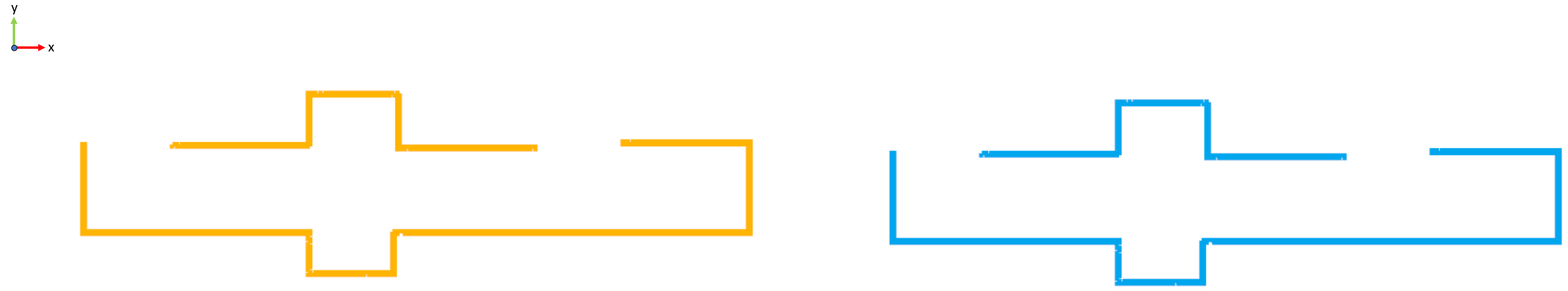


scan_matching.py

```python
import open3d as o3d
import copy


def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([1, 0.706, 0])       #It must be revised
    target_temp.paint_uniform_color([0, 0.651, 0.929])  #It must be revised
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp])
```

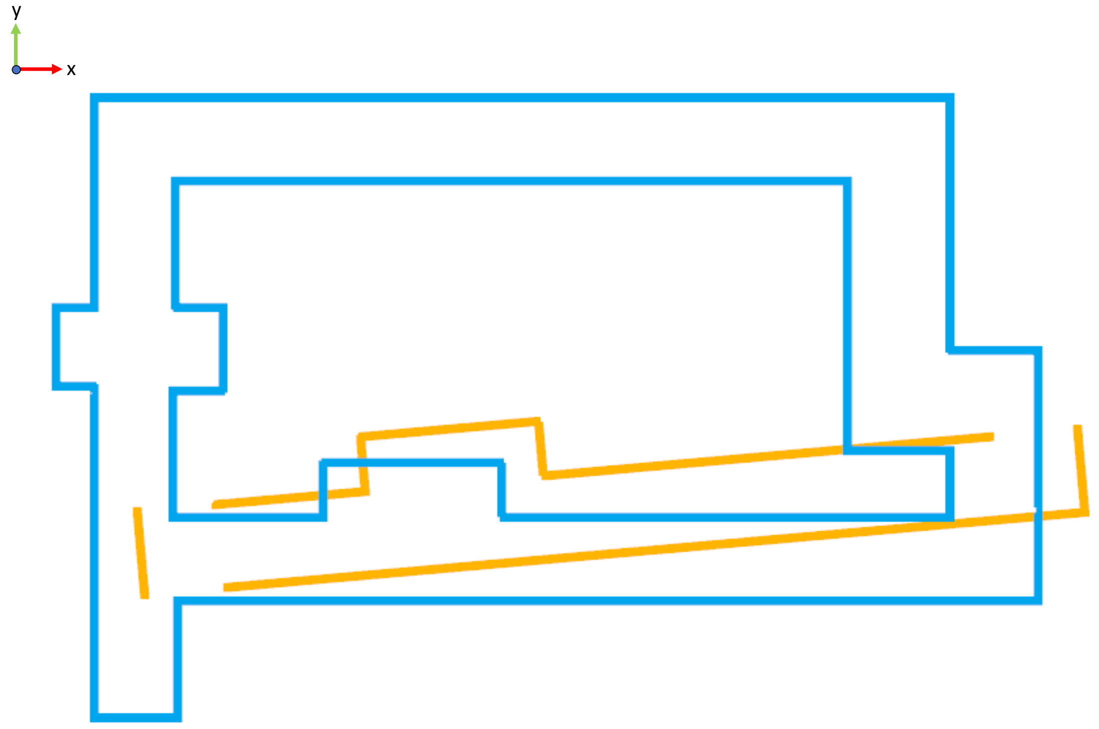
Problem1

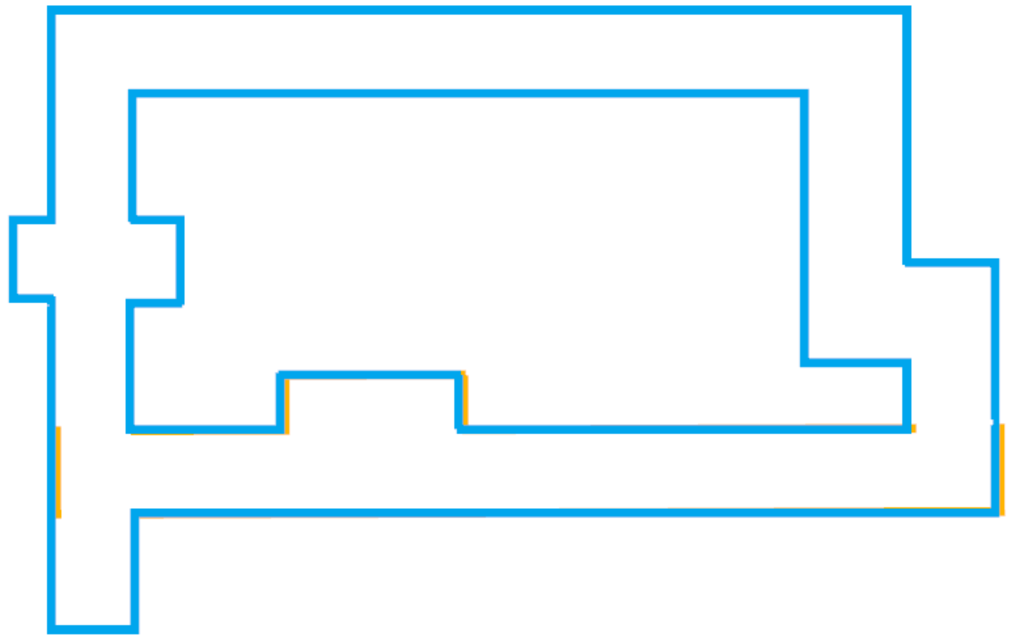
- Yellow scan represents prev_scan, and blue scan represents cur_scan.
- f the vehicle's position for prev_scan is assumed to be (1,1), determine the current vehicle position.



Problem2

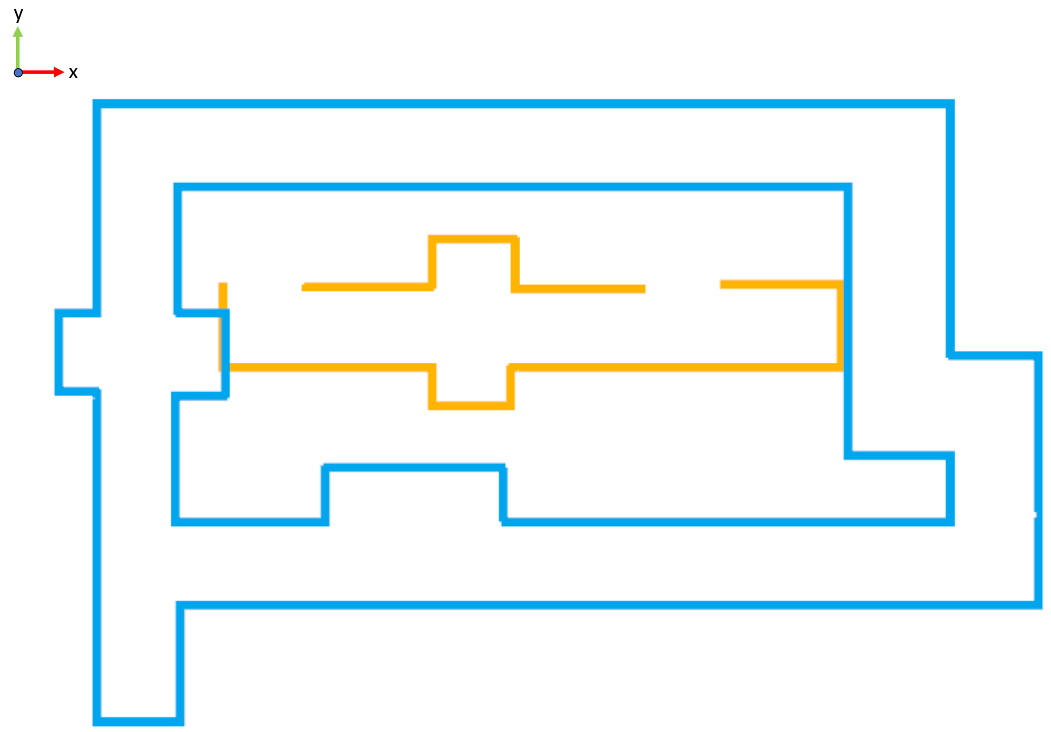
- Yellow scan represents scan_1, and blue scan represents map.
- If the vehicle's position estimation failed for scan_1 and assumed to be (4,3), determine the correct vehicle position.

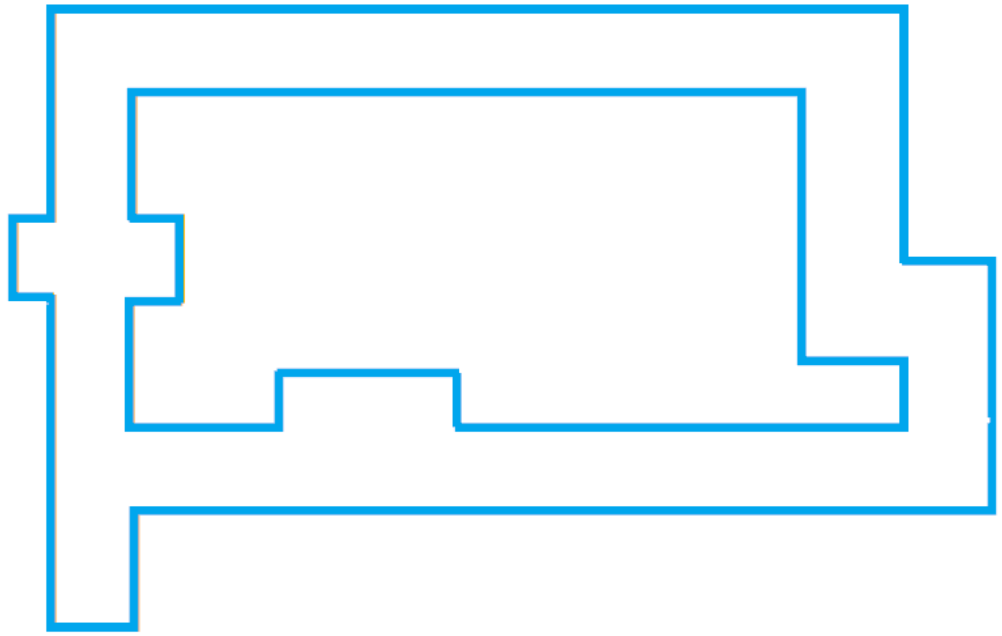




Problem3

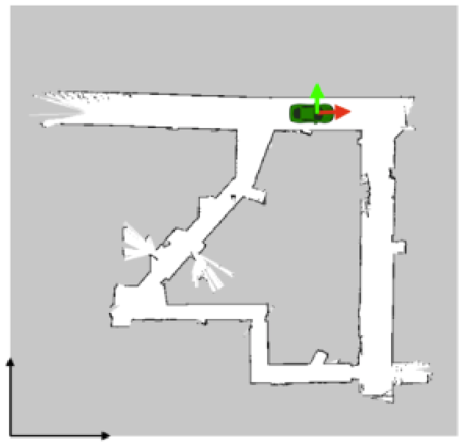
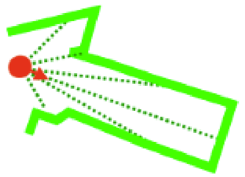
- Yellow scan represents scan_2, and blue scan represents map.
- Calculate the coordinate transformation to align scan_2 with the map.



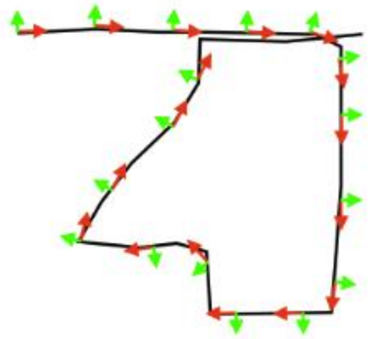
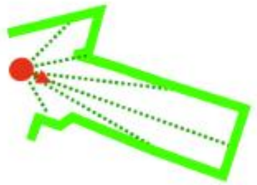


Simultaneous Localization and Mapping (SLAM)

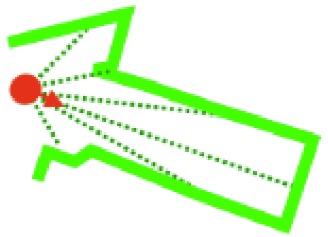
Problem Setting

Localization: given a **map**, use *sensor data* to estimate the current *pose* of the robot

Mapping: given robot *pose* at each time (**trajectory**), use *sensor data* to build map

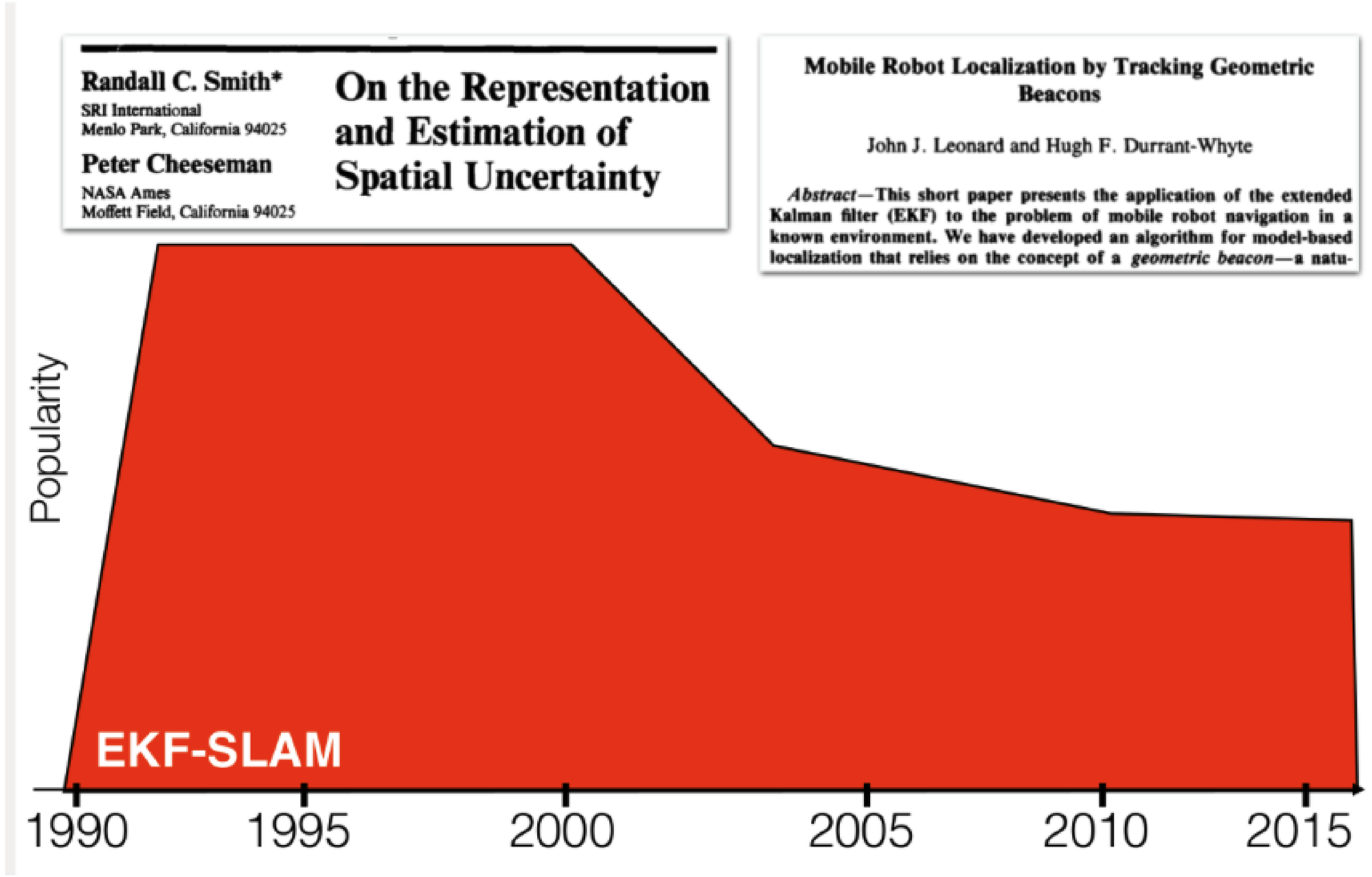


Simultaneous Localization and Mapping (SLAM): use sensor data to build map and estimate robot trajectory

A brief history of SLAM

- **Why do we need a map?**
  - In order to support path planning
  - Limiting the error in state estimates, by providing the opportunity to ‘reset’
  - Later… do we really need map?
- **Historical Development (1986-2004) : Probabilistic Foundations**
  - EKF (you will still find this in visual inertial odometry)
  - Particle Filter (very efficient localization)
  - We will cover these methods next class in the context of localization
- **Modern Era (2004-Now) : Algorithmic Improvements**
  - Maximum a-posteriori Estimation
  - Other names: factor graph optimization, graph-SLAM, smoothing and mapping(SAM), bundle adjustment

A brief history of SLAM



Randall C. Smith*
SRI International
Menlo Park, California 94025
Peter Cheeseman
NASA Ames
Moffett Field, California 94025

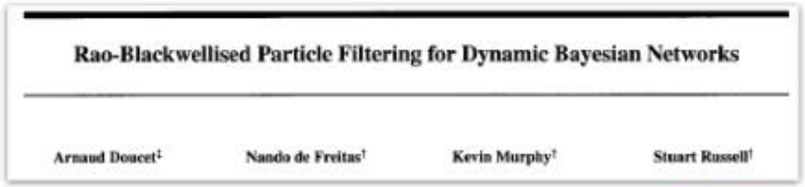
On the Representation and Estimation of Spatial Uncertainty

Mobile Robot Localization by Tracking Geometric Beacons

John J. Leonard and Hugh F. Durrant-Whyte

Abstract—This short paper presents the application of the extended Kalman filter (EKF) to the problem of mobile robot navigation in a known environment. We have developed an algorithm for model-based localization that relies on the concept of a *geometric beacon*—a natu-

A brief history of SLAM



Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks

Arnaud Doucet[‡] Nando de Freitas[†] Kevin Murphy[†] Stuart Russell[†]

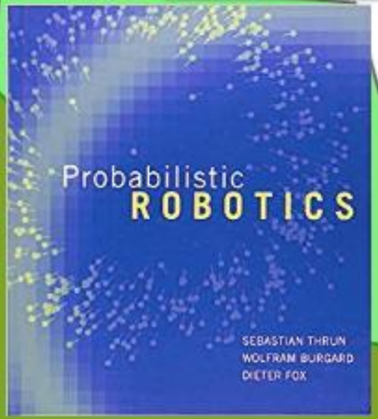
Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM

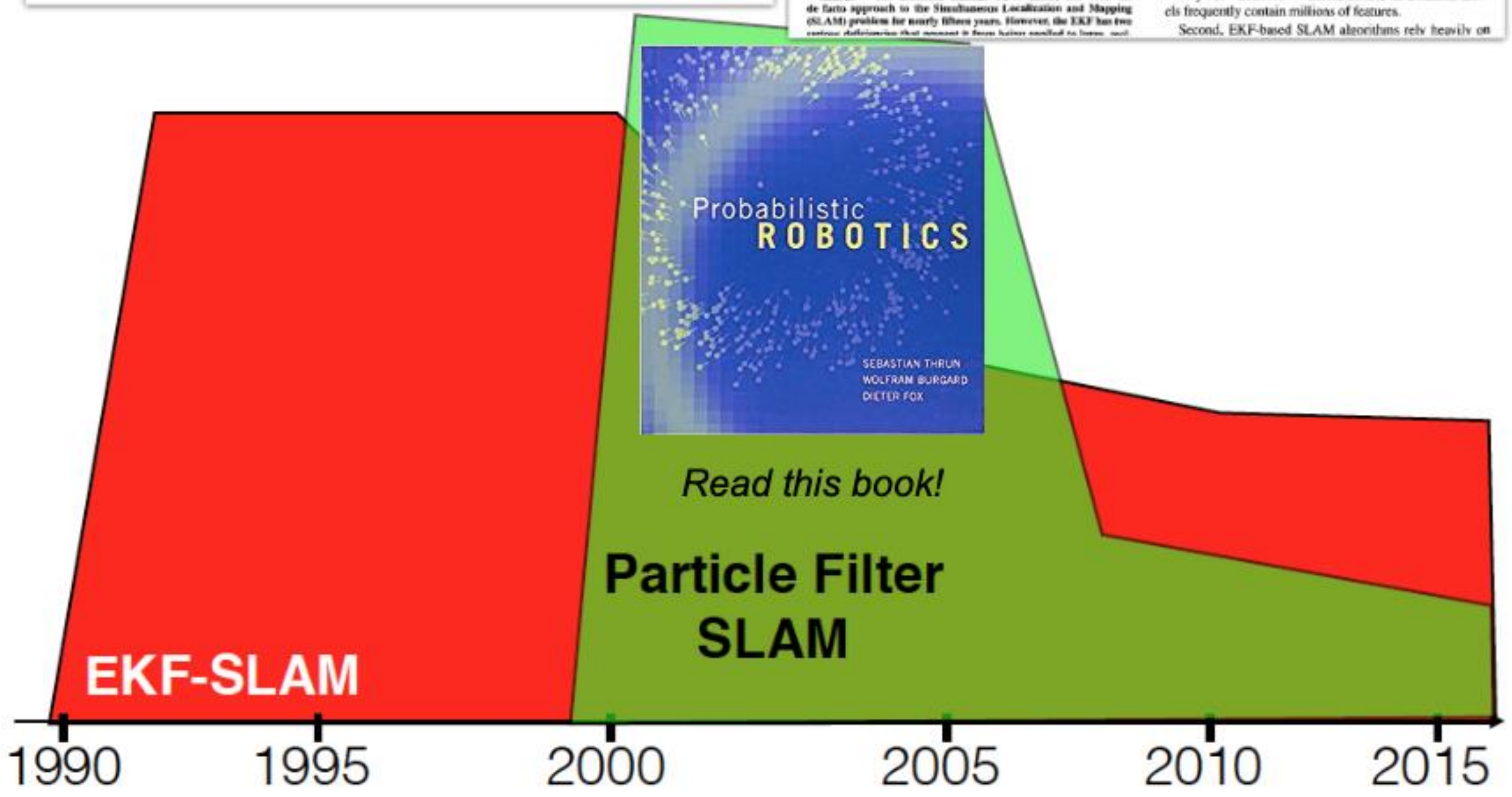
Michael Montemerlo, Sebastian Thrun

Abstract— The Extended Kalman Filter (EKF) has been the de facto approach to the Simultaneous Localization and Mapping (SLAM) problem for nearly fifteen years. However, the EKF has two [illegible]

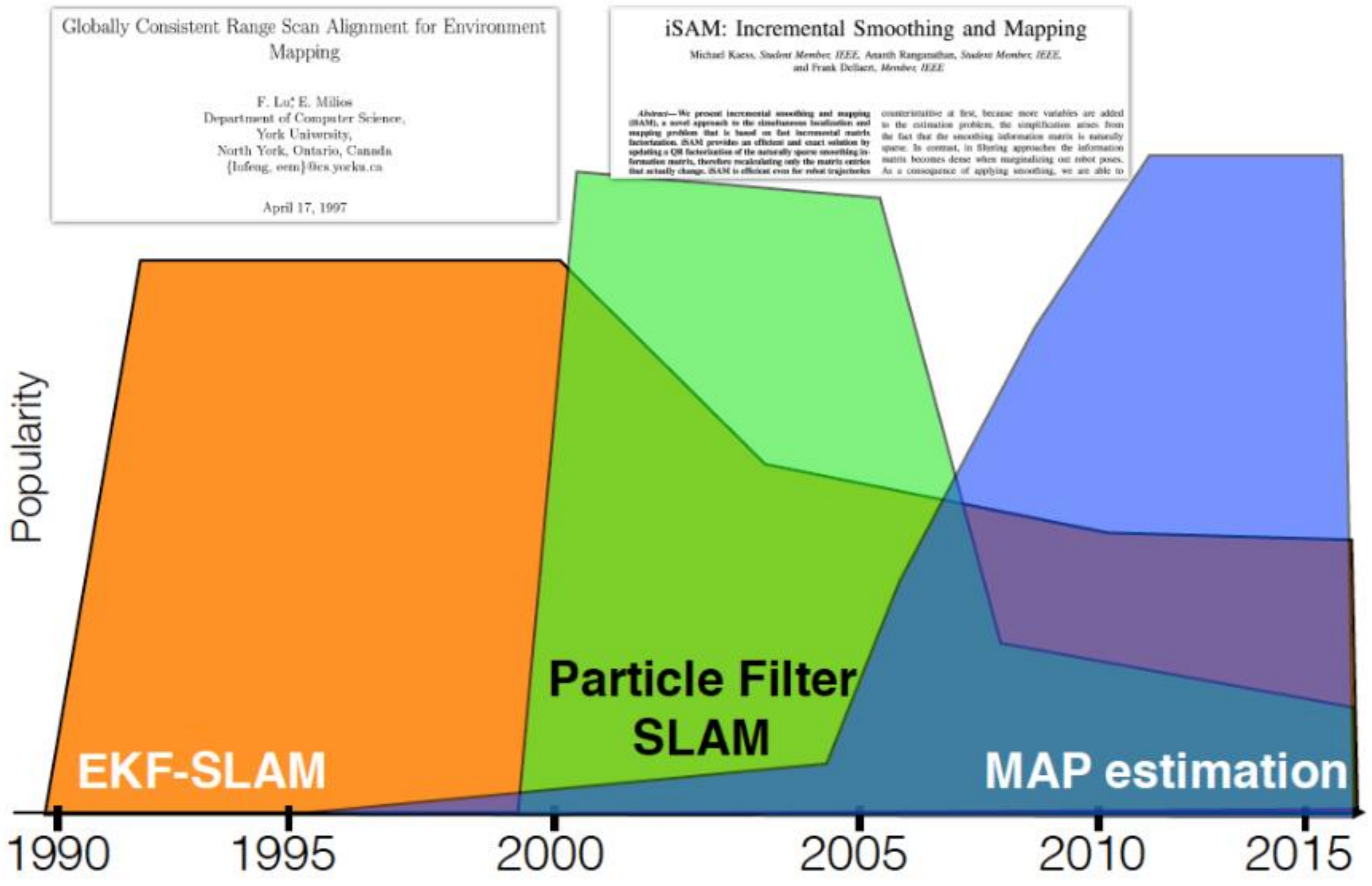
to only a few hundred—whereas natural environment models frequently contain millions of features.

Second, EKF-based SLAM algorithms rely heavily on





A brief history of SLAM



Globally Consistent Range Scan Alignment for Environment Mapping

F. Lu, E. Milios
Department of Computer Science,
York University,
North York, Ontario, Canada
{lufeng, eem}@cs.yorku.ca

April 17, 1997

iSAM: Incremental Smoothing and Mapping

Michael Kaess, *Student Member, IEEE,* Ananth Ranganathan, *Student Member, IEEE,* and Frank Dellaert, *Member, IEEE*

***Abstract*—We present incremental smoothing and mapping (iSAM), a novel approach to the simultaneous localization and mapping problem that is based on fast incremental matrix factorization. iSAM provides an efficient and exact solution by updating a QR factorization of the naturally sparse smoothing information matrix, therefore recalculating only the matrix entries that actually change. iSAM is efficient even for robot trajectories**

counterintuitive at first, because more variables are added to the estimation problem, the simplification arises from the fact that the smoothing information matrix is naturally sparse. In contrast, in filtering approaches the information matrix becomes dense when marginalizing out robot poses. As a consequence of applying smoothing, we are able to

Occupancy Grid Mapping

- Occupancy: binary Random Variable

$$m_{x,y}: \{free, occupied\} \rightarrow \{\, 0, 1 \,\}$$

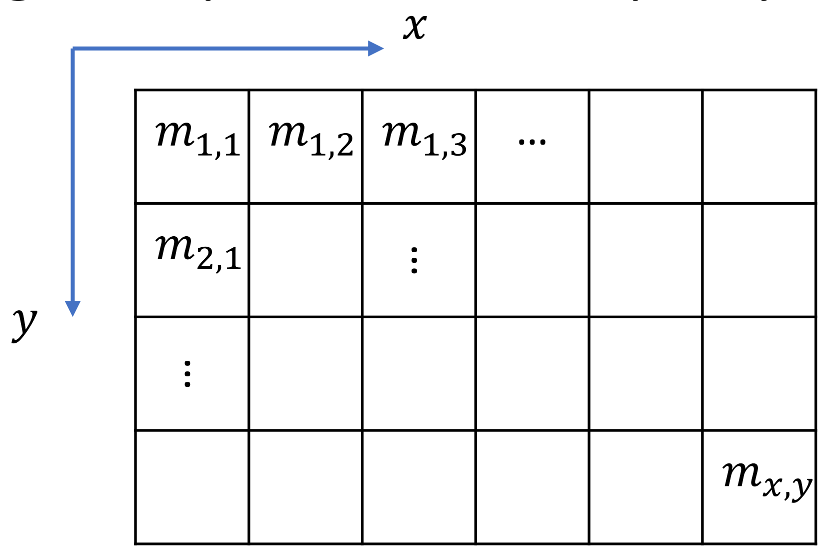
- Review – into Probability
  - Given some probability space (Ω, P), a **random variable** $X: \Omega \;\rightarrow R$ is a *function* that maps the sample space to the reals.

Occupancy Grid Mapping

- Occupancy: binary Random Variable

$$m_{x,y} : \{free, occupied\} \rightarrow \{\, 0, 1 \,\}$$

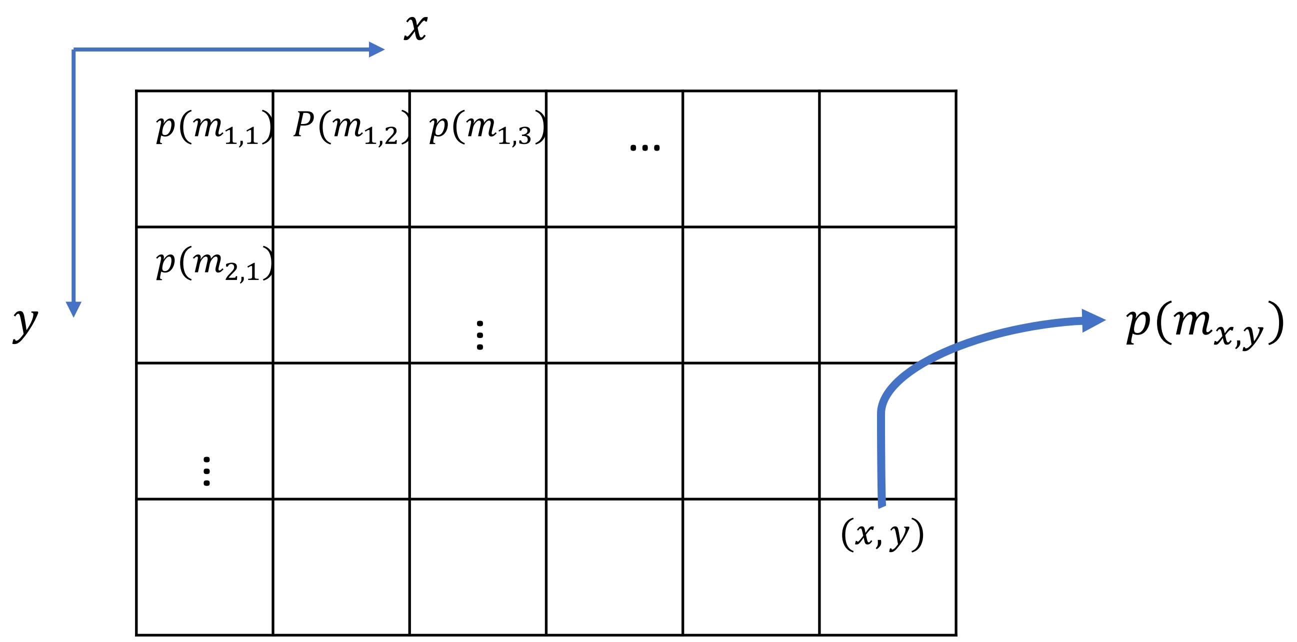
- Occupancy grid map
  - Fine-grained grid map where an occupancy variable associated with each cell

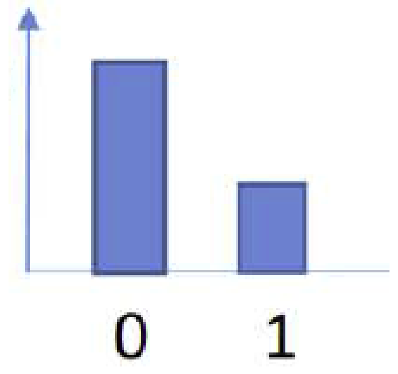


Occupancy Grid Mapping

- Occupancy grid mapping
  - A Bayesian filtering to maintain a occupancy grid map.

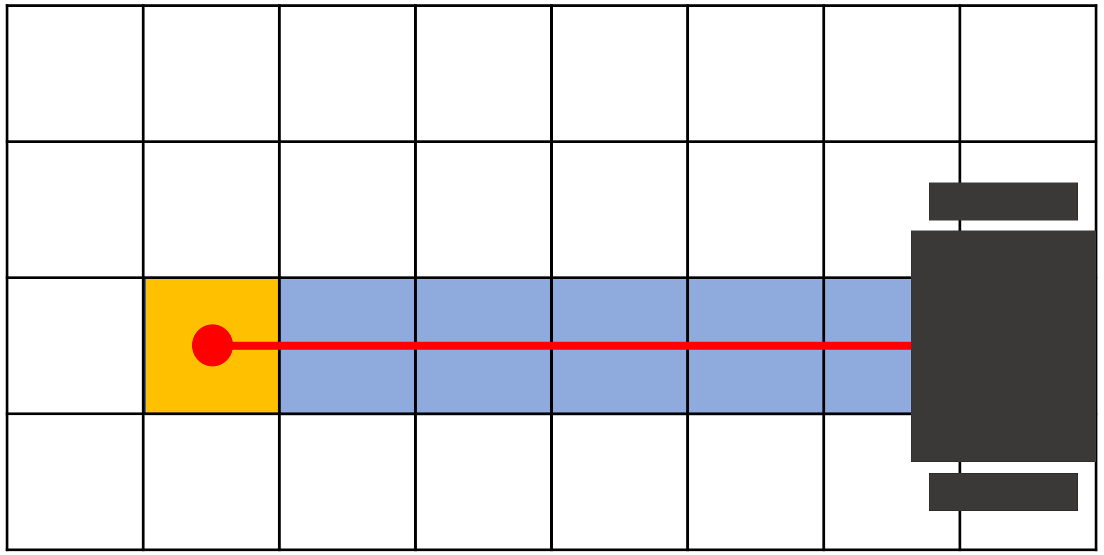
$\hookrightarrow$ Recursively update $p(m_{x,y})$ for each cell





Occupancy Grid Mapping

- Measurement



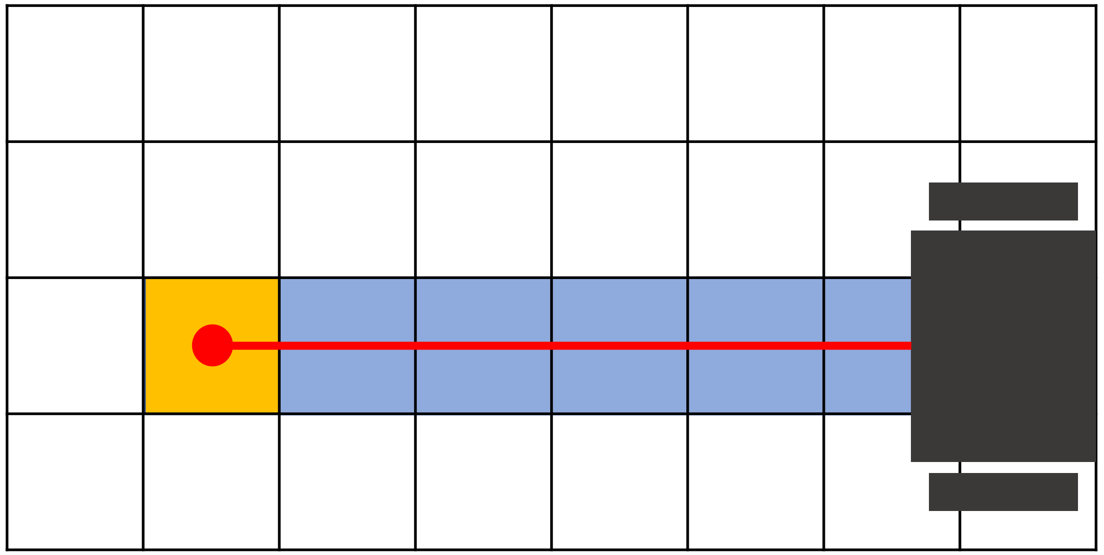
a range sensor

Occupancy Grid Mapping

- Measurement

Free Occupied

$z \sim \{\, 0, 1 \,\}$



a range sensor

Occupancy Grid Mapping

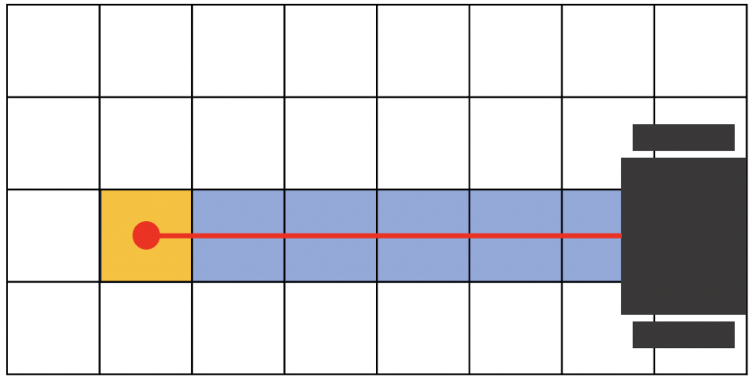
- Measurement

Free Occupied

$$z \sim \{\, 0, 1 \,\}$$

- Measurement model

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



$p(z = 1|m_{x,y} = 1)$: True **occupied** measurement

$p(z = 0|m_{x,y} = 1)$: False **free** measurement

$p(z = 1|m_{x,y} = 0)$: False **occupied** measurement

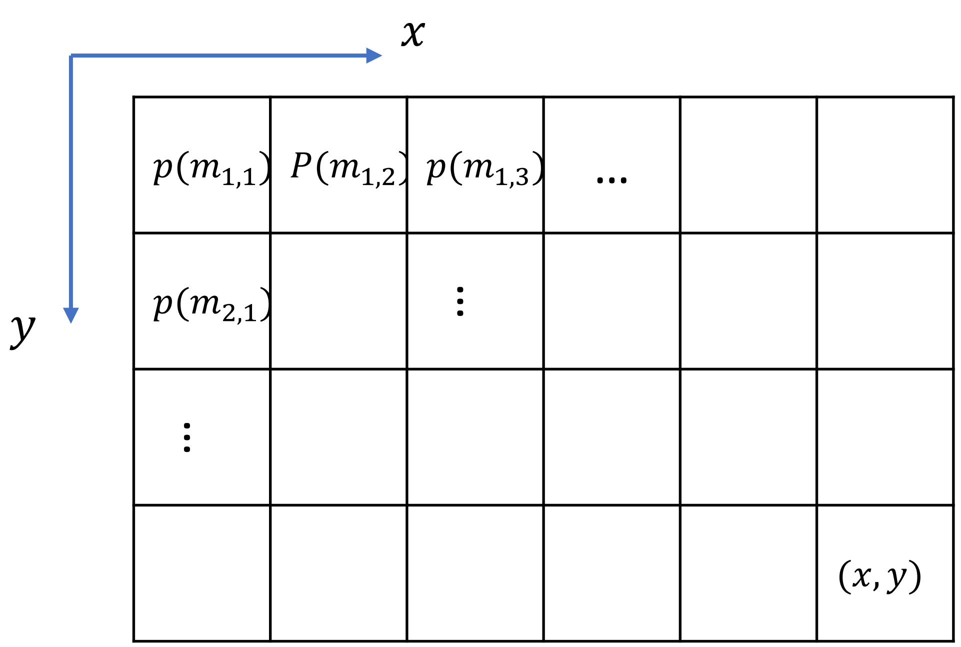
$p(z = 0|m_{x,y} = 0)$: True **free** measurement

Occupancy Grid Mapping

- Measurement Model

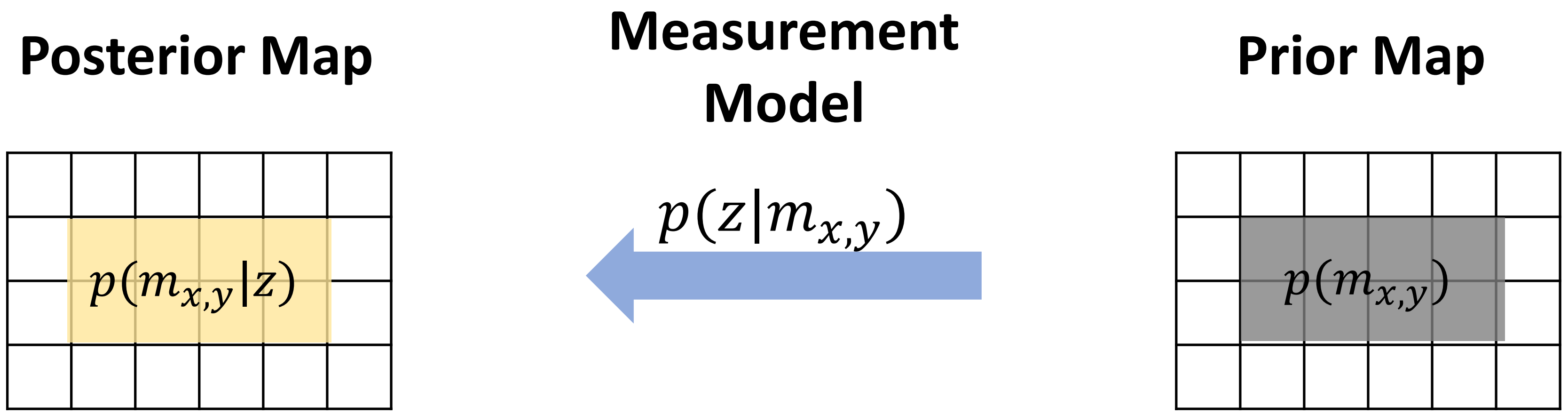
$$p(z|m_{x,y})$$

- Map



Occupancy Grid Mapping

- Measurement Model



$$\overset{\text{Posterior}}{p(m_{x,y}|z)} = \frac{\overset{\text{Likelihood}}{p(z|m_{x,y})}\overset{\text{Prior}}{p(m_{x,y})}}{\underset{\text{Evidence}}{p(z)}}$$

Occupancy Grid Mapping

- More convenient when we use **"Odd"**

$$\boxed{odd := \frac{(X\ happens)}{(X\ not\ happens)} = \frac{p(X)}{p(X^c)}}$$

$$odd\left((m_{x,y} = 1)\ given\ z\right) = \frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)}$$

Occupancy Grid Mapping

- Odd

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

Occupancy Grid Mapping

- Odd

(Bayes' Rule)

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

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

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

(Bayes' Rule)

Occupancy Grid Mapping

- Odd

(Bayes' Rule)

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

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

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

(Bayes' Rule)

Occupancy Grid Mapping

- Odd

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

- Take the log

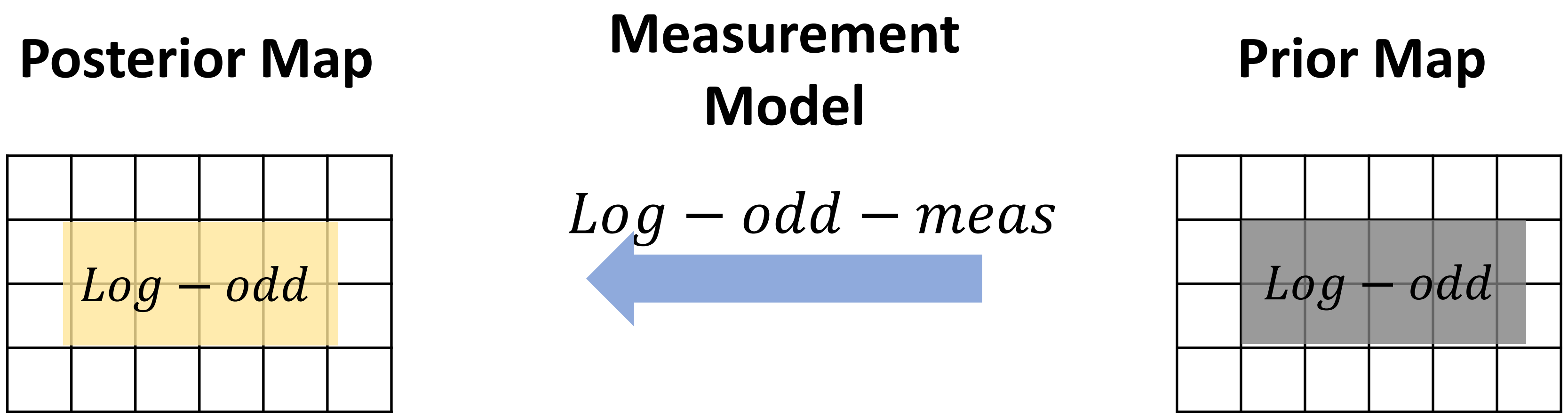
$$Log - odd = log\frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)} = log\frac{p(z|m_{x,y} = 1)p(m_{x,y} = 1)}{p(z|m_{x,y} = 0)p(m_{x,y} = 0)}$$

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

$$\boxed{\log odd^{+} = \log odd\ meas + \log odd^{-}}$$

Occupancy Grid Mapping

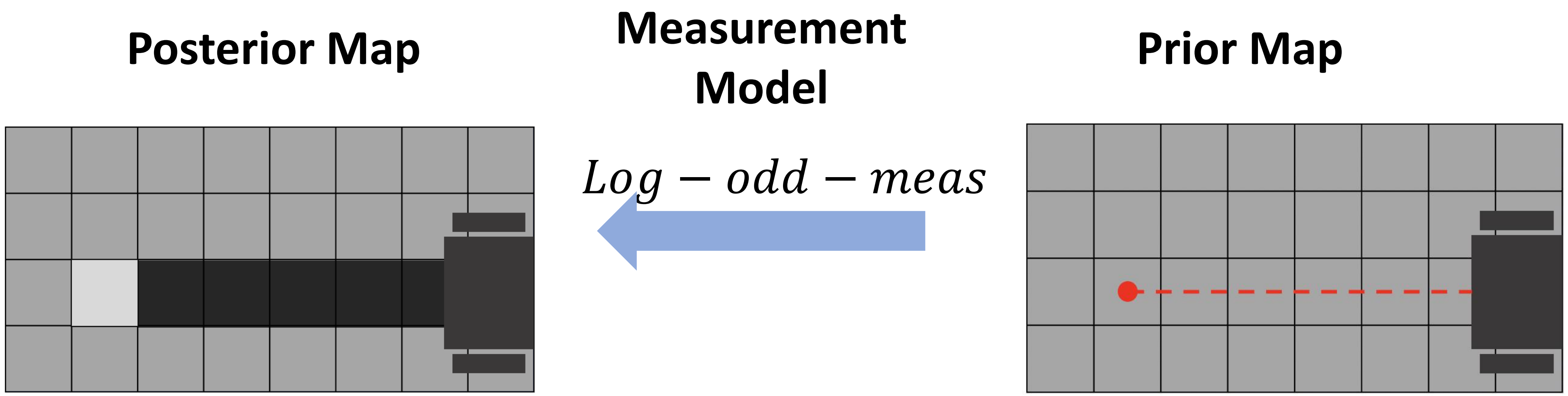
- Log-odd update



$$\log odd^{+} = \log odd\ meas + \log odd^{-}$$

Occupancy Grid Mapping

- Log-odd update



$$\log odd^{+} = \log odd\ meas + \log odd^{-}$$

Occupancy Grid Mapping

- Measurement model in log-odd form

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

- Two possible measurement:
  - Case 1 : cells with z=1

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

  - Case 2 : cells with z=0

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

(Trivial Case : cells not measured)

Occupancy Grid Mapping

- Example

- Constant Measurement Model
  - $\log odd_{occ} \coloneqq 0.9$
  - $\log odd_{free} \coloneqq 0.7$

- Initial Map:
  - $\log odd = 0$ for all (x, y)

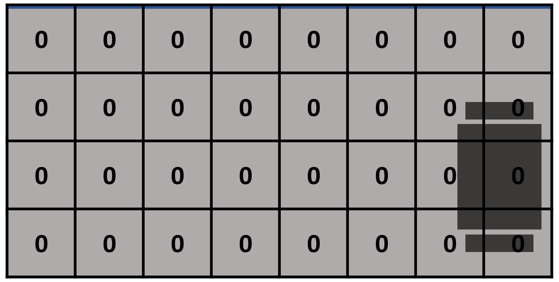
    $\Updownarrow$

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

Update Rule

$$\boxed{\log odd \mathrel{+}= \log odd\_meas}$$

t_0



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Occupancy Grid Mapping

- Example

- Constant Measurement Model
  - $\log odd_{occ} := 0.9$
  - $\log odd_{free} := 0.7$

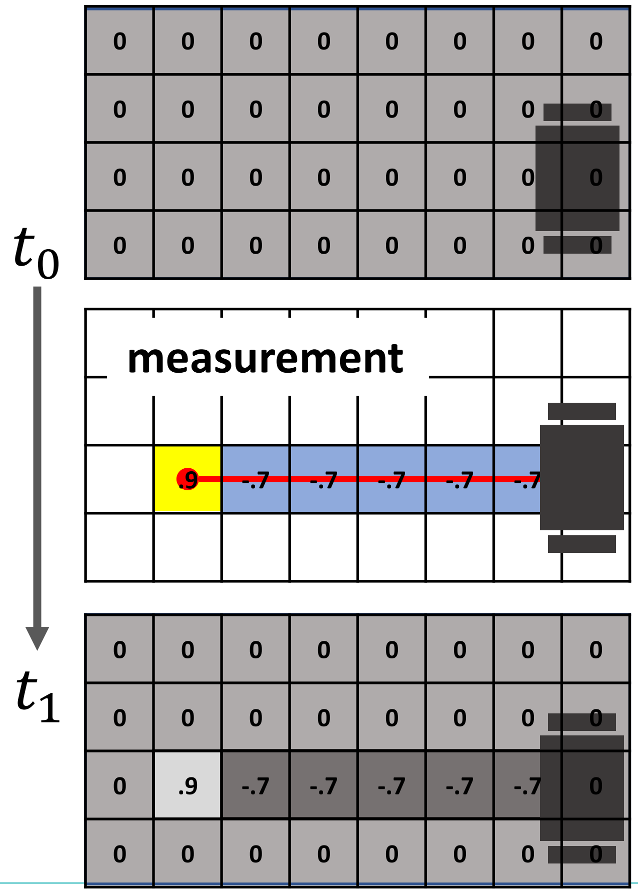
- Update:
  - Case 1 : cells with z=1

    $\log odd \leftarrow 0 + \log odd_{occ}$
  - Case 2 : cells with z=0

    $\log odd \leftarrow 0 - \log odd_{free}$

Update Rule

$$\boxed{\log odd \mathrel{+}= \log odd\_meas}$$



Occupancy Grid Mapping

- Example

- Constant Measurement Model
  - $\log odd_{occ} \coloneqq 0.9$
  - $\log odd_{free} \coloneqq 0.7$

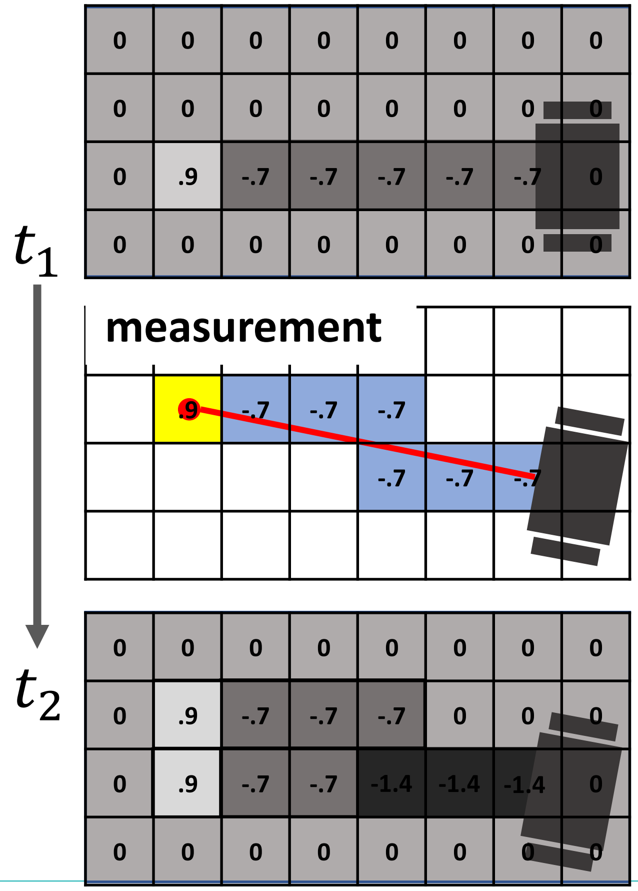
- Update:
  - Case 1 : cells with z=1

    $\log odd \leftarrow 0 + \log odd_{occ}$
  - Case 2 : cells with z=0

    $\log odd \leftarrow 0 - \log odd_{free}$

Update Rule

$$\boxed{\log odd \mathrel{+}= \log odd\_meas}$$



t_1:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	.9	-.7	-.7	-.7	-.7	-.7	0
0	0	0	0	0	0	0	0

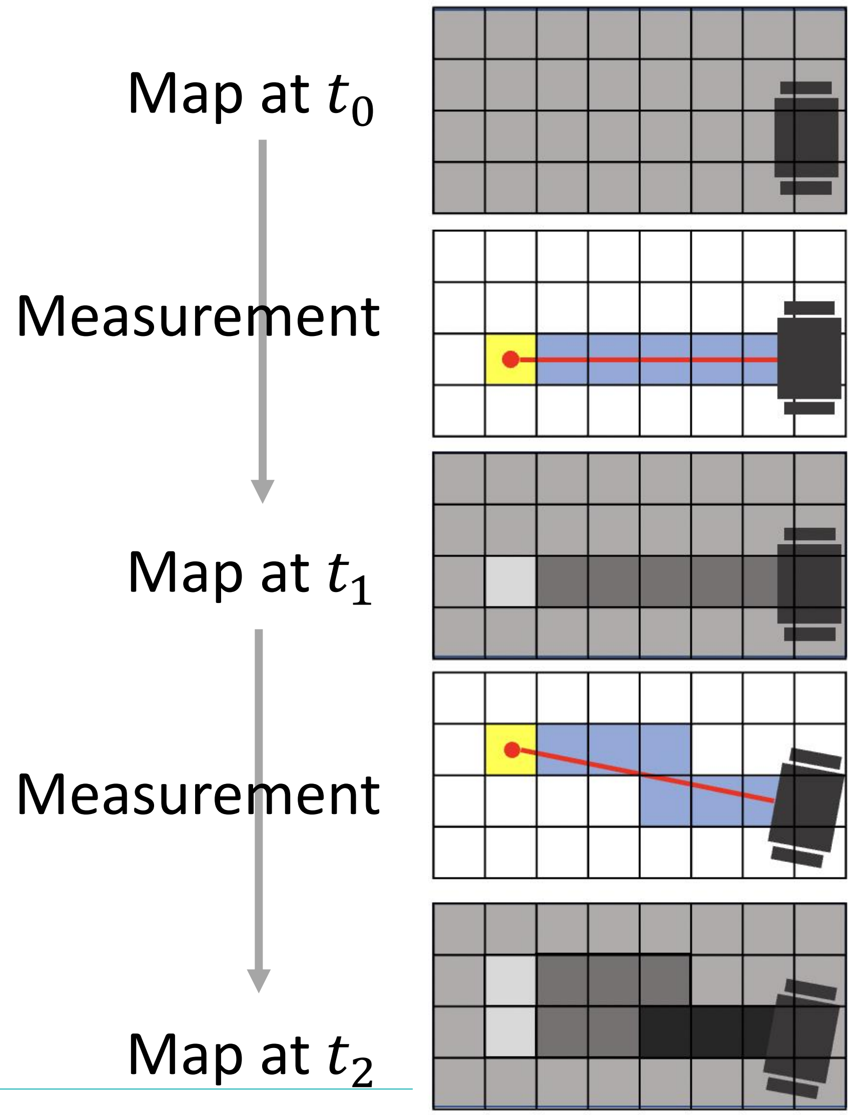
measurement:

	.9	-.7	-.7	-.7			
				-.7	-.7	-.7	

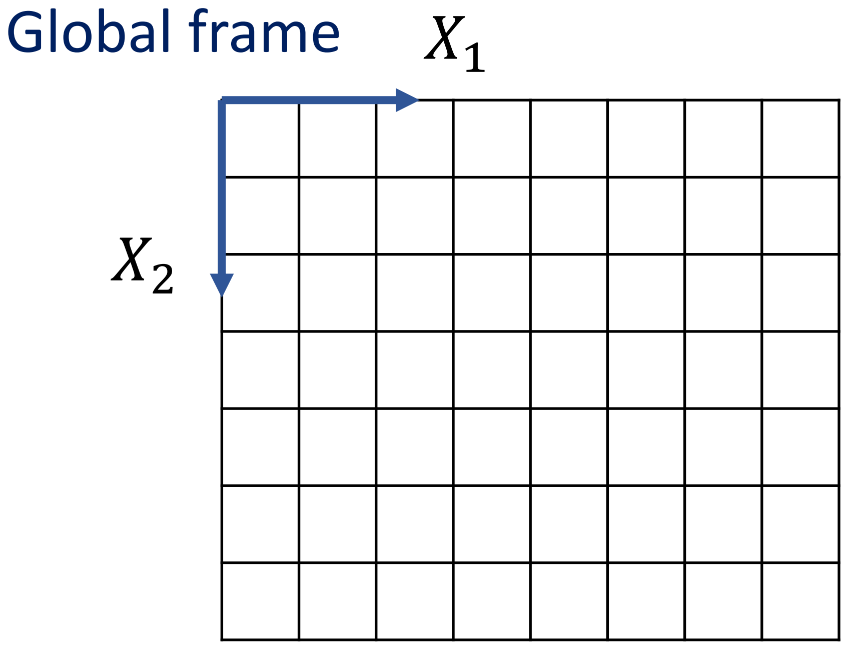
t_2:

0	0	0	0	0	0	0	0
0	.9	-.7	-.7	-.7	0	0	0
0	.9	-.7	-.7	-1.4	-1.4	-1.4	0
0	0	0	0	0	0	0	0

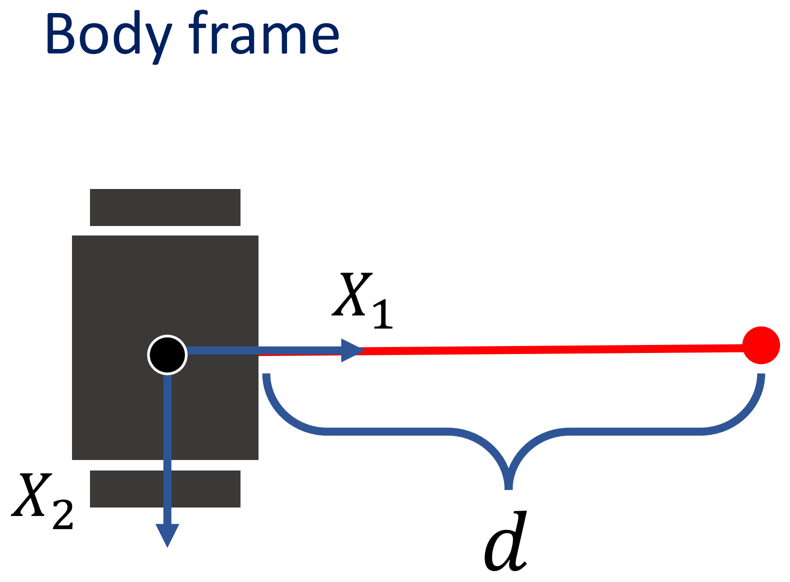
Occupancy Grid Mapping



Handling Range Measurement on Grid

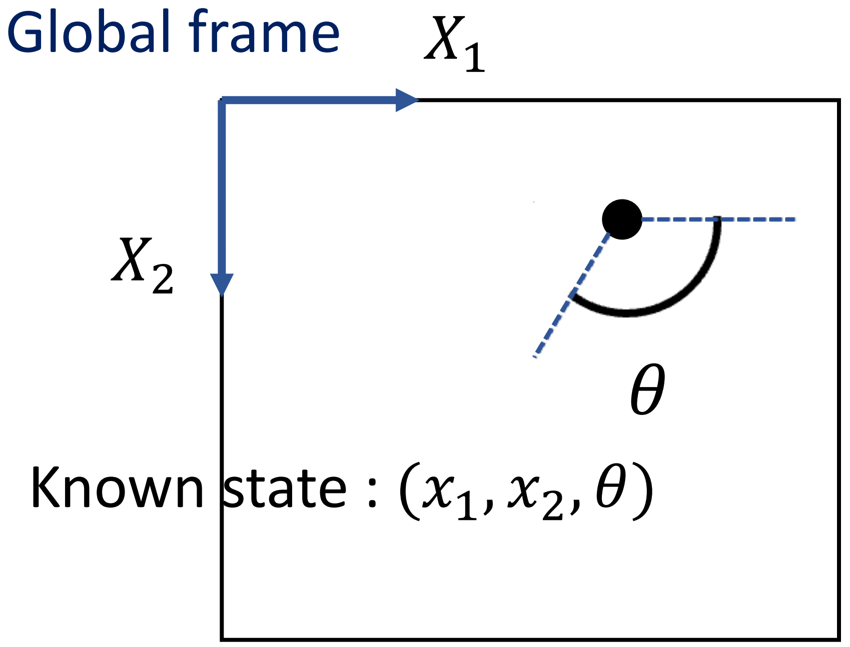


The Map



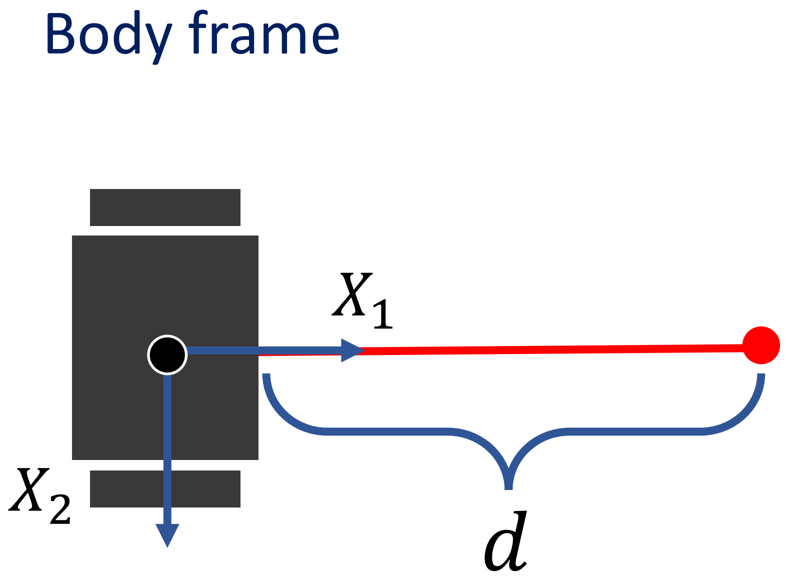
The Robot

Handling Range Measurement on Grid



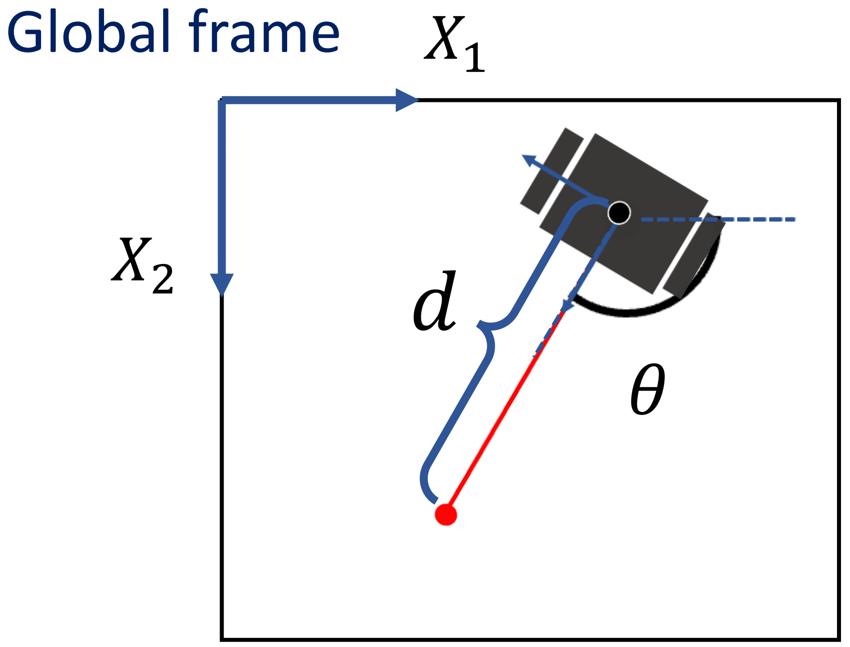
Known state : (x_1, x_2, θ)

The Map



The Robot

Handling Range Measurement on Grid



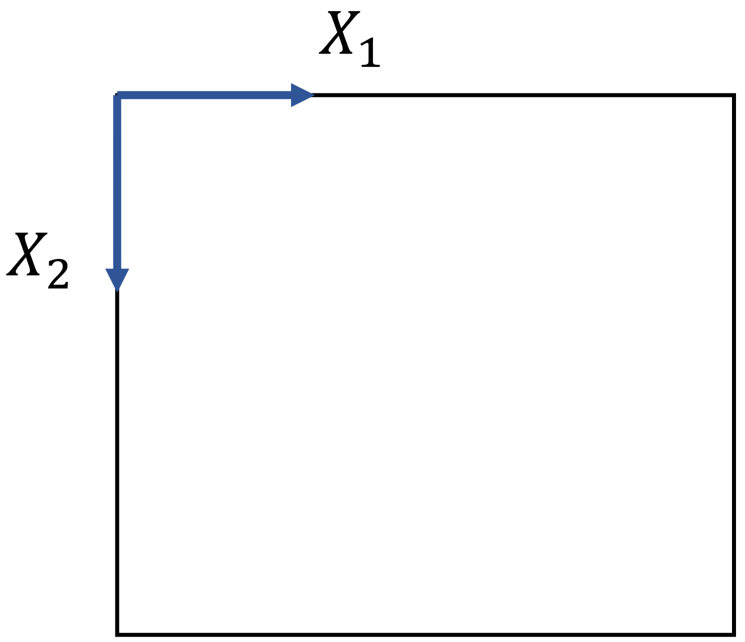
The Map

Distance measurement: d

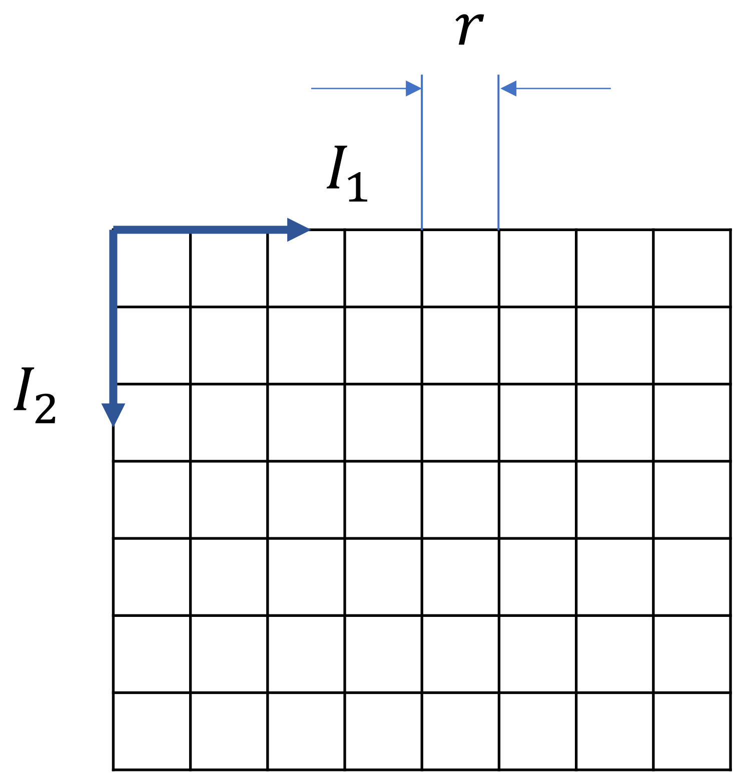
Known state : (x_1, x_2, θ)

$$\begin{bmatrix} x_{1,occ} \\ x_{2,occ} \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} d \\ 0 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Handling Range Measurement on Grid



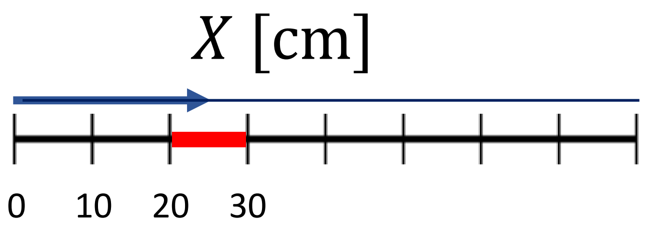
The (Continuous) Map

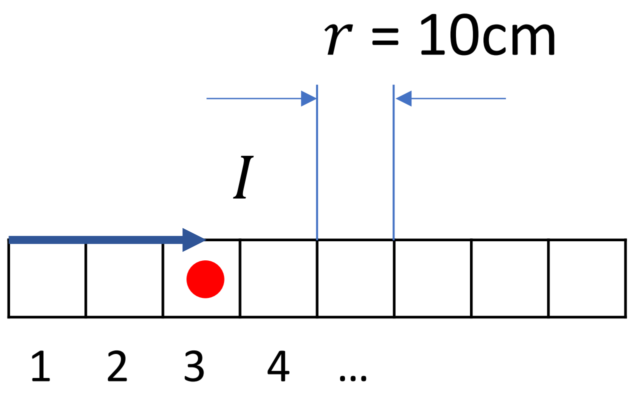


The Discretized Map

Handling Range Measurement on Grid

- Example





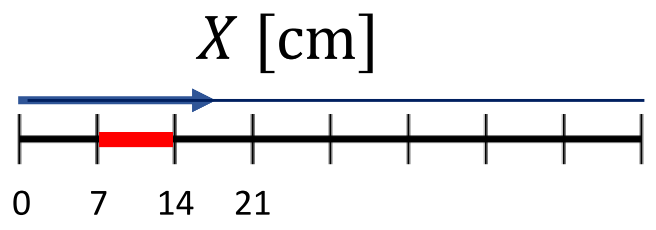
[cm]

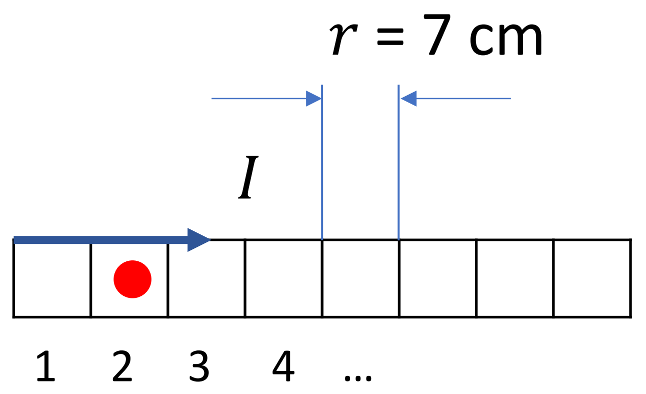
$$
\begin{array}{rcl}
0 < x \leq 10 & \Rightarrow & i = 1 \\
10 < x \leq 20 & \Rightarrow & i = 2 \\
20 < x \leq 30 & \Rightarrow & i = 3 \\
\vdots & & \vdots
\end{array}
$$

[index]

Handling Range Measurement on Grid

- Example





[cm]

$$0 < x \leq 7 \Rightarrow i = 1$$

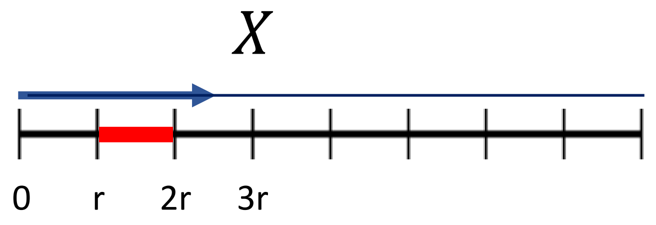
$$7 < x \leq 14 \Rightarrow i = 2$$

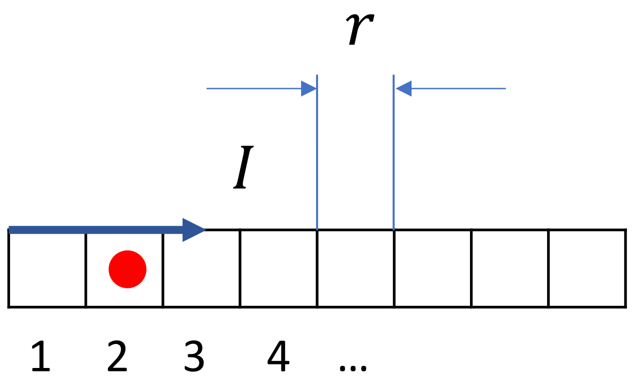
$$\vdots \qquad \vdots$$

[index]

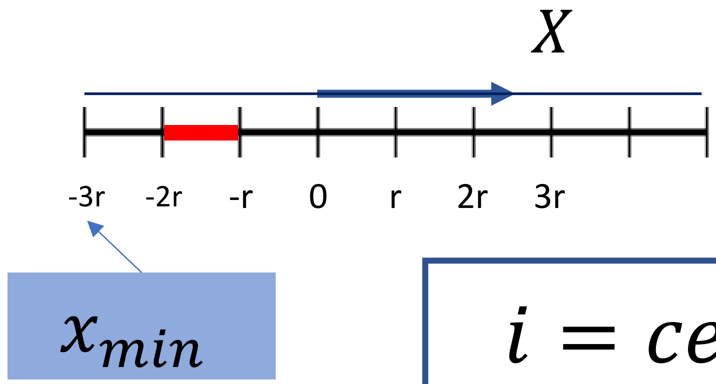
Handling Range Measurement on Grid

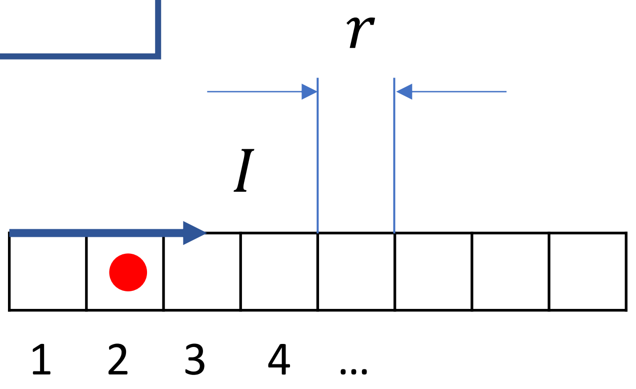
- Example





$$\boxed{i = ceil(x/r)}$$

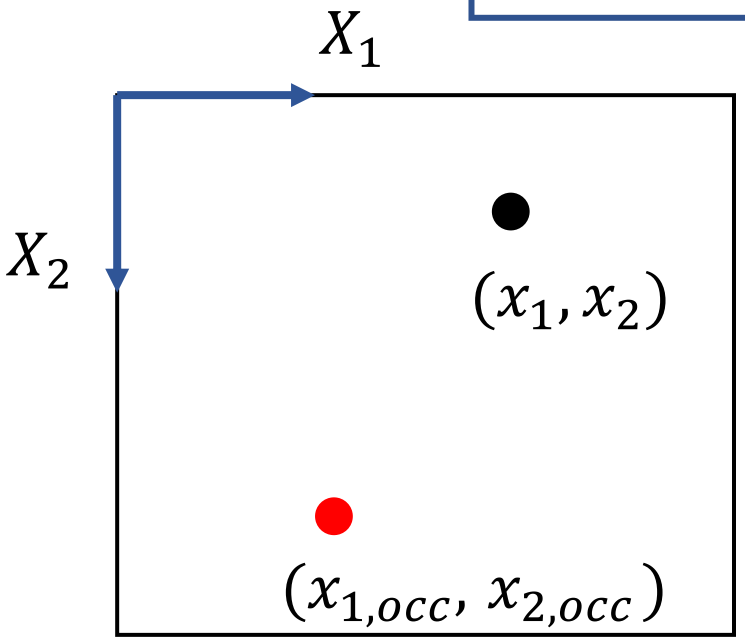




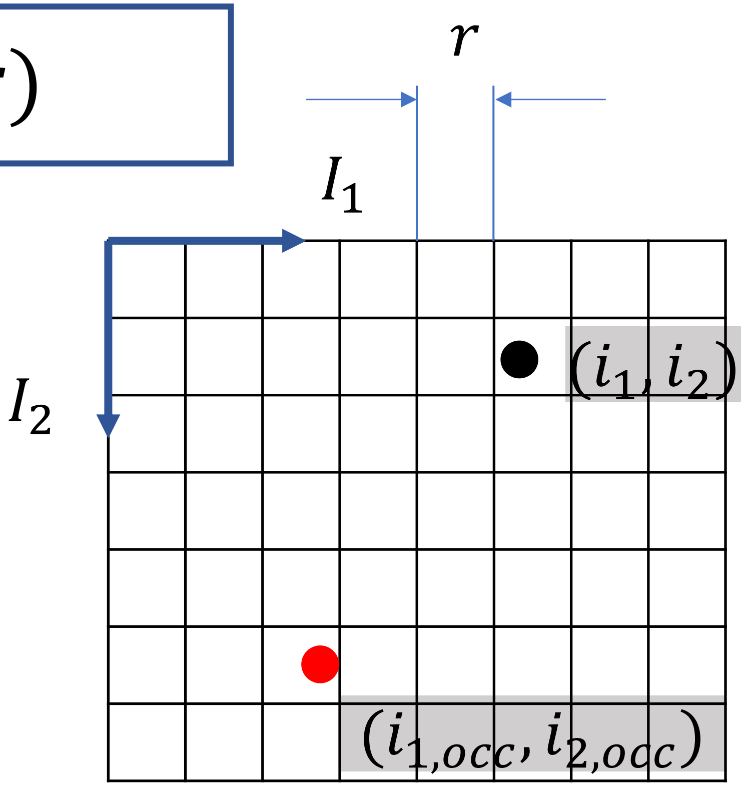
$$\boxed{i = ceil((x - x_{min})/r)}$$

Handling Range Measurement on Grid

$$\boxed{i = ceil(x/r)}$$

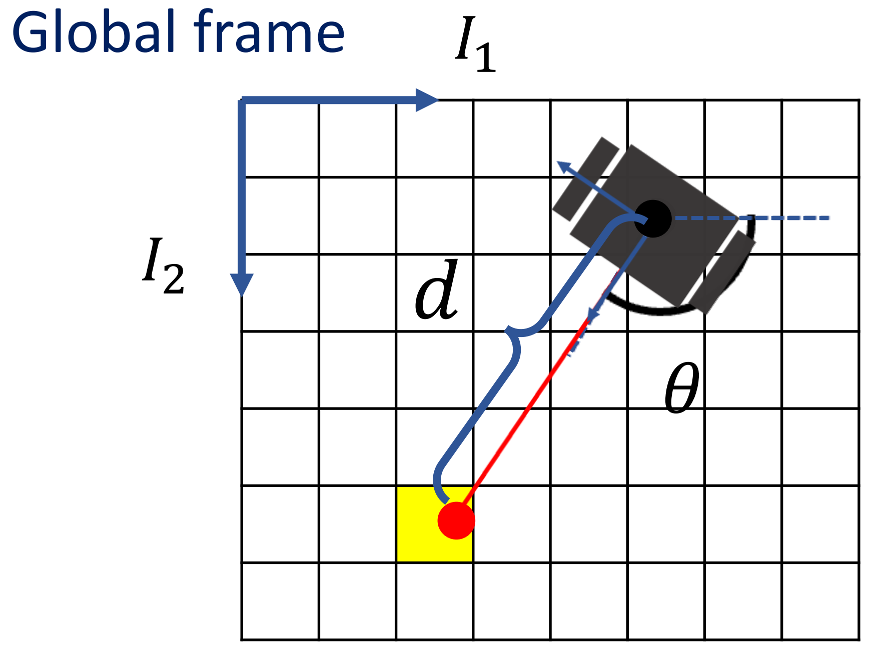


The (Continuous) Map



The Discretized Map

Handling Range Measurement on Grid



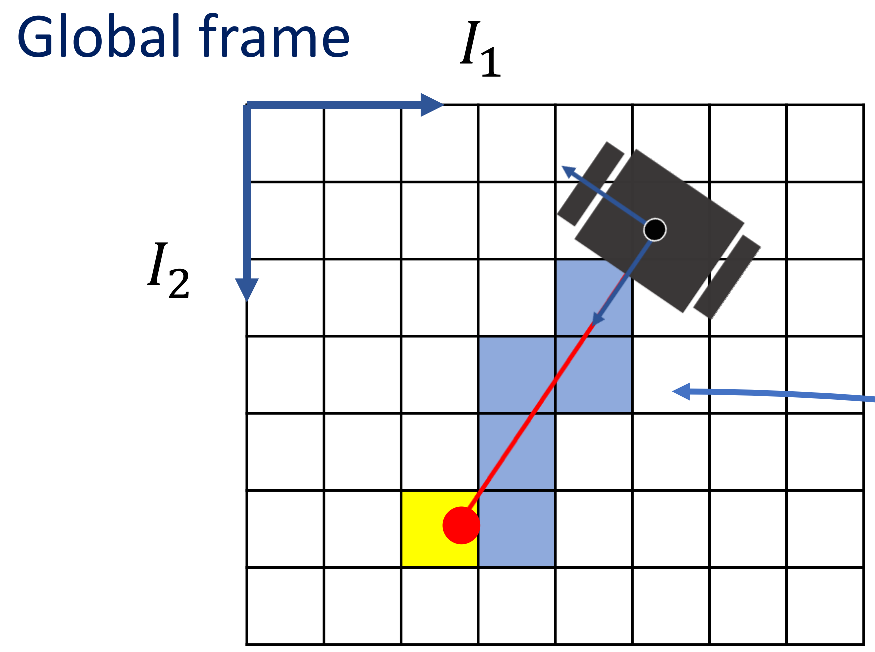
The Map

Distance measurement: d
Known state : (x_1, x_2, θ)

$$\begin{bmatrix} x_{1,occ} \\ x_{2,occ} \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} d \\ 0 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} i_{1,occ} \\ i_{2,occ} \end{bmatrix} = ceil\left(\frac{1}{r} \begin{bmatrix} x_{1,occ} \\ x_{2,occ} \end{bmatrix} \right)$$

Handling Range Measurement on Grid

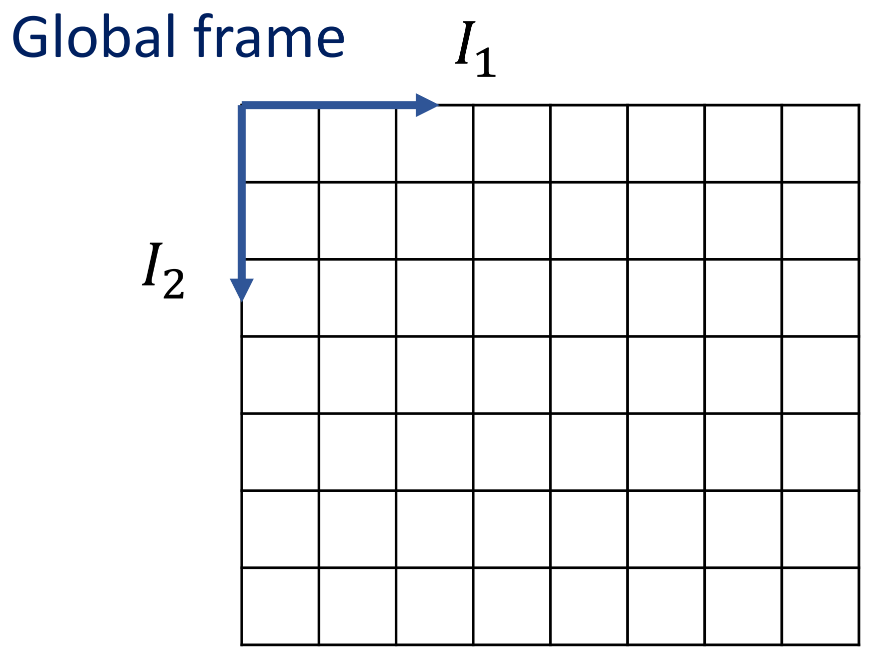


The Map

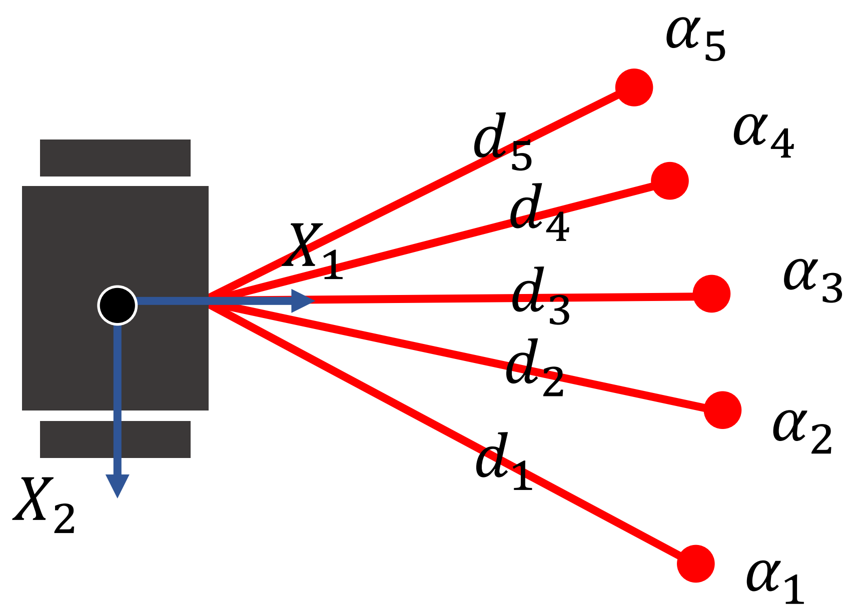
Distance measurement : d
Known state : (x_1, x_2, θ)
Occupied cell : $(x_{1,occ}, x_{2,occ})$

***Bresenham's line algorithm**

Handling Range Measurement on Grid

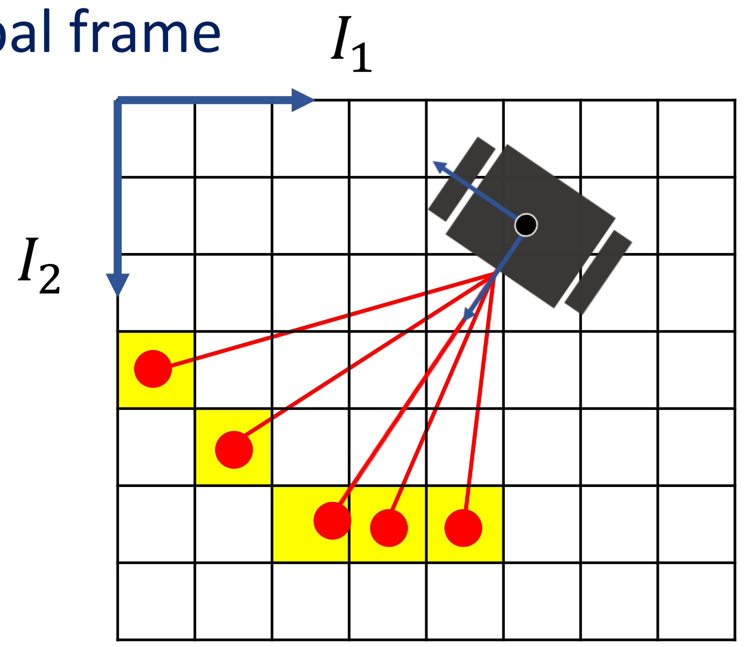


The Map



The Robot

Handling Range Measurement on Grid



The Map

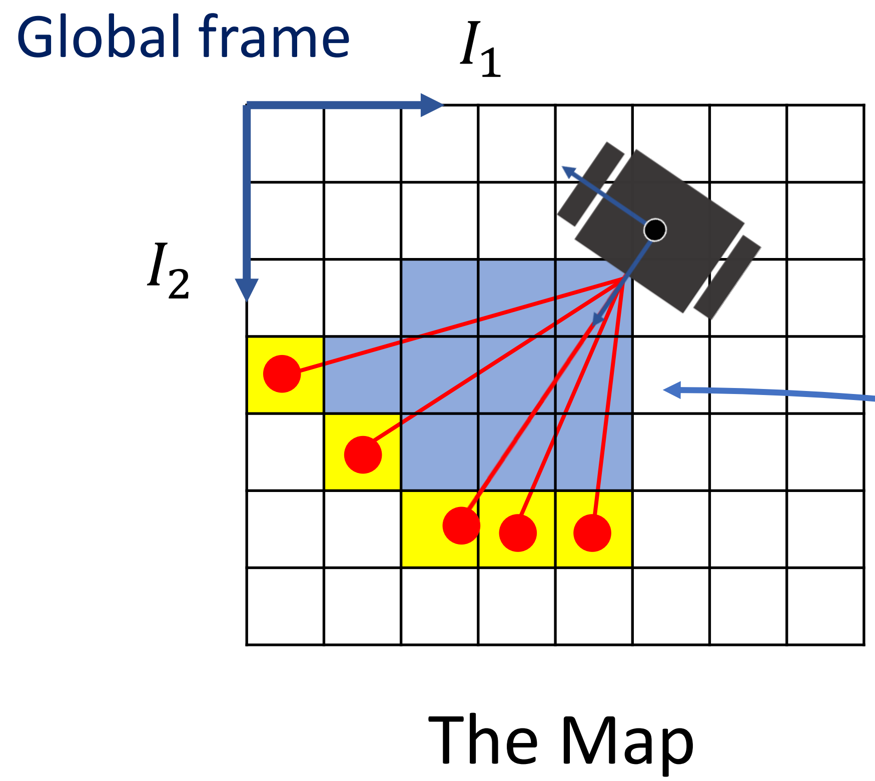
Distance measurement : d
Known state : (x_1, x_2, θ)
Occupied cell : $(x_{1,occ}, x_{2,occ})$

For k-th occupied cell:

$$\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix} = \begin{bmatrix} d_k \cos(\theta + a_k) \\ -d_k \sin(\theta + a_k) \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

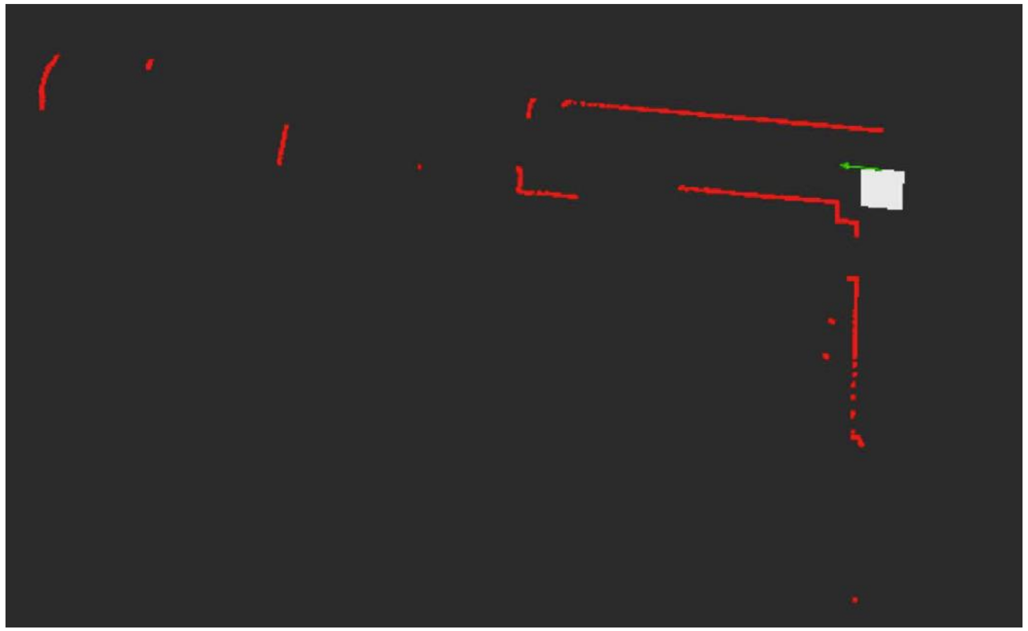
$$\begin{bmatrix} i_{1k} \\ i_{2k} \end{bmatrix} = ceil\left(\frac{1}{r} \begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix} \right)$$

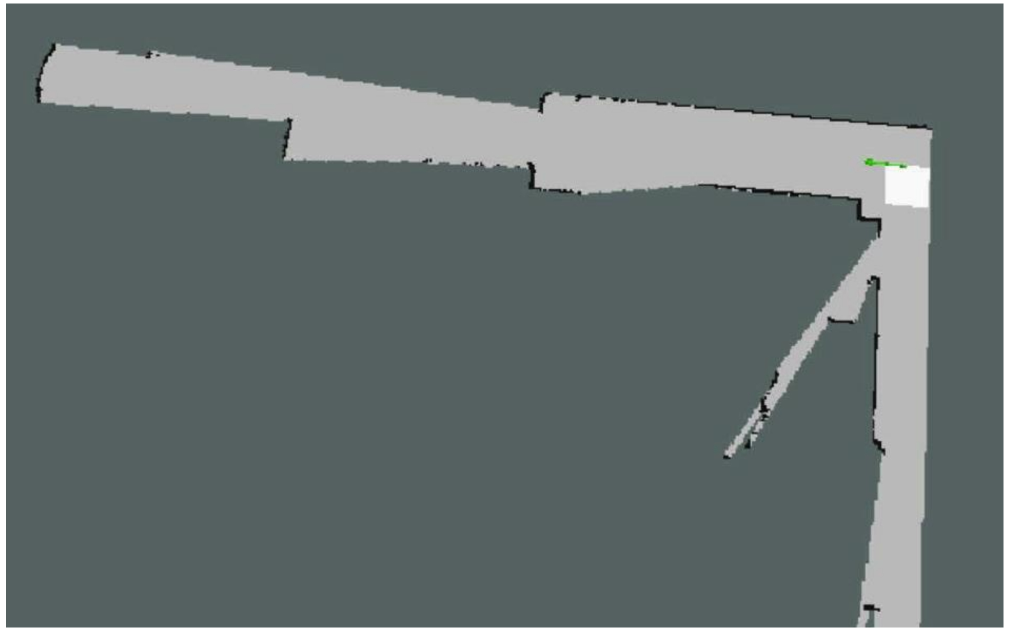
Handling Range Measurement on Grid



*Bresenham's line algorithm

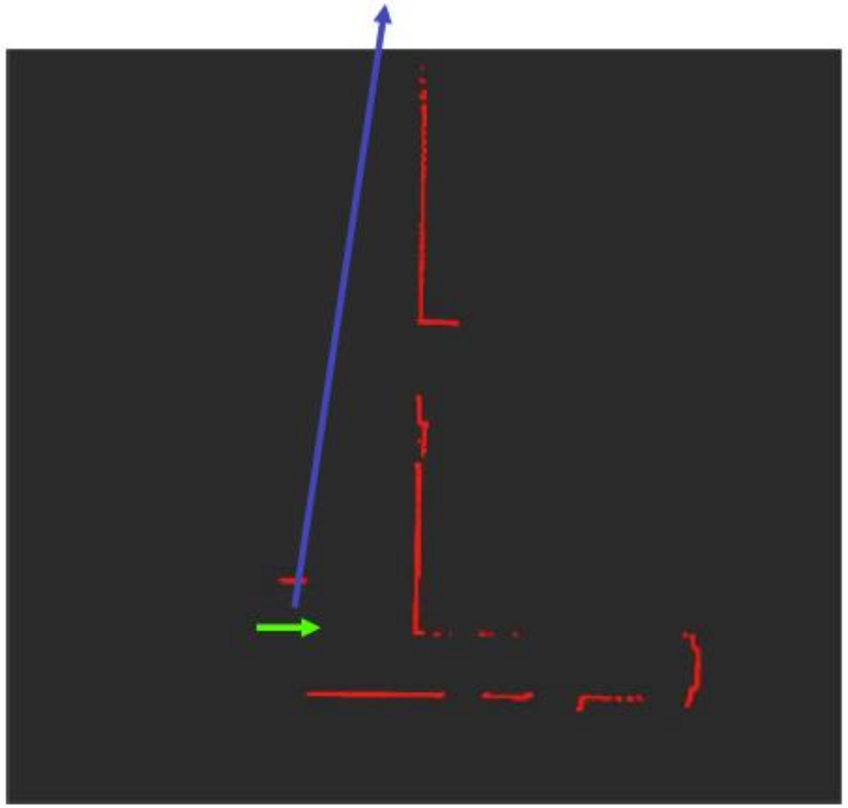
Registering the First Scan





Scan Matching: First Scan

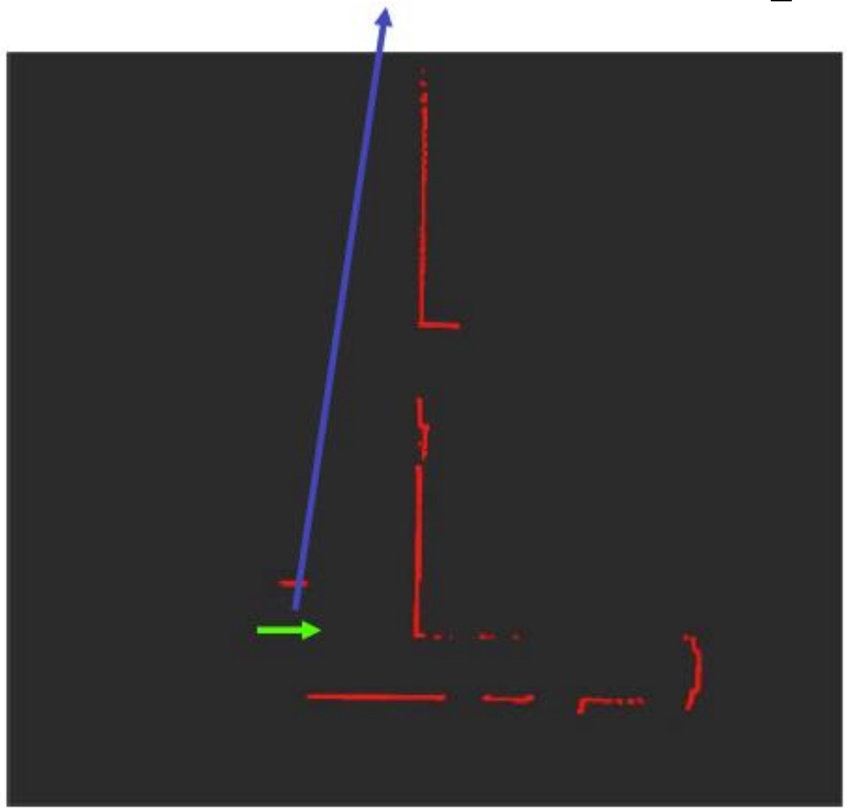
Pose of the Car at $t = t_1$



Laser Scans w.r.t. car at Time $t = t_1$

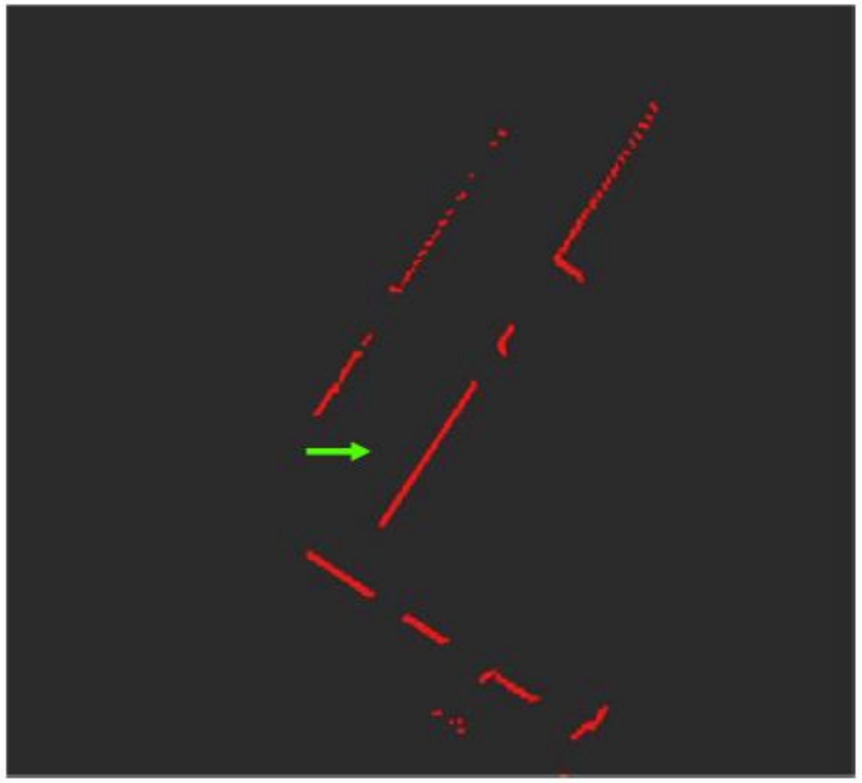
Scan Matching: Next Scan

Pose of the Car at $t = t_1$



Laser Scans w.r.t. car at Time $t = t_1$

Pose of the Car at $t = t_2$

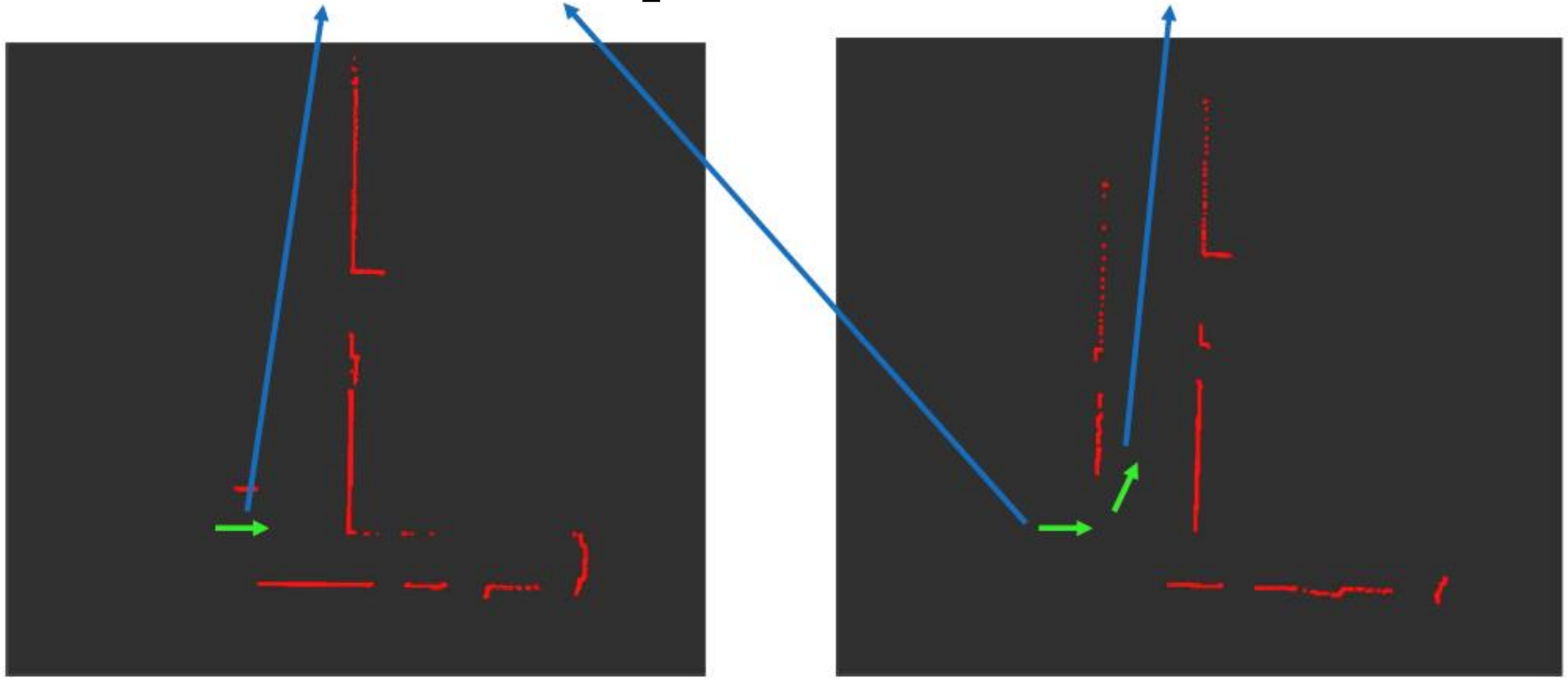


Laser Scans w.r.t. car at Time $t = t_2$

Scan Matching: Frame Alignment

Pose of the Car at $t = t_1$

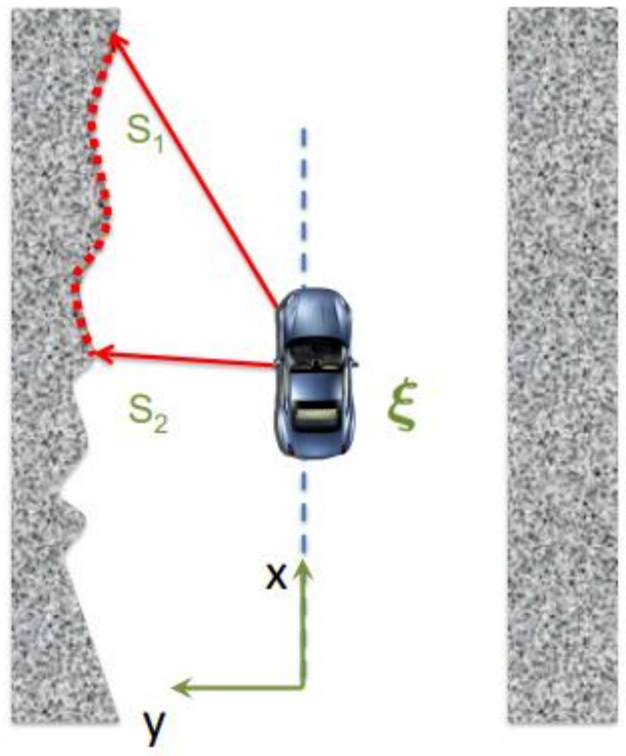
Pose of the Car at $t = t_2$



Laser Scans w.r.t. car at Time $t = t_1$

Laser Scans w.r.t. car at Time $t = t_2$

Scan Matching: Hector Slam



Robot Pose $\xi = \left(p_x, p_y, \psi\right)^T$

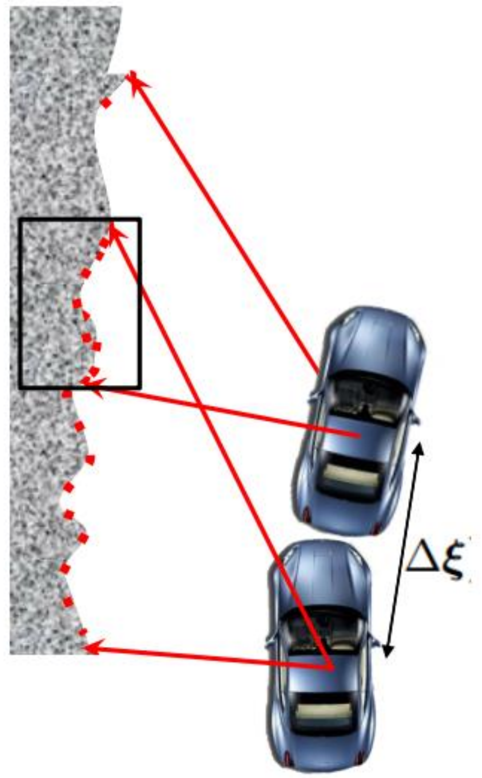
Impact coordinate of i^{th} scan in the world frame

Total of *n* scans

$$\xi^* = \operatorname{argmin}_{\xi} \sum_{i=1}^{n} \left[1 - M\left(S_i(\xi)\right)\right]^2$$

Map value at coordinates given by S_i

Scan Matching: Hector Slam



$$\sum_{i=1}^{n}\left[1 - M\left(S_i(\xi + \Delta\xi)\right)\right]^2 \to 0$$

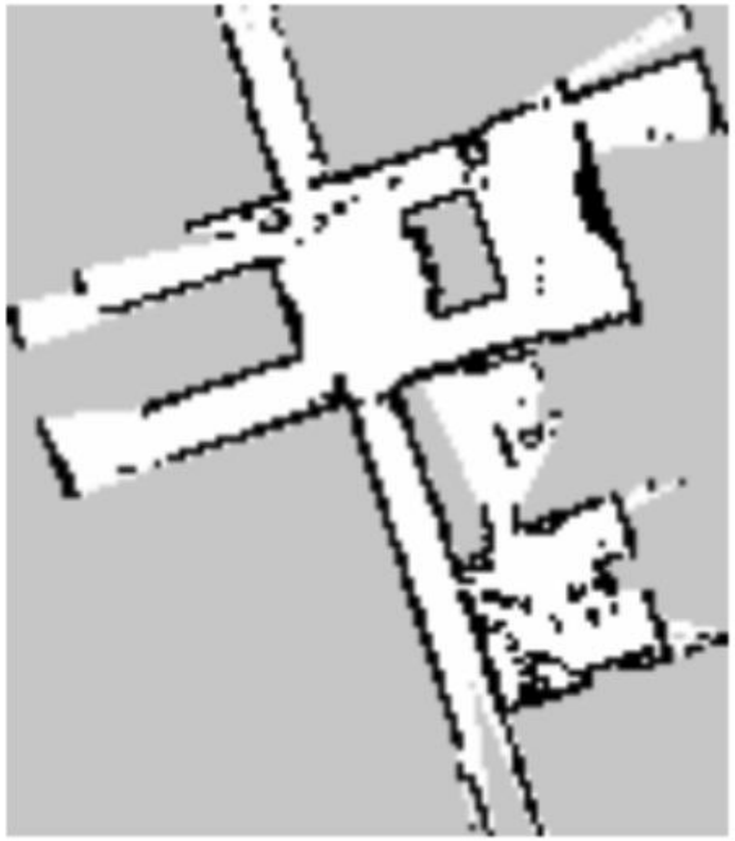
Taylor Expansion of Function M

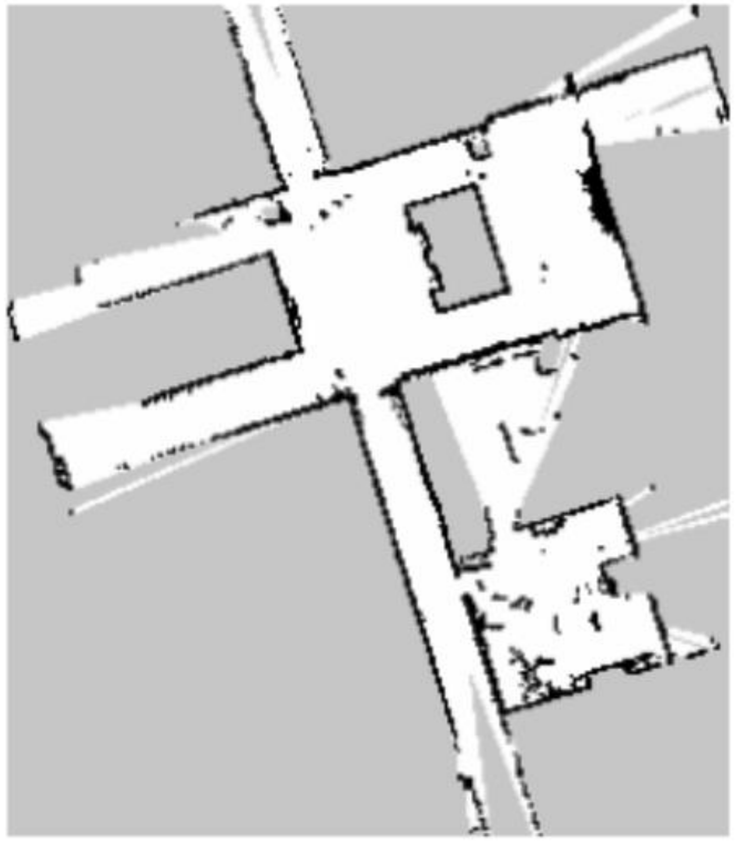
$$\sum_{i=1}^{n}\left[1 - M\left(S_i(\xi)\right) - \nabla M\left(S_i(\xi)\right)\frac{\partial S_i(\xi)}{\partial \xi}\Delta\xi\right]^2 \to 0$$

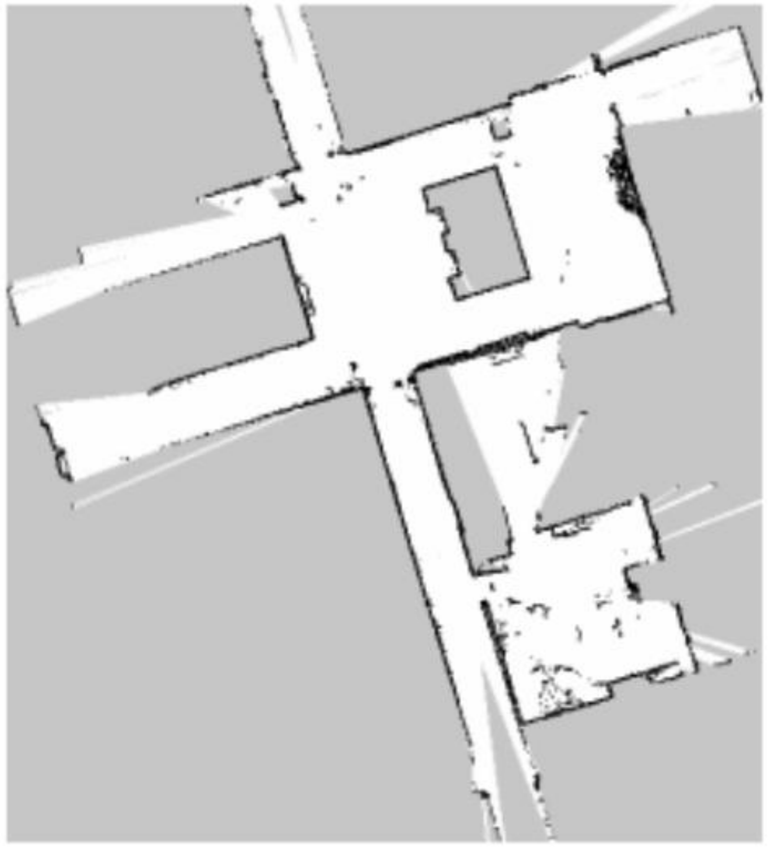
Solving for $\Delta\xi$ yields Gauss-Newton Equation

Evaluation of Gaussian-Newton equation gives a step $\Delta\xi$ that minimize the objective function

Multi-Resolution Map Representation


20 cm Grid Cell


10 cm Grid Cell


5 cm Grid Cell

State-of-the-art SLAM: Cartographer

- The major contribution of this algorithm is a novel method for ***reducing the computational requirements of computing <u>loop closure</u> constraints*** from laser range data.

Loop-closure

- Two regions in the map are found to be the same region in the world even though their position is incompatible given the uncertainty estimate in the map → the classic loop closure problem

- The system must then be able to calculate the transformation needed to align these two regions to ‘close the loop’.

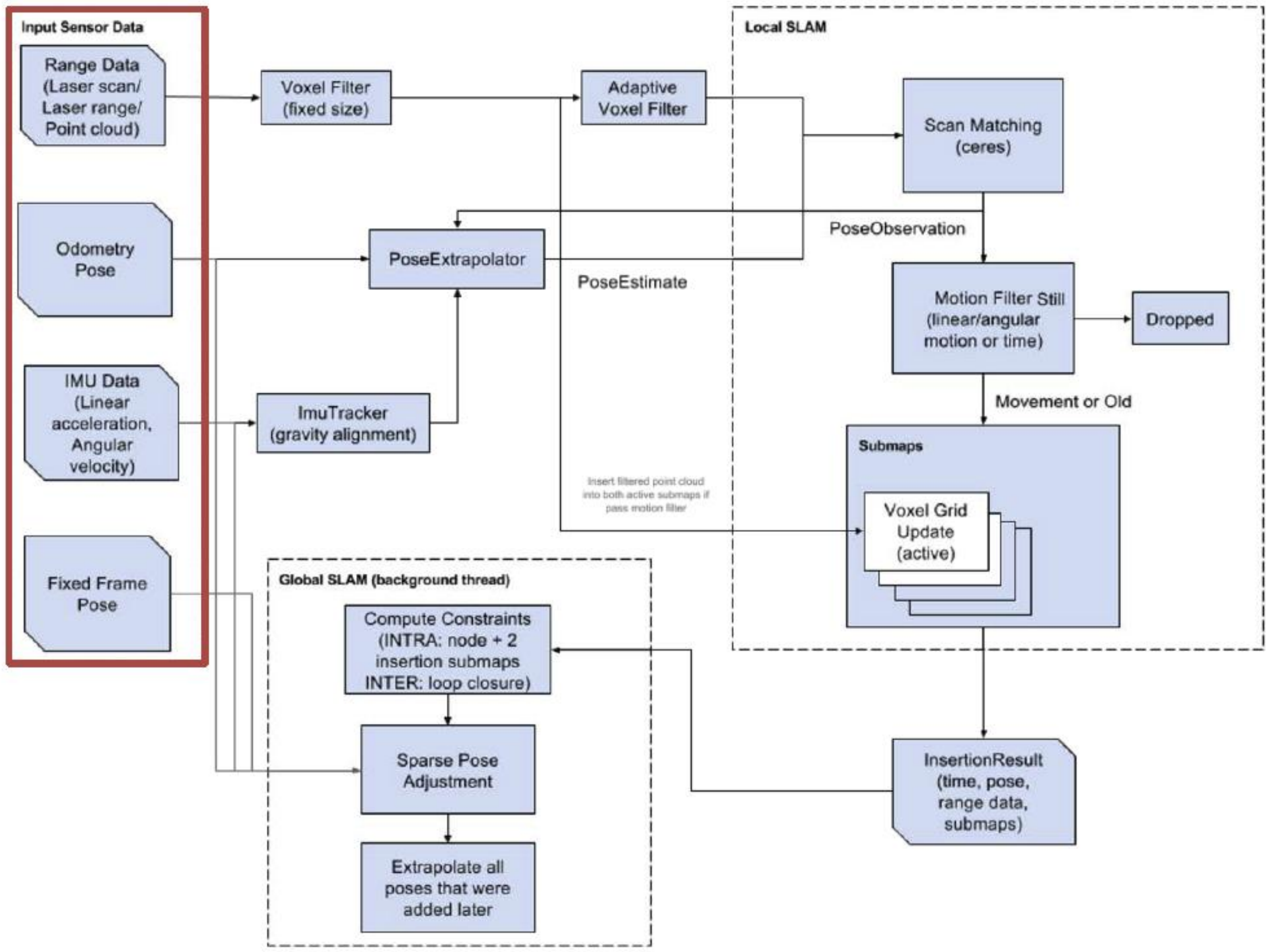
System Overview: Sensor Inputs

- Up to four types of sensor measurements:
  (1)Range (2)Odometry (3)IMU
  (4)Fixed Frame Pose

- The F1/10 vehicle supplies two sensors

2D LIDAR
(Range)

VESC
(Odometry)

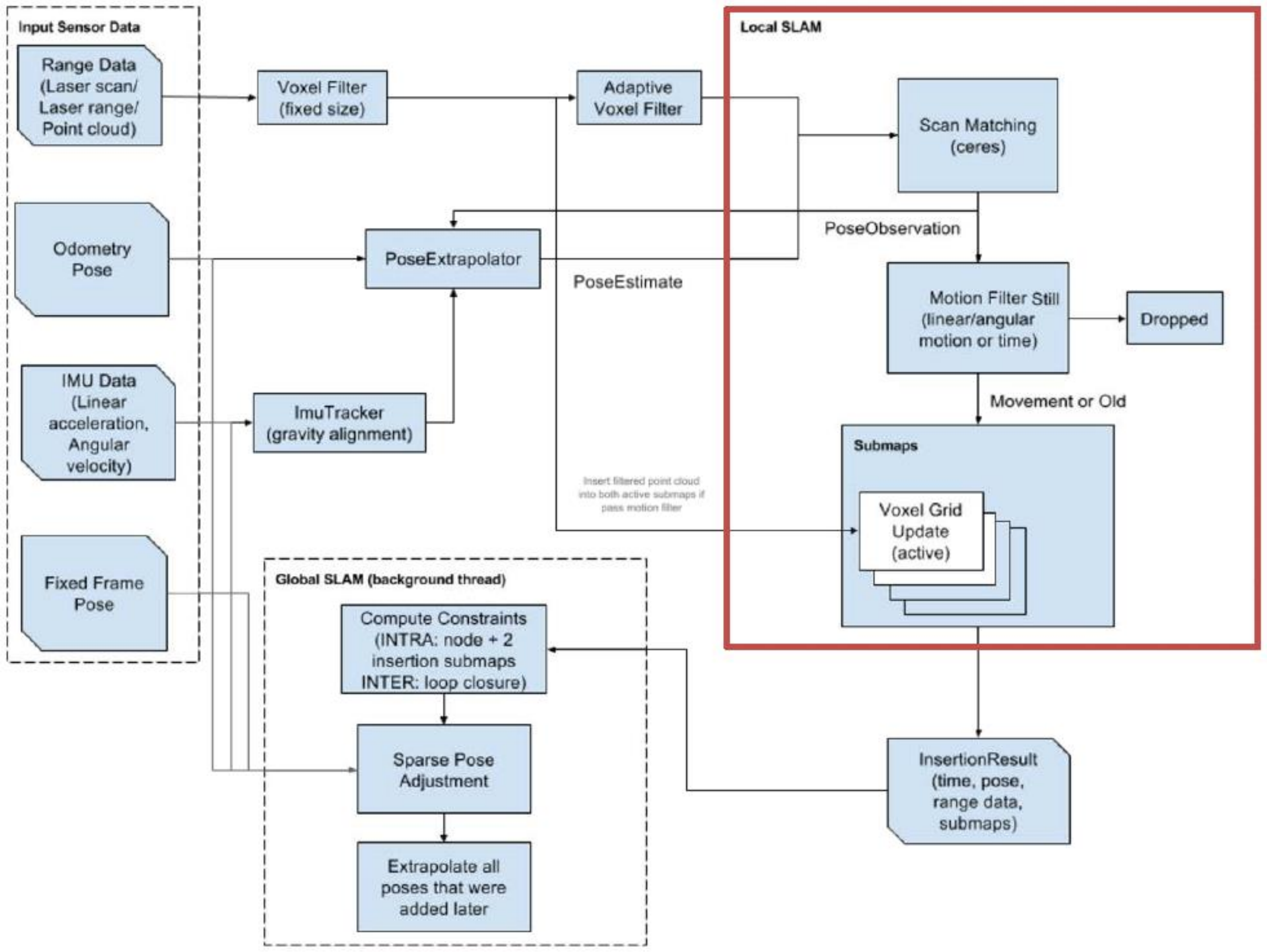

Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Insert filtered point cloud into both active submaps if pass motion filter
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
InsertionResult (time, pose, range data, submaps)
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later

System Overview: Frontend

- Cartographer consists of two separate subsystems: local SLAM(frontend or trajectory builder) and global SLAM(backend)

- **The job of local SLAM is to generate good submaps.**

- The job of global SLAM is to tie the submaps consistently together.

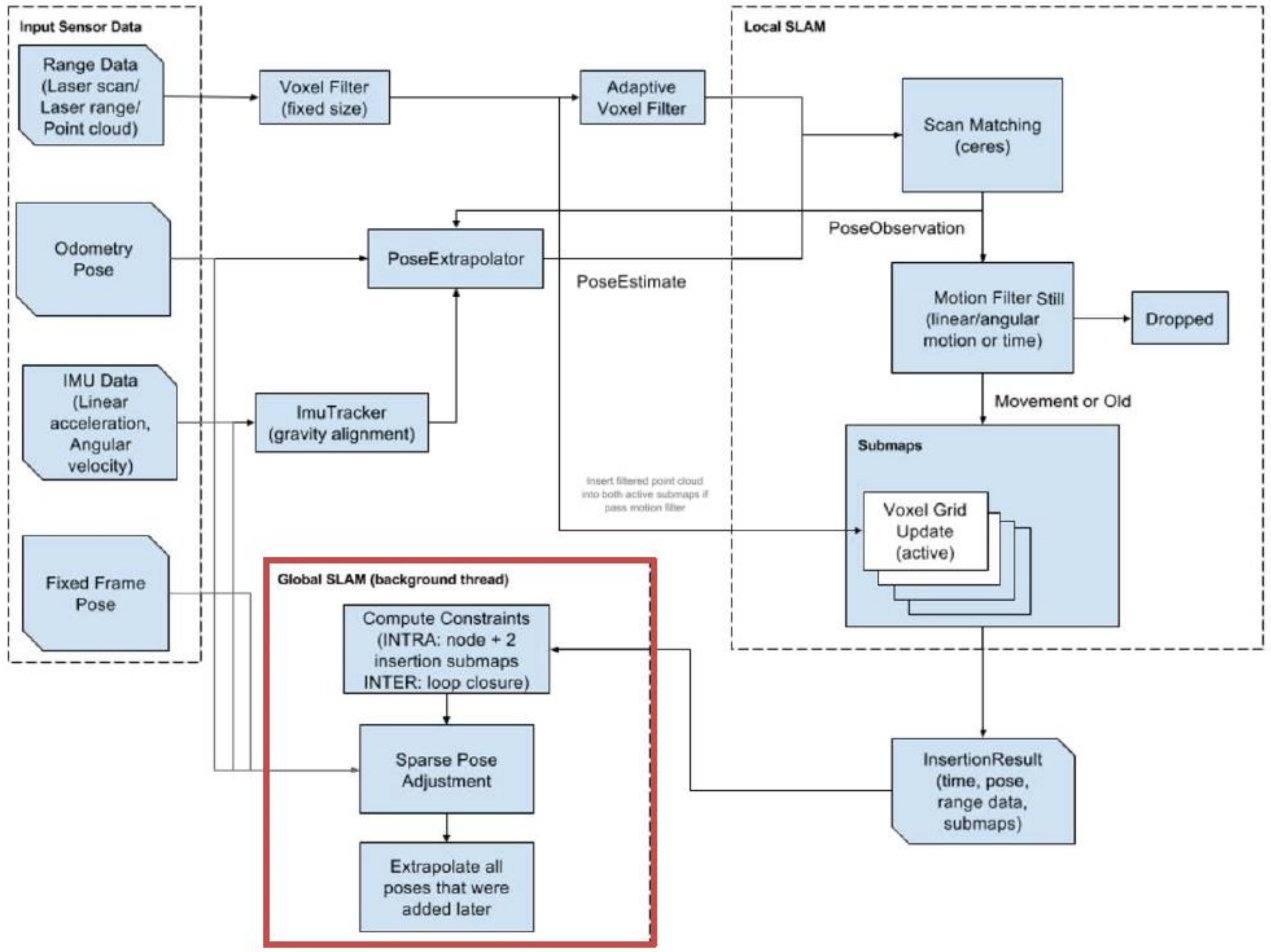

Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Insert filtered point cloud into both active submaps if pass motion filter
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)

System Overview: Backend

- Cartographer consists of two separate subsystems:
  local SLAM(frontend or trajectory builder) and global SLAM(backend)

- The job of local SLAM is to generate good submaps.

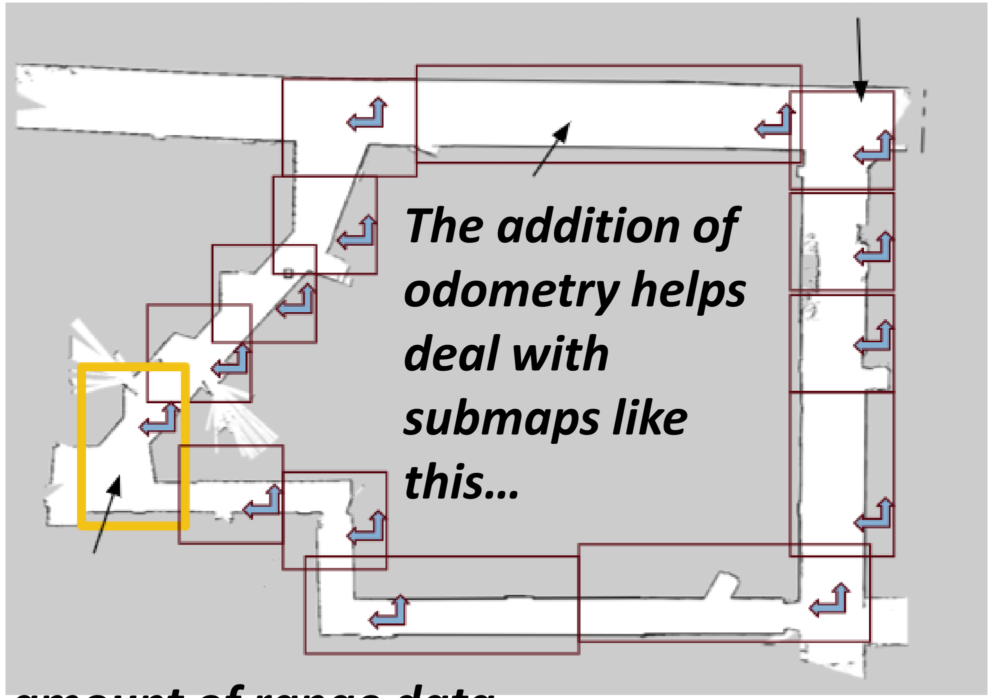
- **The job of global SLAM is to tie the submaps consistently together.**


Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Insert filtered point cloud into both active submaps if pass motion filter
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)

Frontend: Local SLAM

- What is a submap?
  - A submap is considered as complete when the local SLAM has received a given amount of range data.
  - Submaps must be small enough so that the drift inside them is below the resolution of the occupancy grid, so that they are locally correct.
  - On the other hand, they should be large enough to be distinct for loop closure to work properly. More on this later

Submaps are small chunks of the world filled with a fixed amount of registered range data

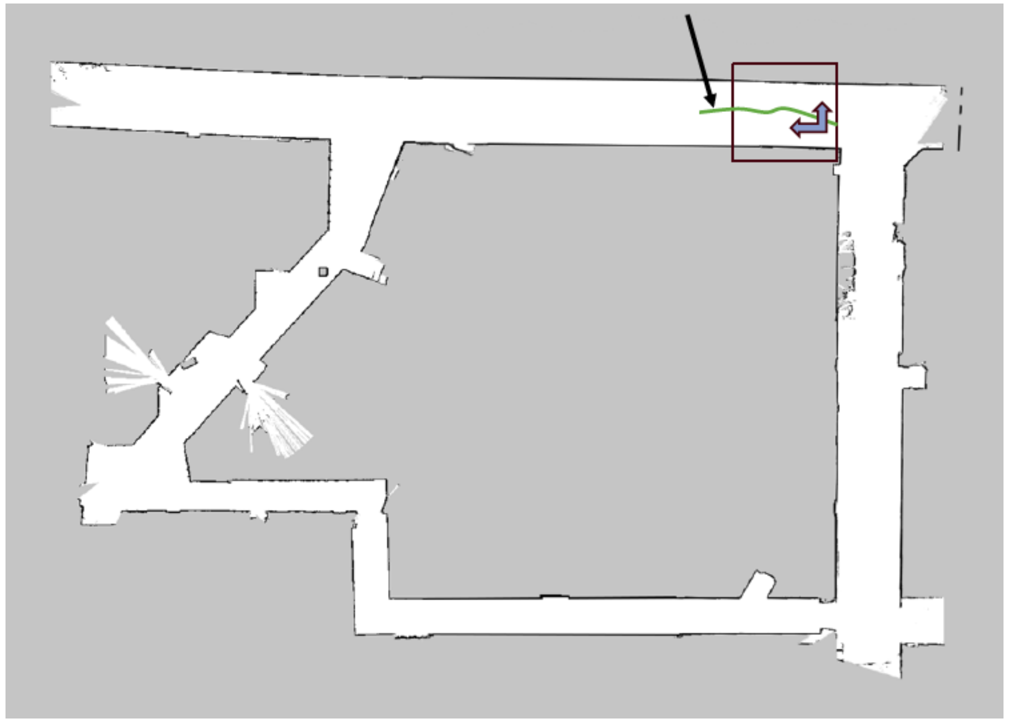


The amount of range data should be large enough that submaps are distinguishable…

Frontend: Local SLAM

- What is a submap?
  - Submaps each have their own static transform and contain a collection of registered range-measurements.
  - Consecutive measurements are connected by constraints which are 'local'.
  - Here local means derived from odometry or recent scan overlaps and resultant scan matches.

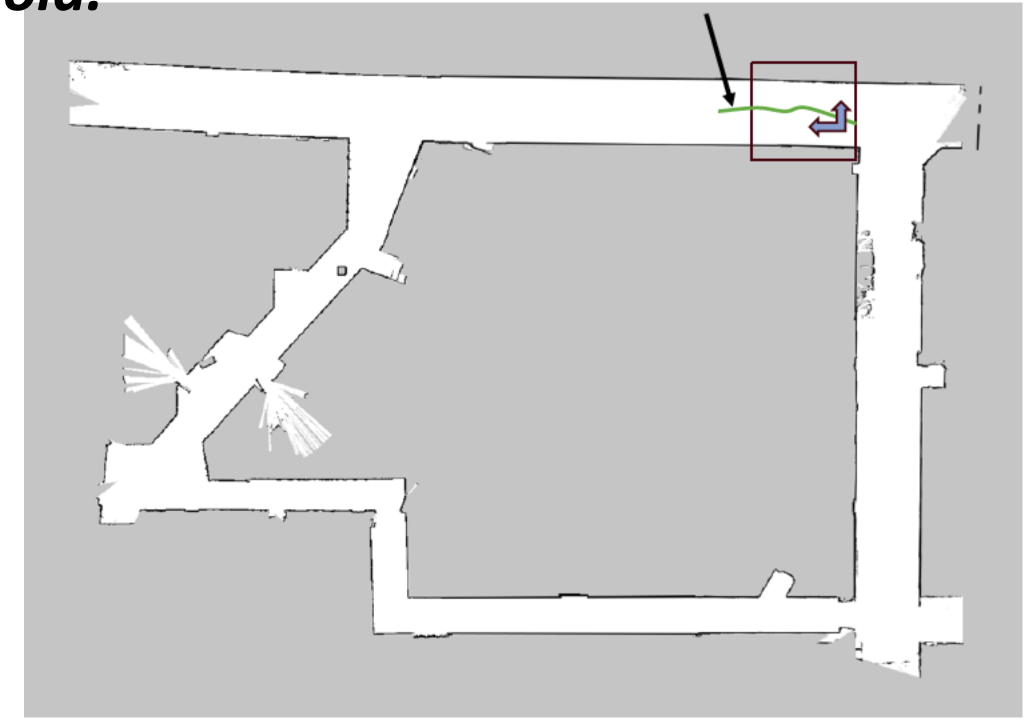
Localization: the robot's trajectory in a submap is computed via scan matching.



Frontend: Local SLAM

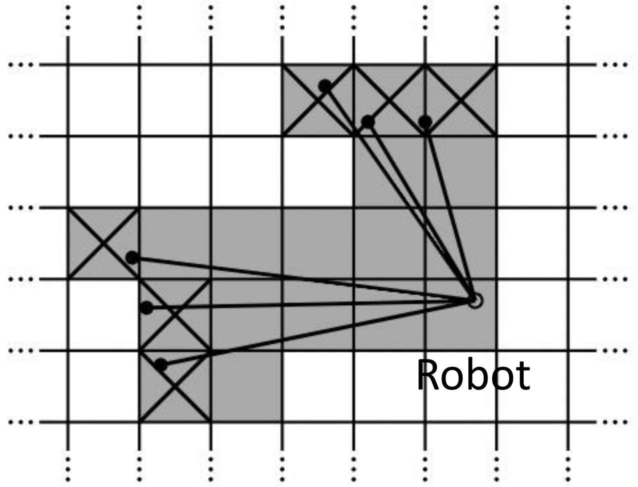
- What is a submap?
  - Submaps each have their own static transform and contain a collection of registered range-measurements.
  - Consecutive measurements are connected by constraints which are 'local'.
  - Here local means derived from odometry or recent scan overlaps and resultant scan matches.

Once a motion between two scans is found by the scan matcher, it goes through a motion filter. A scan is inserted into the current submap only if its motion is above a certain distance, angle or time threshold.



Frontend: Local SLAM

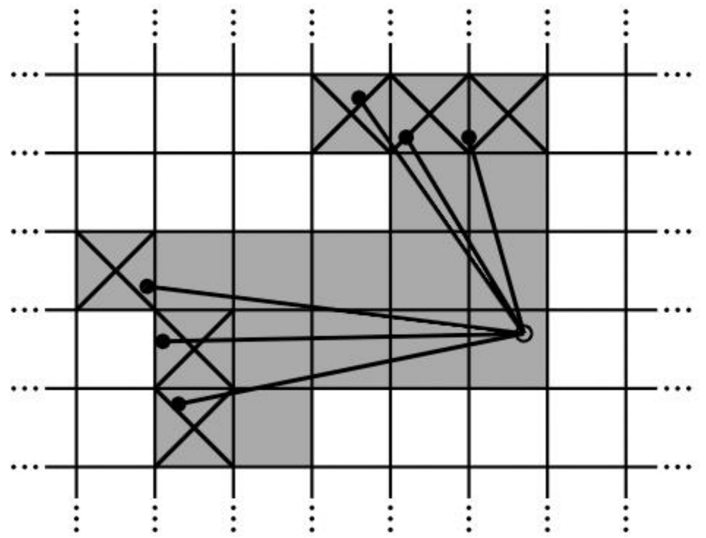
- Submap Representation
  - Submaps can **store their range data in a probability grid**. For 2D a signed distance field representation is also possible.
  - **Probability grids** are a 2D table where each **cell has a fixed size and contains the odds** of being obstructed.
  - Odds are updated according to **"hits"** (where the range data is measured) and **"misses"** (the free space between the sensor and the measured points).



"hits" : shaded and crossed out
"misses" : shaded only

Frontend: Local SLAM

- Updating the submap
  - 1. For every hit, we **insert the closest grid point** into the hit set.
  - 2. For every miss, we **insert the grid point associated with each pixel that intersects one of the rays between the scan origin and each scan point**, excluding grid points which are already in the hit set.
  - 3. If a grid point **has yet to be observed it is assigned a value p_{hit} or p_{miss}** depending on which set it is in.



Frontend: Local SLAM

- Updating the submap
  - Define the submap as:

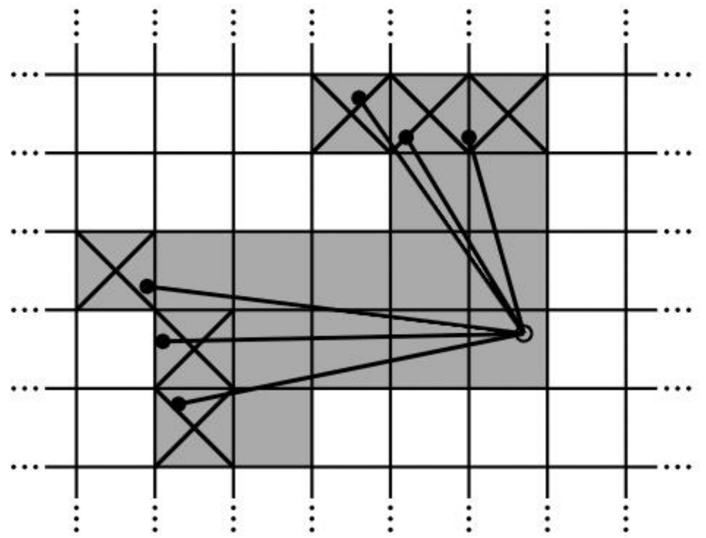
$$M : r\mathbf{Z} \times r\mathbf{Z} \rightarrow [p_{min}, p_{max}]$$

  - Then the map is updated according to the following operation:

$$M_{new}(x) = \boldsymbol{clamp}\left(odds^{-1}\left(odds\left(M_{old}(x)\right) odds(p_{hit}) \right)\right)$$

  - Recall the definition of the odds function:

$$odds(p) = \frac{p}{1-p}$$

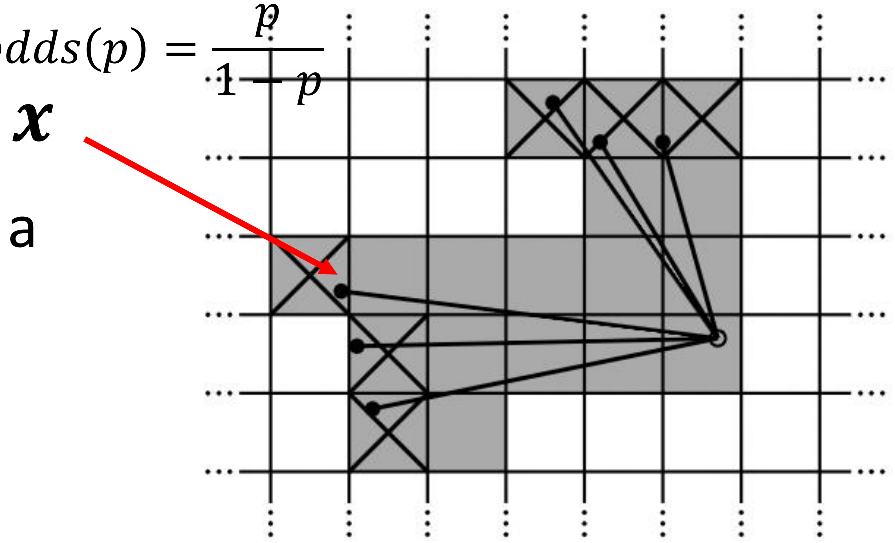


Frontend: Local SLAM

$$M : r\mathbf{Z} \times r\mathbf{Z} \rightarrow [p_{min}, p_{max}]$$

$$M_{new}(x) = \boldsymbol{clamp}\,(\,odds^{-1}(\,odds(M_{old}(x))\,odds(p_{hit})\,))$$

$$odds(p) = \frac{p}{1-p}$$



- Example
  - 1. Add x to the hit set
  - 2. The grid point has yet to be observed it is assigned a value $p_{hit} = 0.67$
  - 3. $M_{old}(x) = 0.67$
  - 4. Next scan: x is hit again
  - 5. $odds(p_{hit}) = 2.03 = odds(M_{old})$
  - 6. $M_{new}(x) = odds^{-1}(2.03^2)$ therefore $M_{new}(x) = 0.80$

- **The more times you observe the point in the same set the more certain the map is that it is occupied**

Now, back to Scan Matching

- How do we know that ‘hits’ and ‘misses’ map to a particular cell in the probability grid?
  - The collection of scan points relative to a moving reference frame attached to the robot are, $H = \{h_k\}$ for $k = I \dots . K$ where h_k is a point in R^2.
  - These points are placed in a submap at pose ξ in the global reference frame by applying a rigid body transform to each (rotation and translation).
  - Submap construction is the iterative process of repeatedly aligning scan and submap coordinate frames.

Does Cartographer use ICP?

- No. Cartographer needs to support 3D LIDARs and the correspondence problem is not well solved for ICP.
- Scan matching variants:
  - Iterative closest point (ICP)
  - Scan-to-scan
  - Scan-to-map
  - Map-to-map
  - Feature-based
  - RANSAC for outlier rejection
  - Correlative matching
- **Cartographer uses the Ceres-solver to formulate a nonlinear least-squares correlative scan matching problem.**

Correlation-based Scan Matching

- In PL-ICP and ICP, correspondences between two scans are explicitly computed, allowing a rigid-body transformation to be computed.
- Susceptible to local minima; poor initial estimates lead to incorrect data associations and divergence.
- **Correlation based methods search for a rigid-body transformation (without computing correspondences) that projects one set of LIDAR points (the query scan) on top of a reference map.**
- The reference map is generally implemented as a look-up table that assigns a cost for every projected query point.

Correlation-based Scan Matching

Sum over each scan point

Transform scan point from scan frame to local submap frame

Coordinate of scan return

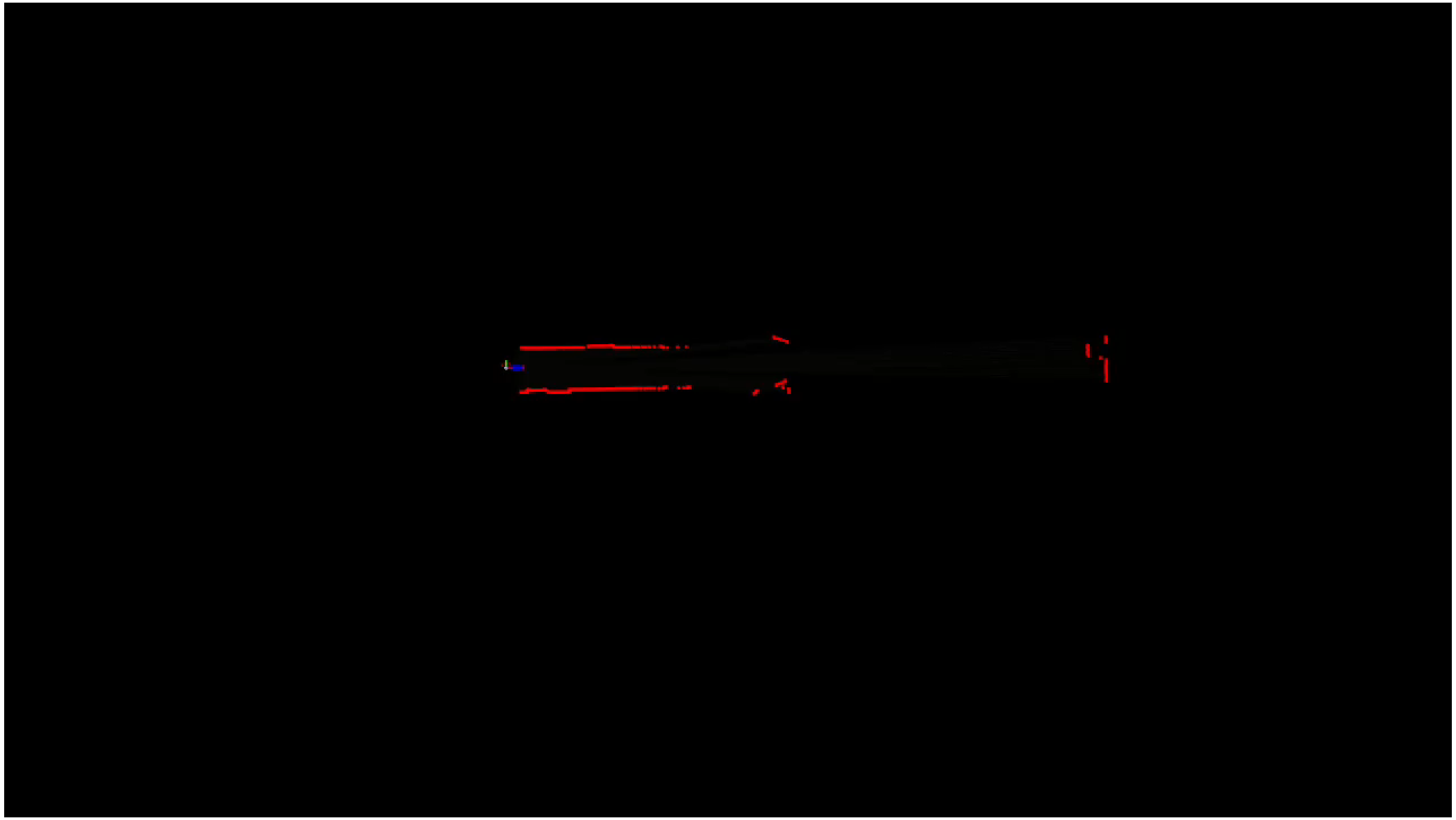
$$argmin_{\xi} \sum_{k=1}^{K} \left(1 - M_{smooth}\left(T_{\xi} h_k\right)\right)^2$$

Pose in local submap reference frame

Bicubic interpolation of probability grid

Cartographer

Video link : https://drive.google.com/file/d/1nkw8ICFoNe1U44aPM9KJsYYr3HRff6N6/view



Backend: Introduction

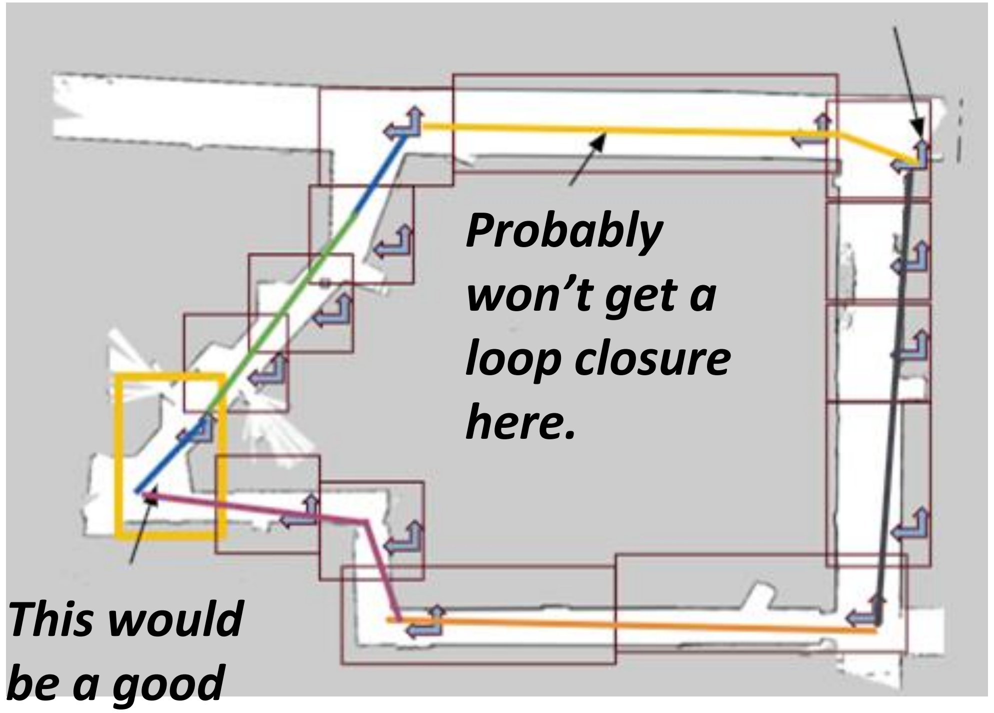
- So we can understand why scan matching alone (front-end) is insufficient for robust localization and mapping.

- That's why we need **Backend** to minimize **accumulated errors.**

Backend: Global SLAM

- Closing Loops
  - Goal: is our current scan in one of the submaps we have already seen.
  - Constraints take the form of relative pose ξ_{ij} and associated covariance matrices Σ_{ij}. **Relative poses now include both submap and scan poses.**
  - For a pair of submap i and scan j, the pose ξ_{ij} describes where in the submap coordinate frame the scan was matched.

Pose of the submap is now a decision variable.



Probably won't get a loop closure here.

This would be a good candidate for loop closure.

Backend: Global SLAM

- Loop Closure

Sum over each scan, submap combination

Residual, see paper

Covariance matrix

$$argmin_{\Xi_m,\Xi_s} \frac{1}{2}\sum_{i,j} \rho\left(E^2\left(\xi_i^m, \xi_j^s; \Sigma_{i,j}, \xi_{ij}\right)\right)$$

Submap poses

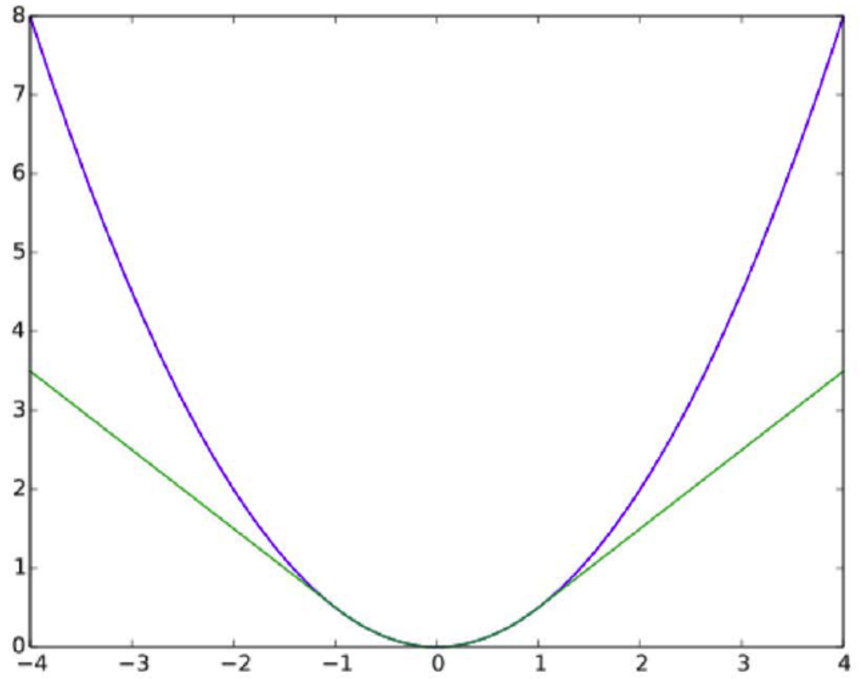
Scan poses

Huber loss, outlier rejection

Relative pose between submap and scan(constraint)

Backend: Global SLAM

- Huber Loss
  - Uses a Huber loss to make the objective less susceptible to spurious constraints which are often added in symmetric environments… False positive is a disaster



$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & for\ |a| \le \delta, \\ \delta\left(|a| - \frac{1}{2}\delta\right), & otherwise. \end{cases}$$

Using Hector SLAM

```
# Install ‘hector SLAM’
sudo apt-get update
sudo apt-get install ros-melodic-hector-map*

# Edit bashrc
alias hector=‘rosrun hector_mapping hector mapping’
alias mapsave=‘rosrun map_server map_saver –f map_data’
source ~/.bashrc

# Run ‘hector SLAM’
hector
mapsave
```

Using Cartographer

```
# Install wstool and rosdep.
sudo apt-get update
sudo apt-get install -y python3-wstool python3-rosdep ninja-build stow

# Create a new workspace in ‘catkin_ws’
mkdir catkin_ws
cd catkin_ws
wstool init src

# Merge the cartographer_ros.rosinstall file and fetch code for dependencies.
wstool merge -t src https://raw.githubusercontent.com/cartographerproject/cartographer_ros/master/cartographer_ros.rosinstall
wstool update -t src

# Install dependencies.
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=melodic –y

# Install abseii-cpp and delete conflicting version.
src/cartographer/scripts/install_abseil.sh
sudo apt-get remove ros-melodic-abseil-cpp

# Build and install.
catkin_make_isolated --install --use-ninja
```

Particle Filter

Uncertainty in Robotics

- 1. Sensors
  - Limited in what they can perceive
  - Range and resolution
  - Noise
  - Faulty sensor
- 2. Robot actuation (Motor)
  - Control noise
  - Wear and tear
  - Mechanical failure
- 3. Algorithmic approximation
  - Robots are real-time system
  - Sacrifice accuracy (approximation) to achieve timely response

Probabilistic Robotics

- Managing **uncertainty** is possibly the most important step towards **robust real-world robot systems.**

- Probabilistic robotics is a relatively new approach to robotics that pays tribute to the uncertainty in robot perception and action.

- The key idea in probabilistic robotics is to represent uncertainty explicitly using the calculus of **probability**.

- <u>**Probabilistic algorithms** represent information by **probability distributions** over a whole space of guesses.</u>

Basic Concepts in Probability

- $p(X = x). \rightarrow disrete\ case\ and\ continuous\ case$
- $\sum_x p(X = x) = 1$
- $p(x, y) = p(X = x\ and\ Y = y) \rightarrow joint\ distribution$
- If they are independent $p(x, y) = p(x)p(y)$
- $p(x|y) = \frac{p(x,y)}{p(y)} \rightarrow conditional\ probability \rightarrow show\ independence\ case$
- Law of total probability $p(x) = \sum_y p(x|y)p(y)$
- Bayes rule(Bayes theorem / Formula)
- Expectation $E[X] = \sum_x x\, p(x) :\ same\ with\ average$
- Normalization

Basic Concepts in Probability

- $\boldsymbol{p(X = x)}$: Denote the probability that the **random variable** $\boldsymbol{X}$ has **value** $\boldsymbol{x}$

Discrete case

$$\sum_{x} p(X = x) = 1$$

Continuous case

$$\int p(x)\, dx = 1$$

- $\boldsymbol{p(x, y) = p(X = x \text{ and } Y = y)}$: **Joint distribution**
  - This expression describes the probability of the event that the random variable X takes on the value x and that Y takes on the value y.
  - If X and Y are *independent*, $\boldsymbol{p(x, y) = p(x)p(y)}$

Marginalization

Discrete case

$$p(x) = \sum_{y} p(x, y)$$

Continuous case

$$p(x) = \int p(x, y)\, dy$$

Basic Concepts in Probability

- $\boldsymbol{p(x|y) = p(X = x|Y = y)}$: **Conditional probability**
  - Probability that $X's$ value is x when we already know that $Y's$ value is y .
  - If $p(y) > 0$, then the conditional probability is defined as $$\boldsymbol{p(x|y) = \frac{p(x,y)}{p(y)}}$$
  - If X and Y are independent, $$\boldsymbol{p(x|y) = \frac{p(x)p(y)}{p(y)} = p(x)}$$
  - In other words, if X and Y are independent, Y tells us nothing about the value of X.

Law of Total Probability

Discrete case

$$\boldsymbol{p(x) = \sum_{y} p(x|y)p(y)}$$

Continuous case

$$\boldsymbol{p(x) = \int p(x|y)\, p(y)dy}$$

Basic Concepts in Probability

- Bayes Rule

$$p(x,y) = p(x|y)p(y) = p(y|x)p(x)$$

$$\boxed{p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{likelihood \cdot prior}{evidence}}$$

Normalization

$$= \eta\, p(y|x)p(x)$$

$$\eta = p(y)^{-1} = \frac{1}{\sum_x p(y|x)p(x)}$$

- Bayes Rule with Background Knowledge

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)}$$

Basic Concepts in Probability

- **Recursive Bayesian Updating**
  - $p(x|z_1, \dots, z_n) = \dfrac{p(z_n|x, z_1, \dots, z_{n-1})p(x|z_1, \dots, z_{n-1})}{p(z_n|z_1, \dots, z_{n-1})}$

- **Markov assumption**:
  - z_n is *independent* of $z_1, \dots, z_{n-1}$, if we know x
  - In other word it postulates that past and future data are independent if one knows the current state.
  - $p(x|z_1, \dots, z_n) = \dfrac{p(z_n|x)p(x|z_1, \dots, z_{n-1})}{p(z_n|z_1, \dots, z_{n-1})} = \eta\, p(z_n|x)p(x|z_1, \dots, z_{n-1})$

$$= \eta_{1\dots n} \prod_{i=1\dots n} p(z_i|x)p(x)$$

Bayes Filter

- Given:
  - Stream of observations z and action data u:

$$d_t = \{u_1, z_1, \dots, u_t, z_t\}$$

  - Sensor model $p(z|x)$
  - Action model $p(x|u, x')$
  - Prior probability of the system state $p(x)$
- Wanted:
  - Estimate of the state X of a dynamical system
  - The posterior of the state is also called **Belief**:

$$\boldsymbol{Bel(x_t) = p(x_t|u_1, z_1, \dots, u_t, z_t)}$$

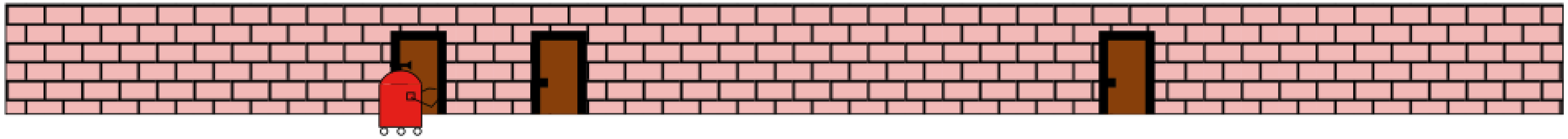
Probabilistic Robotics - Example

- Example : Mobile Robot Localization

- Robot localization is the problem of estimating a robot’s coordinates relative to an external reference frame.

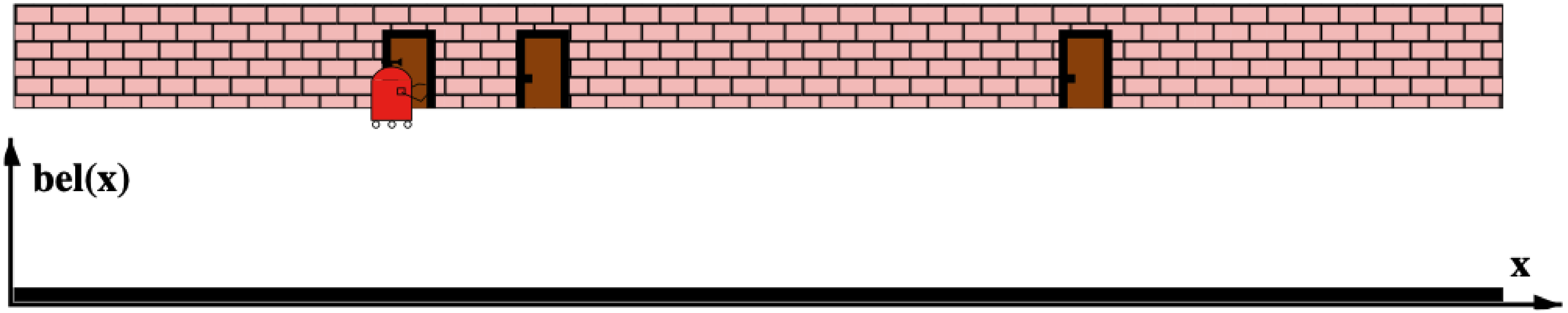
- The robot is given a map of its environment, but to localize itself relative to this map it needs to consult its sensor data

Probabilistic Robotics - Example



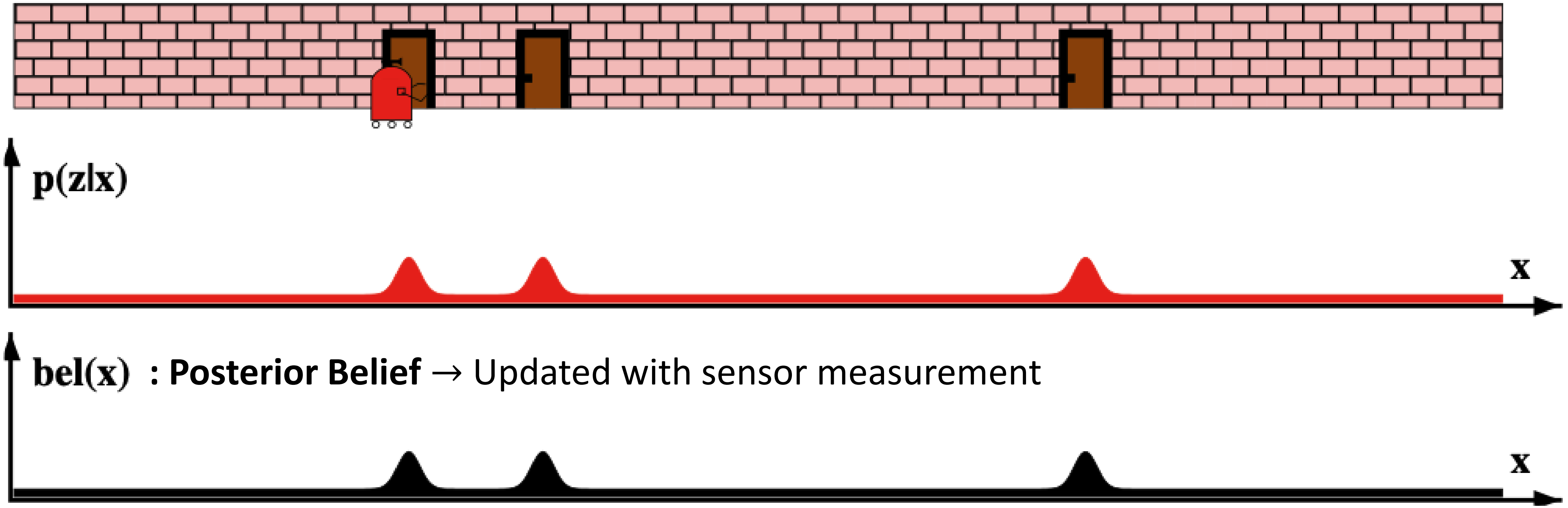
- There are three indistinguishable doors.
- The task of the robot is to find out where it is, through sensing and motion.
- This specific localization problem is known as **global localization**
- In global localization, a robot is placed somewhere in a known environment and has to localize from scratch.
- The probabilistic paradigm represents the robot's momentary **belief** by a probability density function over the space of all locations.

Probabilistic Robotics - Example



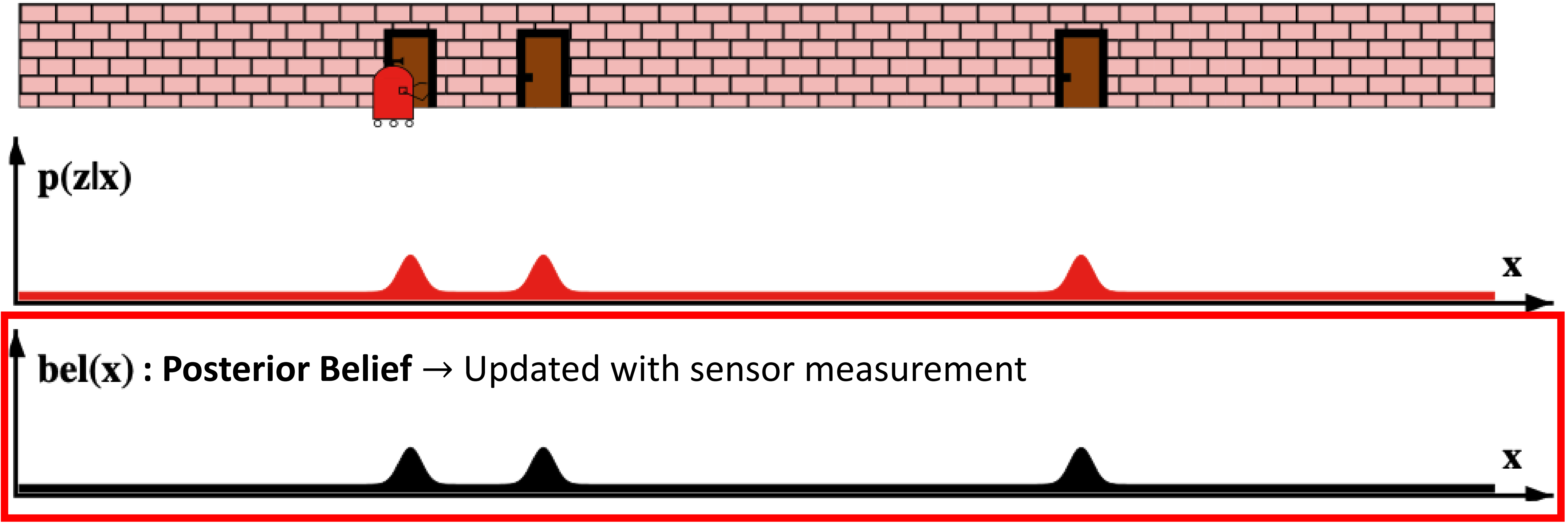
- This diagram shows a uniform distribution over all locations.
- Now suppose the robot takes a first sensor measurement and observes that it is next to a door

Probabilistic Robotics - Example



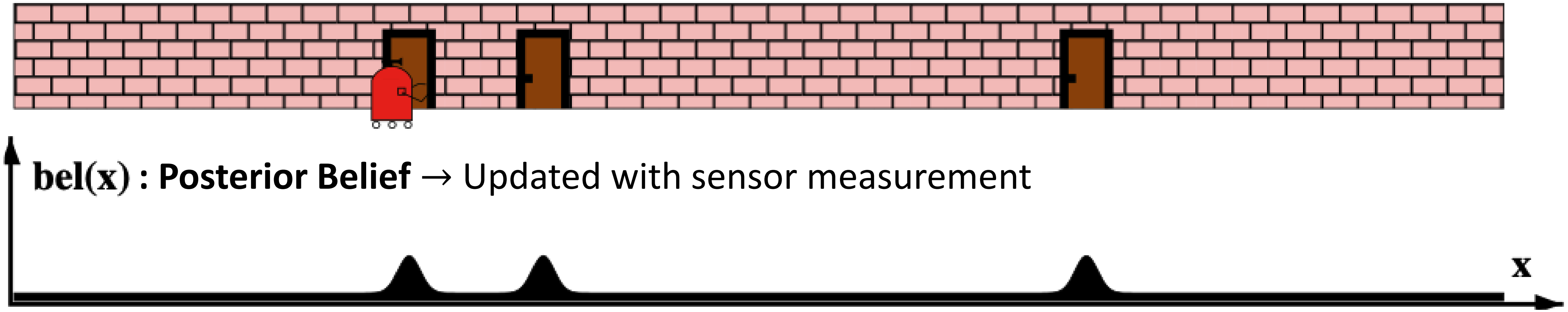
: Posterior Belief → Updated with sensor measurement

Probabilistic Robotics - Example



- It places an increased probability at places near doors, and lower probability near walls.

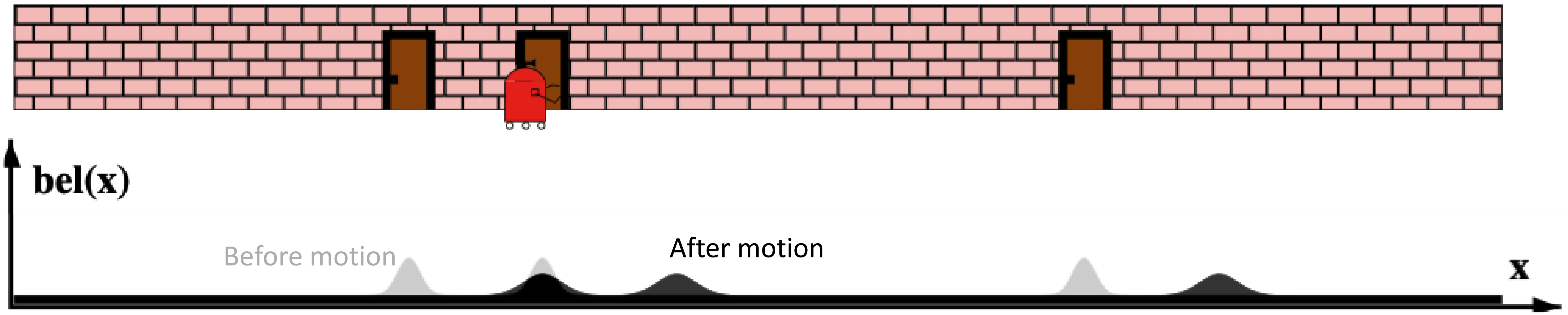
Probabilistic Robotics - Example



- Notice that this distribution possesses three peaks, each corresponding to one of the indistinguishable doors in the environment.
- Thus, by no means does the robot know where it is.
- Instead, it now has three, distinct hypotheses which are each equally plausible given the sensor data.
- Also note that the robot assigns positive probability to places not next to a door.

Probabilistic Robotics - Example

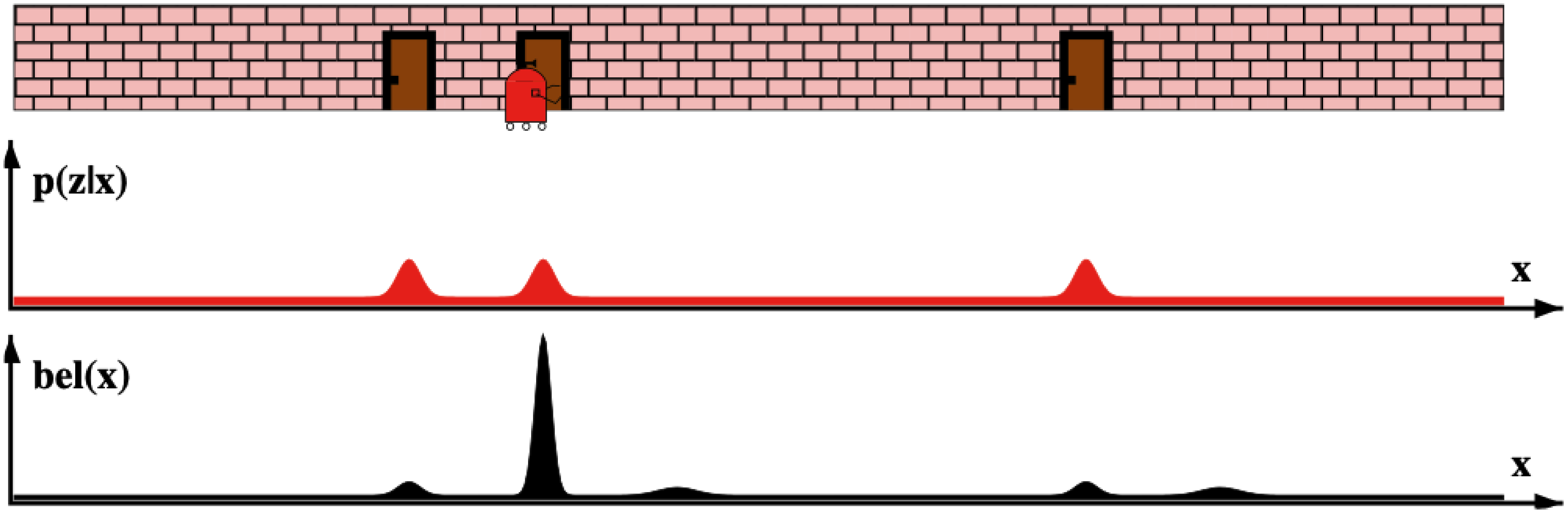
- Now suppose the robot moves.



- The belief has been shifted in the direction of motion.
- It also possesses a larger spread, which reflects the uncertainty that is introduced by robot motion.

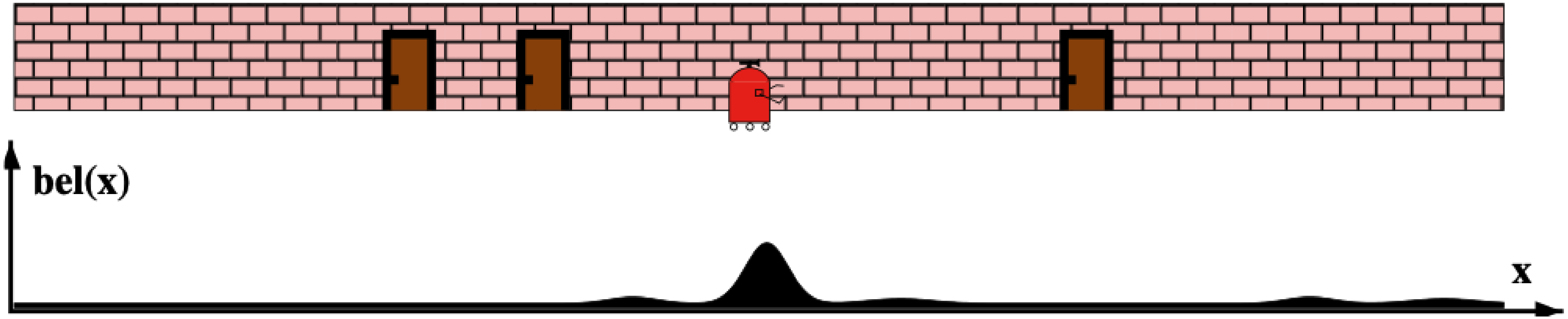
Probabilistic Robotics - Example

- Belief after observing another door.



Probabilistic Robotics - Example

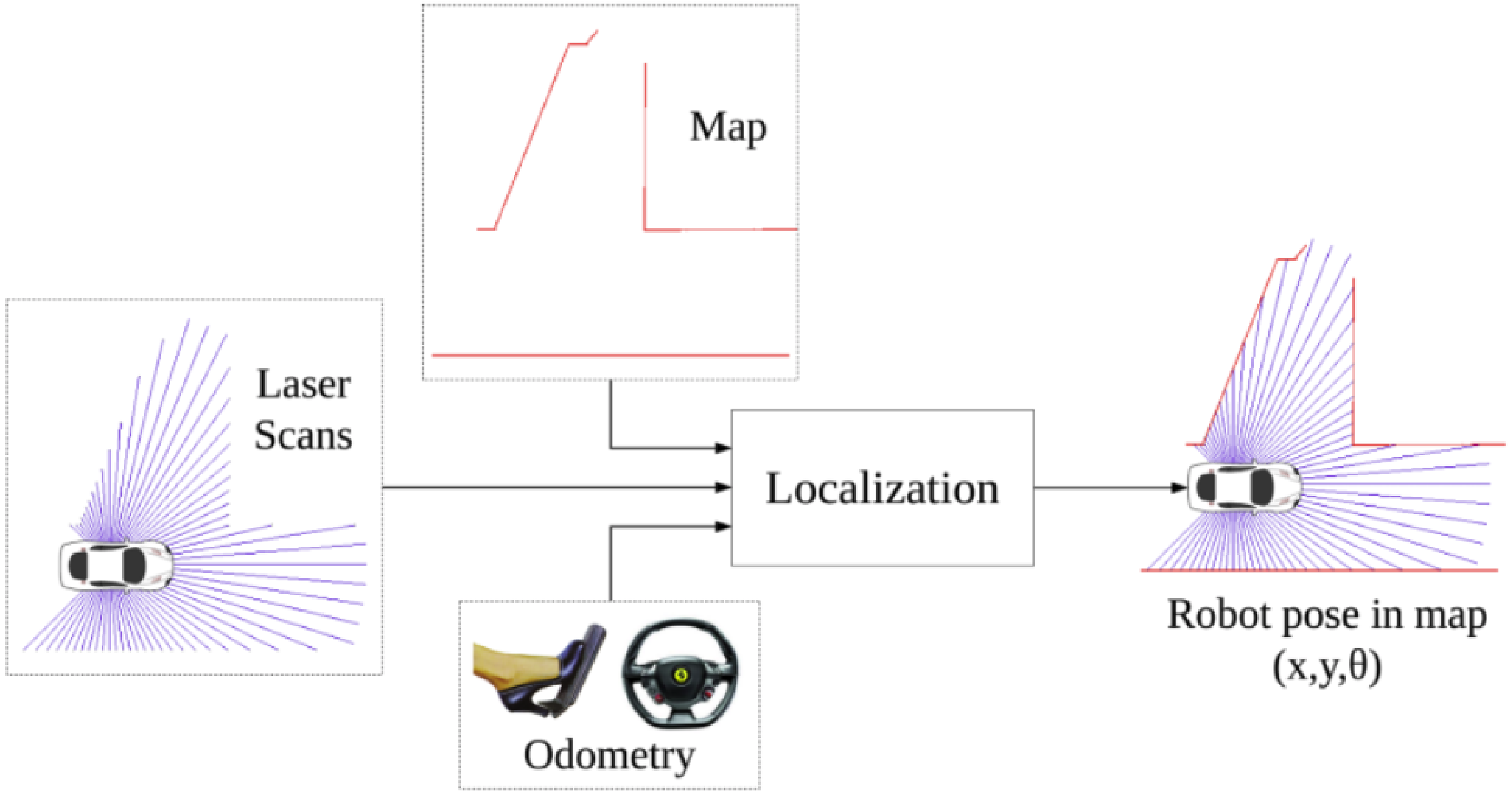
- Belief as robot travels further down the corridor.



Probabilistic Robotics - Example

- This example illustrates many aspects of the probabilistic paradigm.
- The robot perception problem is a **state estimation** problem.
- This localization example uses an algorithm known as Bayes filter for posterior estimation over the space of robot locations.
- The representation of information is a probability density function.
- The **updates** of this function represents the **information gained through sensor measurements**, or the **information lost through processes in the world that increase a robot's uncertainty.**

Problem Setting

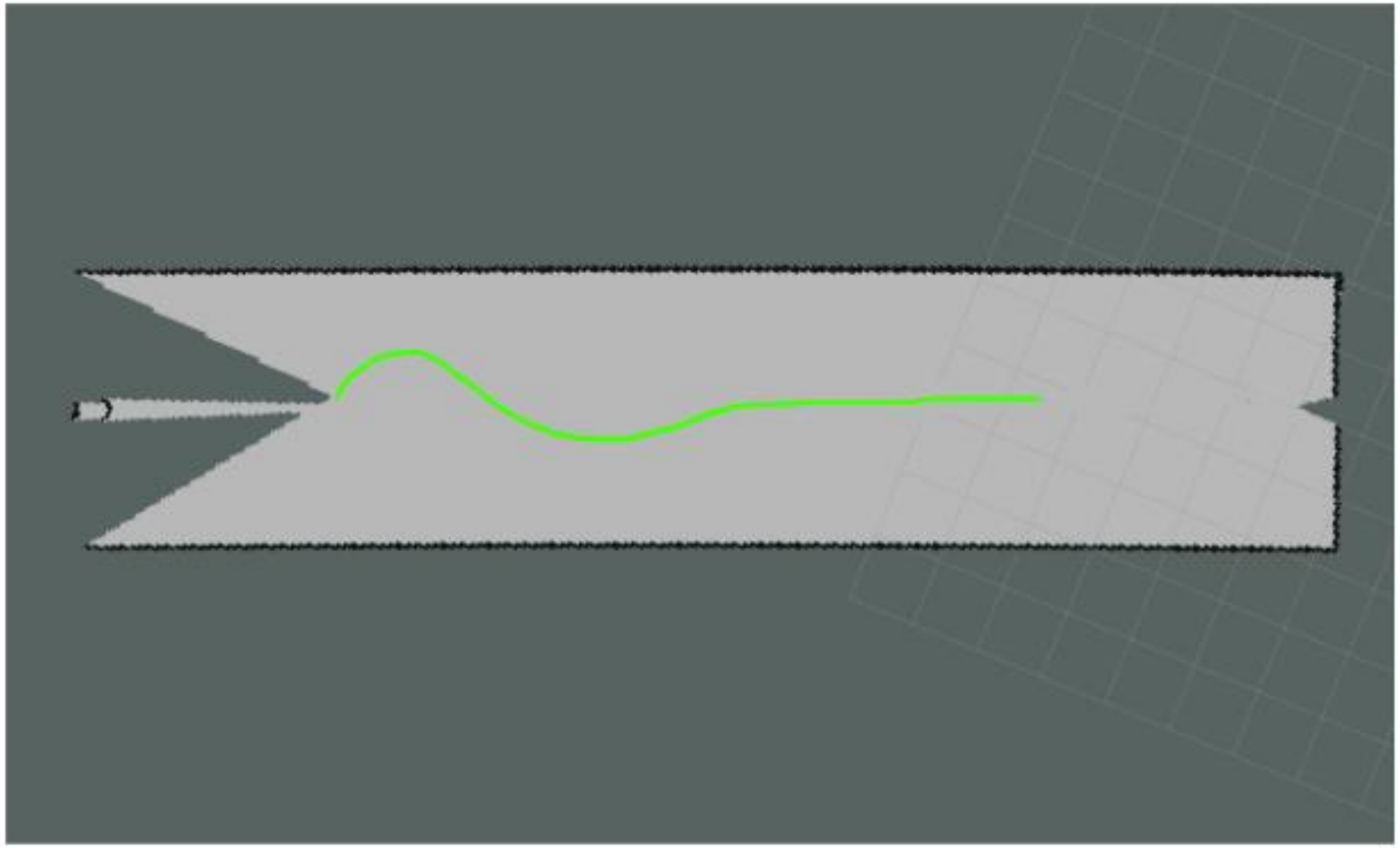


Overview

- Input
  - 1. Laser Scan

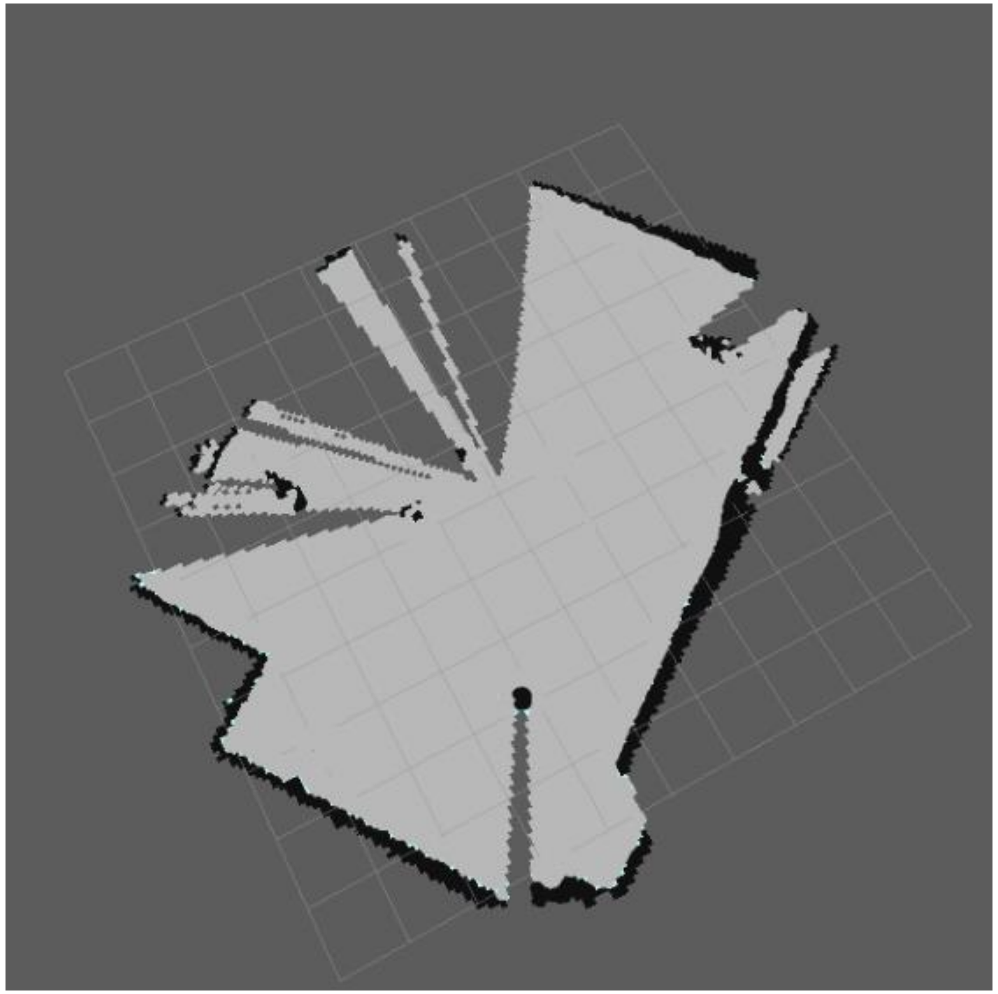
Overview

- Input
  - 1. Laser Scan
  - 2. Control Input



Overview

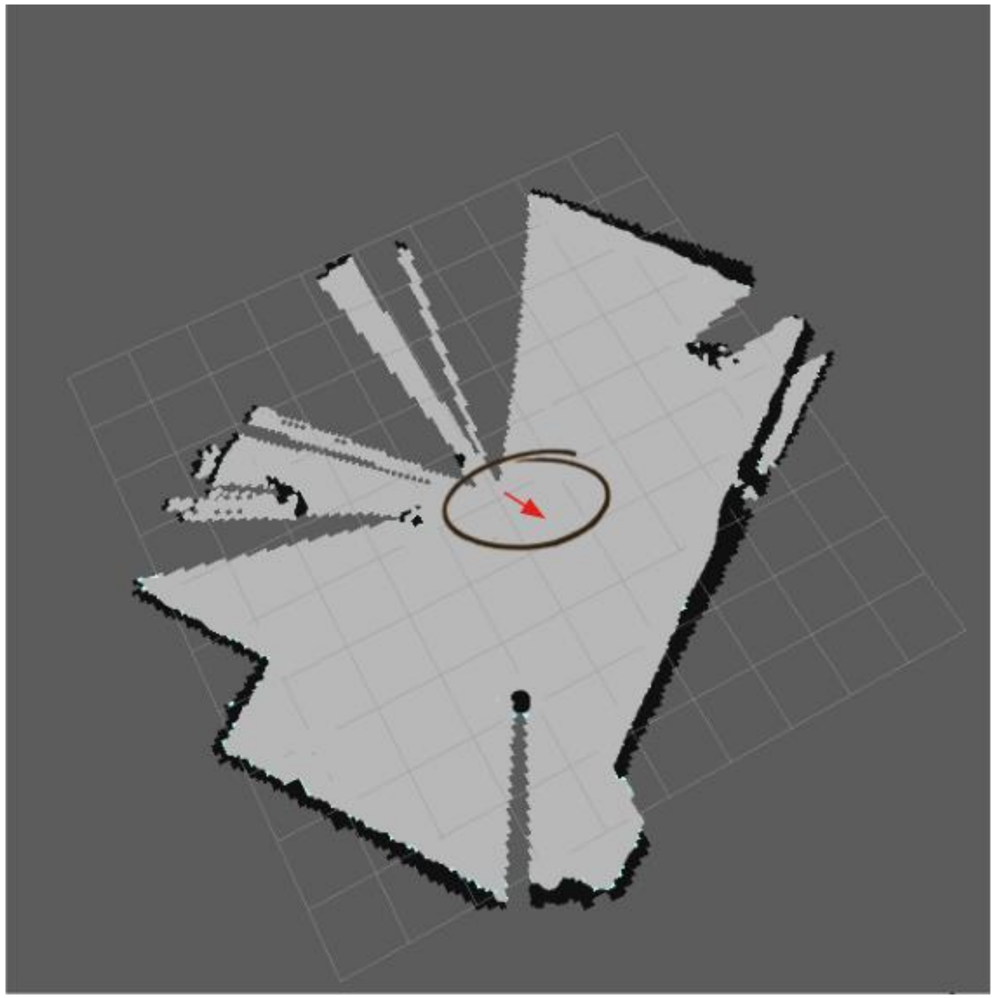
- Input
  - 1. Laser Scan
  - 2. Control Input
  - 3. Map



Overview

- Input
  - 1. Laser Scan
  - 2. Control Input
  - 3. Map

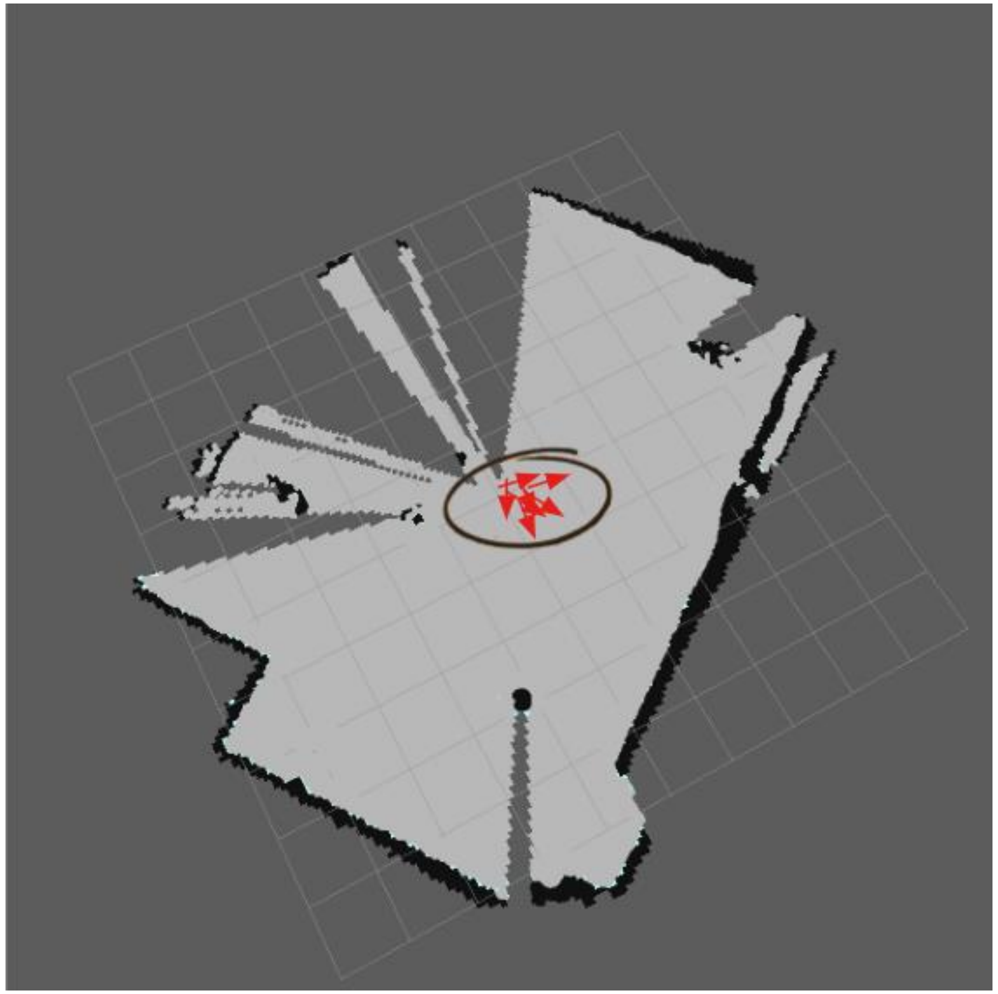
- Output
  - 1. Pose



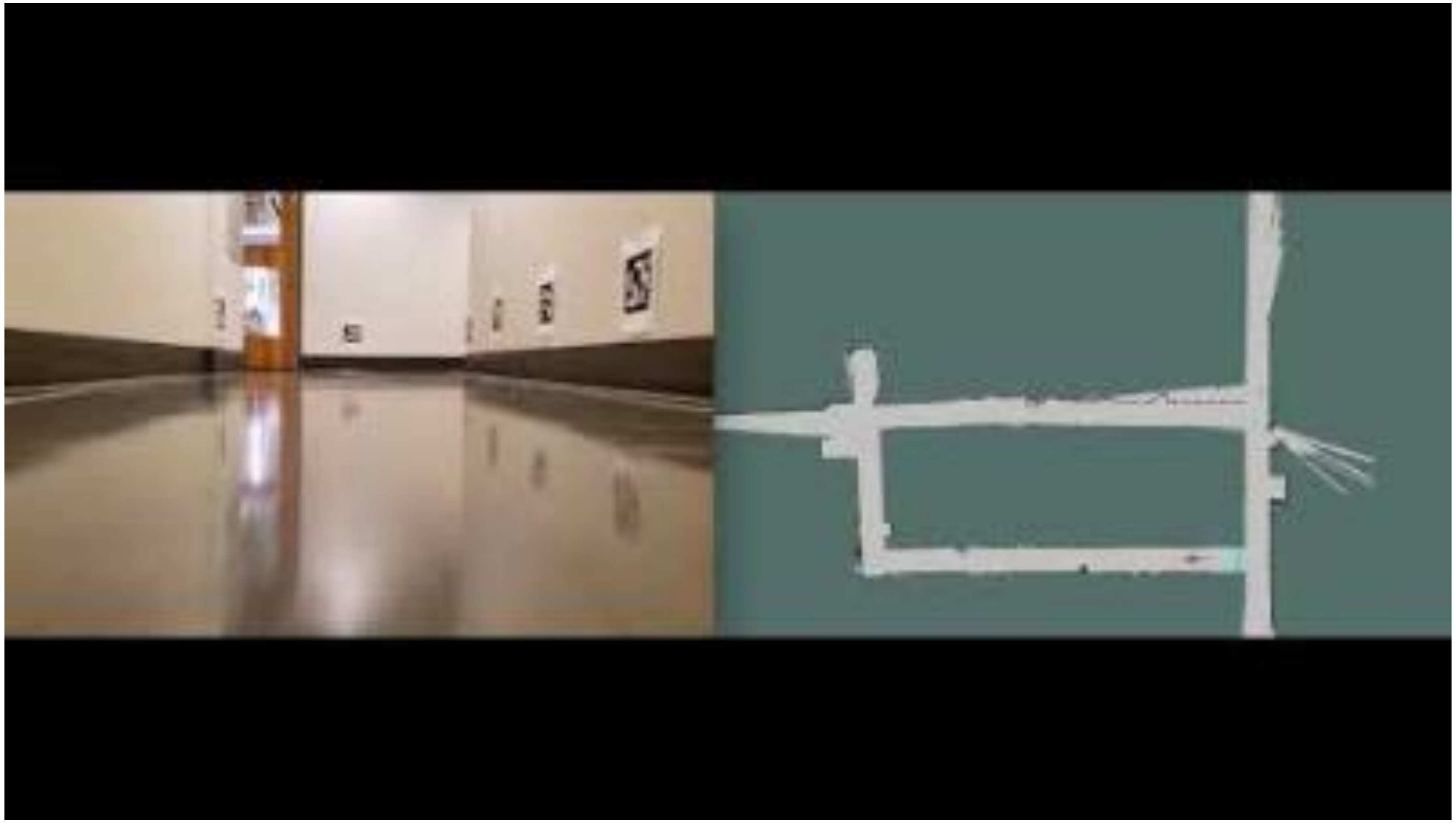
Overview

- Input
  - 1. Laser Scan
  - 2. Control Input
  - 3. Map

- Output
  - 1. Pose
  - 2. Particles



Overview



Particle Filter Localization – Key Idea

- The key idea of Bayes filtering is to estimate a probability density over the state-space conditioned on the data. This posterior is typically called the belief and is denoted:

$$Bel(x_t) = p(x_t | d_{0 \ldots t})$$

Belief or posterior

Robot state

At time t

The data from time 0 to t.

Particle Filter Localization – Questions

- 1. How would you describe the robot state in the localization problem?
- 2. Is the belief a probability distribution, what kind, how do you write it down?
- 3. How would you describe the data from the car?

$$Bel(x_t) = p(x_t | d_{0 \ldots t})$$

Belief or posterior

Robot state

At time t

The data from time 0 to t.

Particle Filter Localization – Getting more specific…

- For mobile robots, we distinguish two types of data: perceptual data such as laser range measurements, and odometry data, which carries information about robot motion. Denoting the former by o (for observation) and the latter by a (for action), we have:

$$Bel(x_t) = p(x_t | o_t, a_{t-1}, o_{t-1}, \dots)$$

Belief or posterior

Robot state

Perceptual Observation

Odometry

History…

HMCL
High-assurance Mobility Control Lab

Particle Filter Localization - Recursion

- After some simplification we have:

$$Bel(x_t) = \eta\, p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

Integrate out over previous beliefs. In practice finite approximation

Normalization Constant
Make sure everything adds up to 1!

Sensor Model
Compute how likely your measurements were given updated particles.

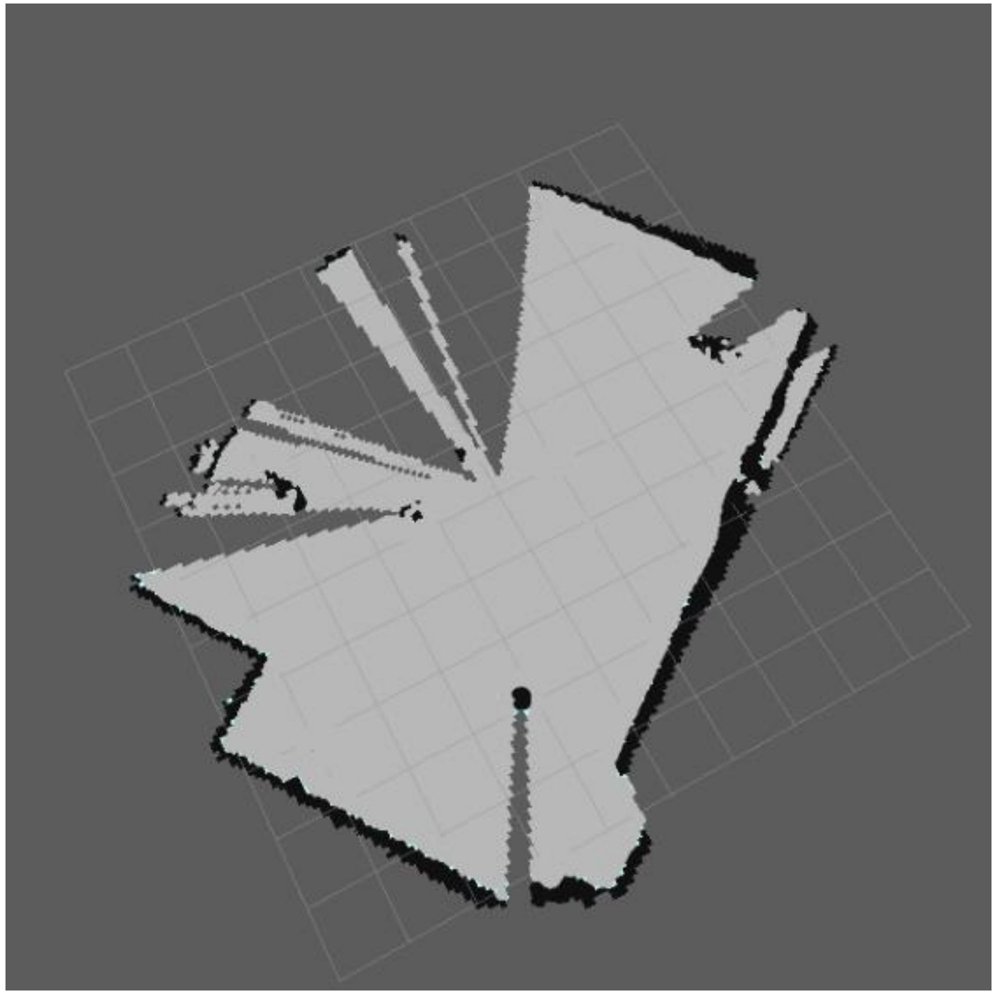
Motion Model
Simulate noisy dynamics of particles based on control input.

Previous Belief
Draw your particles with replacement according to importance weight.

In practice we represent the distributions non-parametrically using particles (a finite set of samples). More in the next slides.

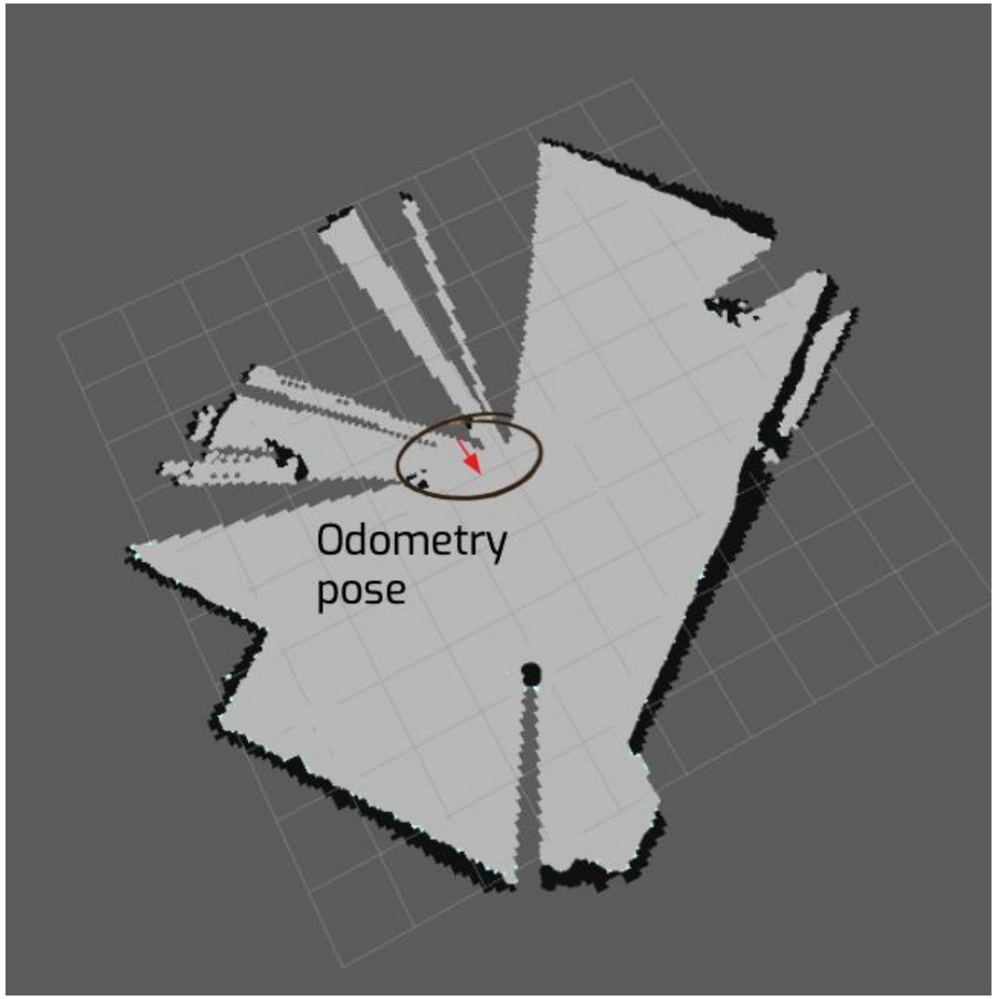
Particle Filter Localization - Initialization

- Now let’s look at particle filters in three dimensions and the process of using them in our car.
- Here we see a part of a map.
- The black pixels on the map denote the walls and the grey pixels denote free space.



Particle Filter Localization - Initialization

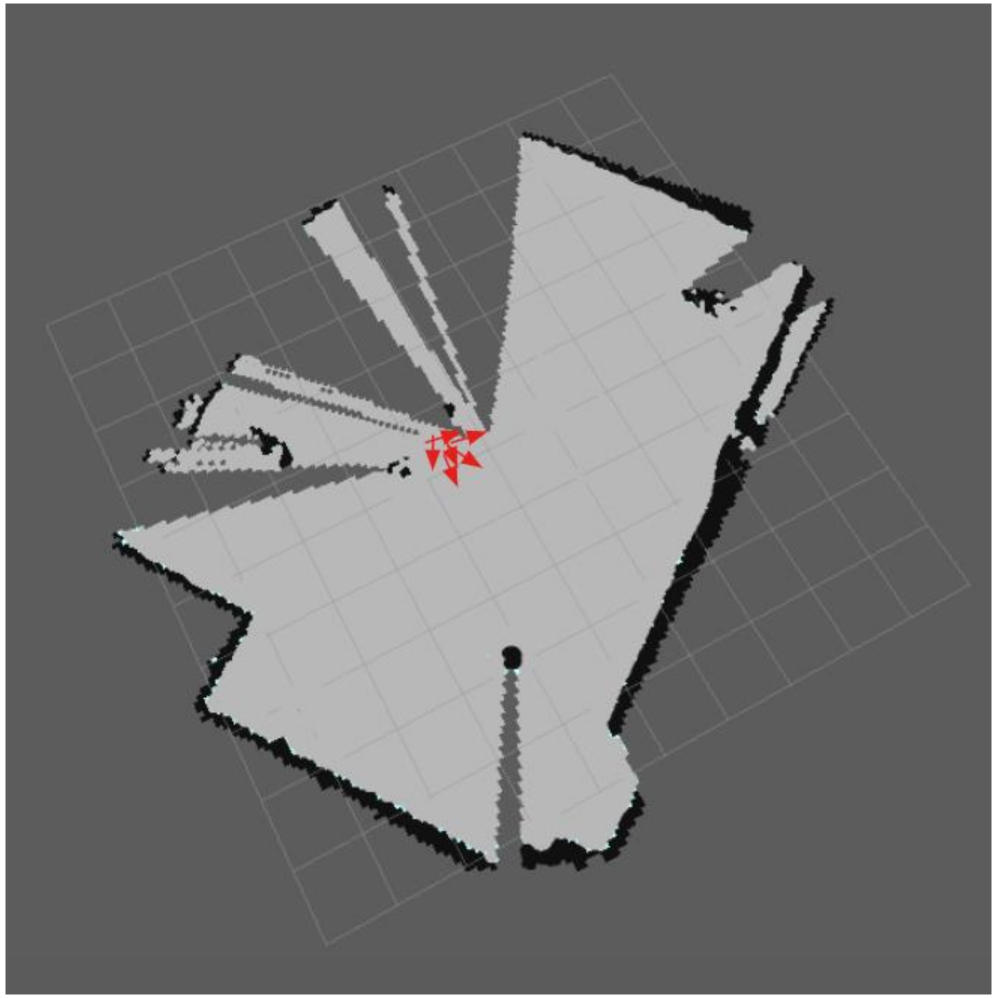
- The red arrow indicates out initial pose obtained from user input.
- Let’s use particle filters to solve the pose tracking problem and localize more accurately in the map.



Particle Filter Localization - Initialization

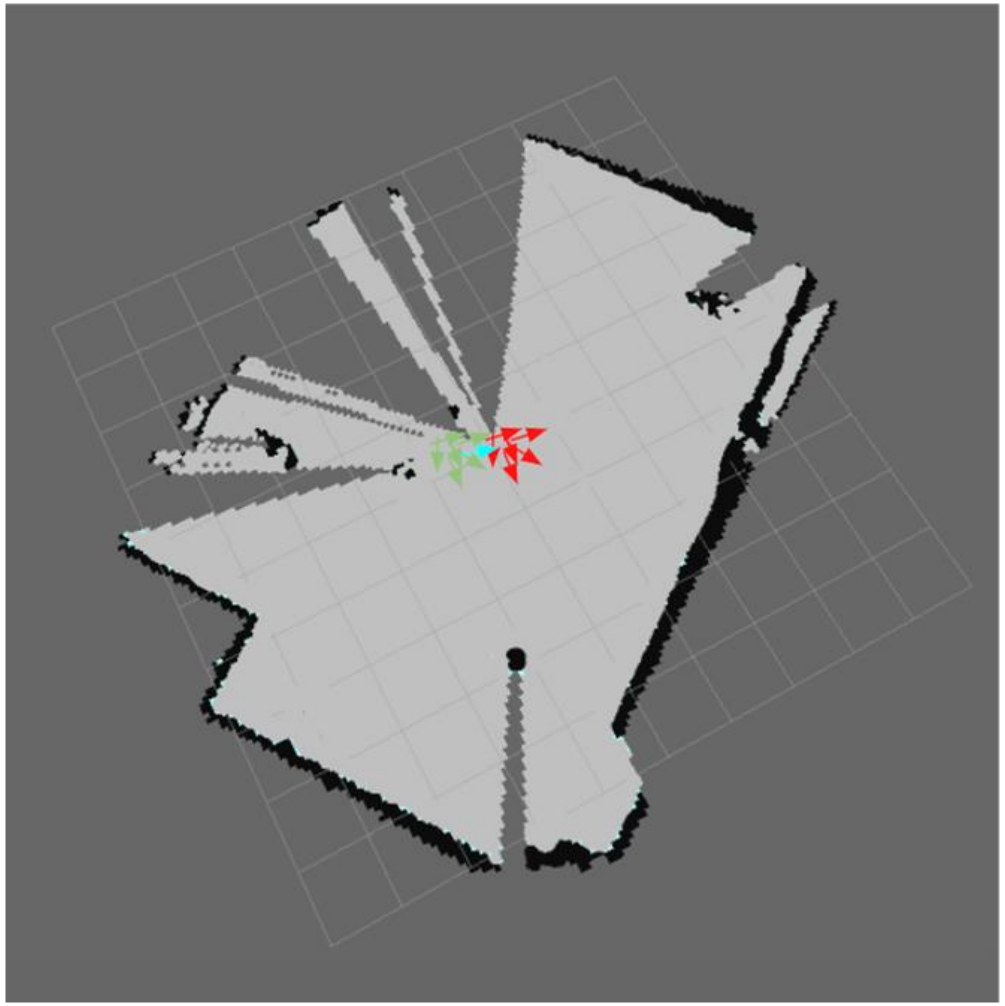
- First, we need to generate a set of hypothesis for our first position.

- These the discrete particles drawn from a Gaussian distribution of mean being the initial guess and with a small covariance.



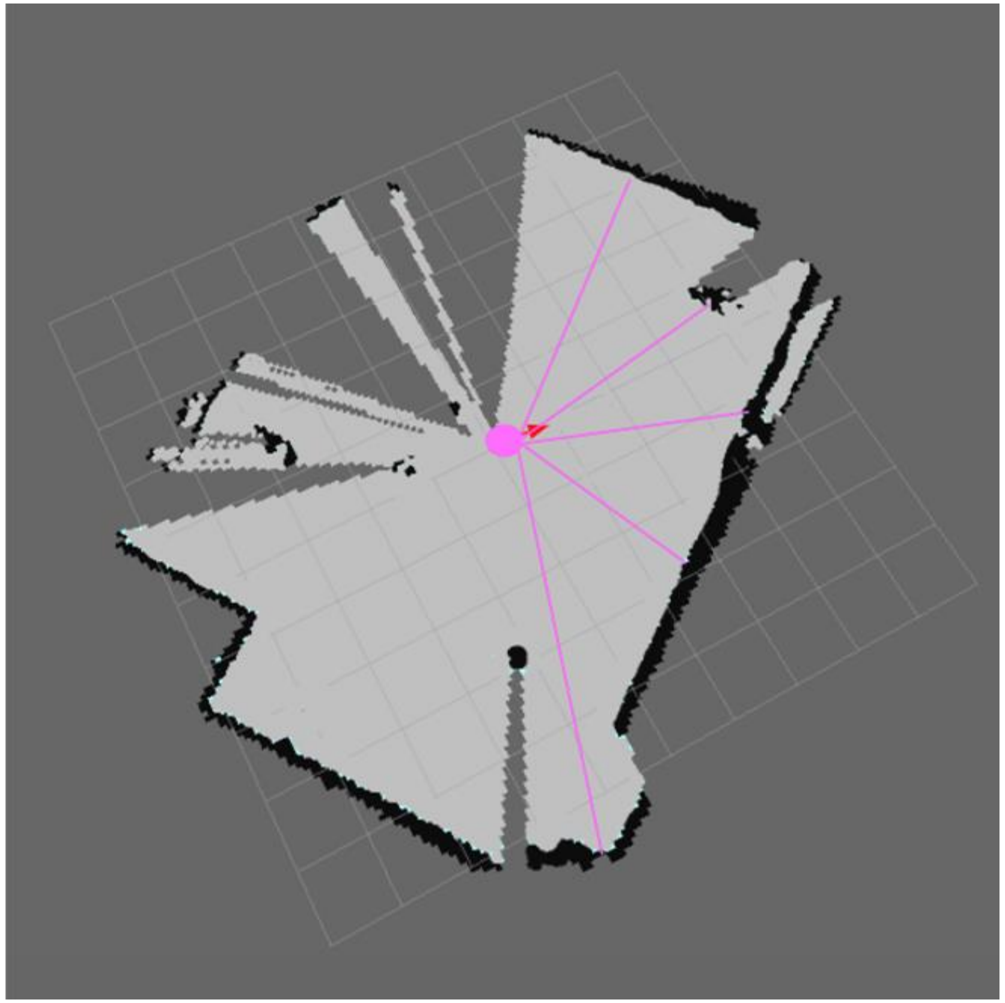
Particle Filter Localization – Motion Model

- Applying the measured control input to the motion model (with noise), yields an updates set of possible poses.



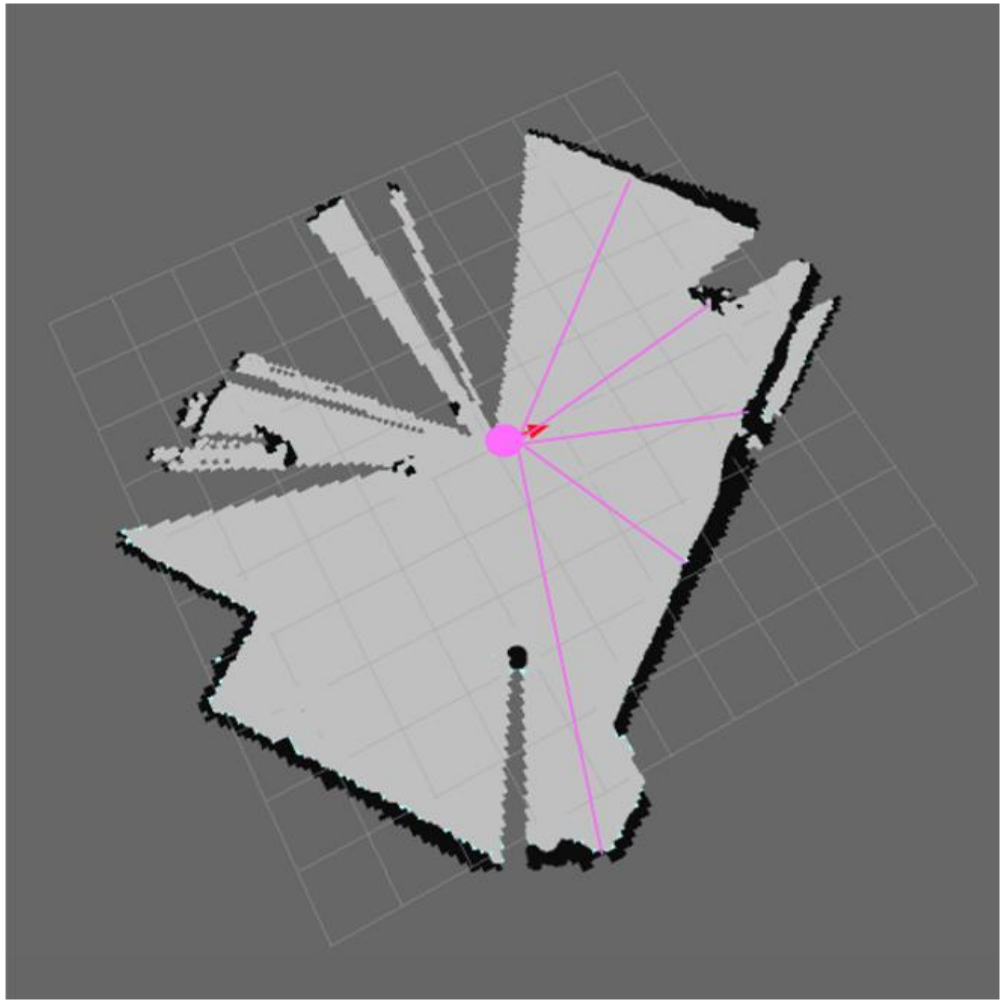
Particle Filter Localization – Sensor Model

- For each particle we can create a ‘fake’ laser scan by raycasting against the map at the particles pose.



Particle Filter Localization - Questions

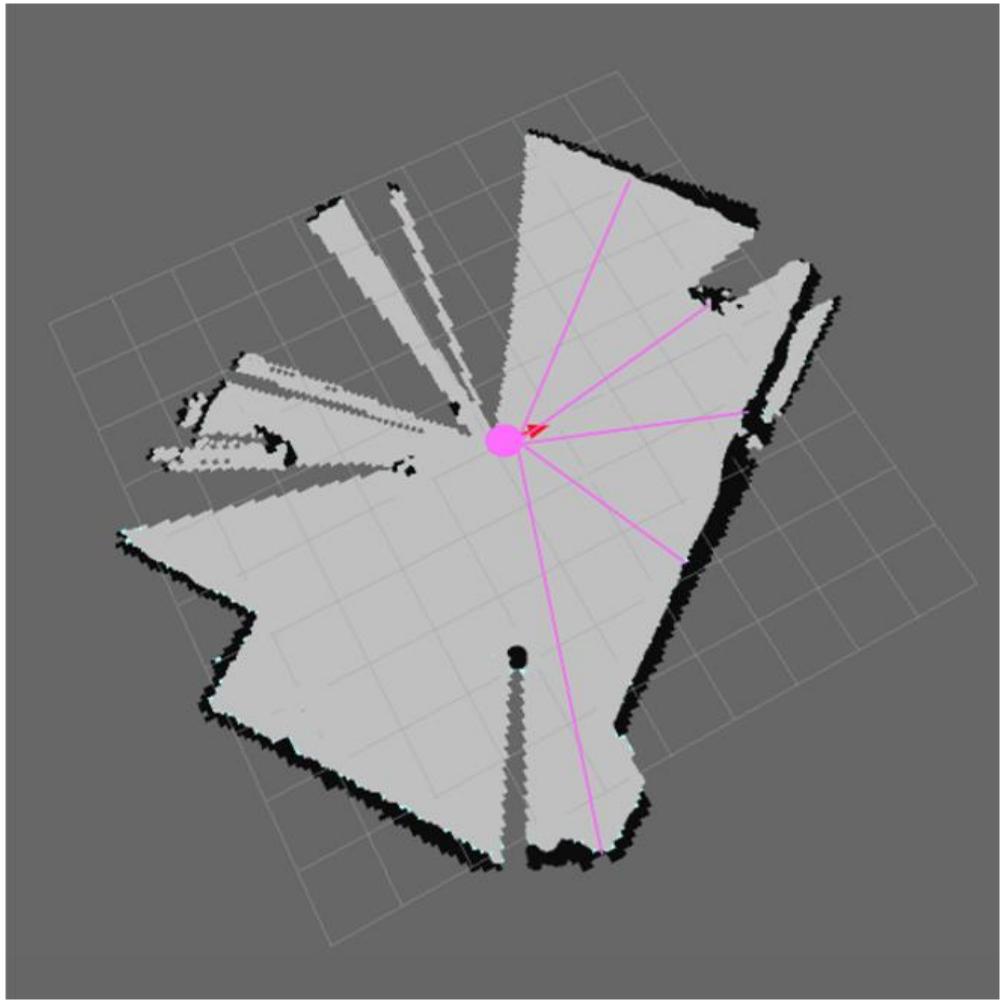
- How hard is creating the fake laser scan relative to computing the motion model update?
- What is special about the sensor model?



Particle Filter Localization - Questions

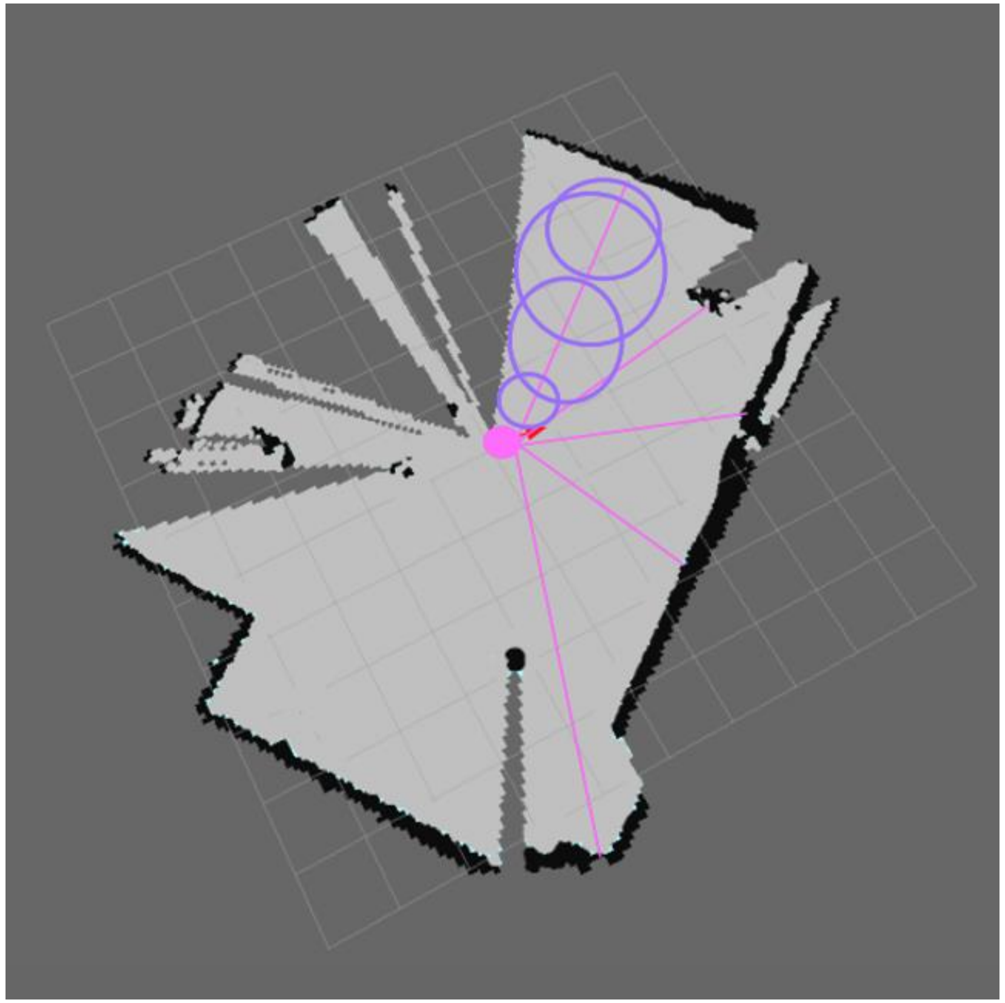
- How hard is creating the fake laser scan relative to computing the motion model update?
  → Way harder.

- What is special about the sensor model?
  → It's embarrassingly parallel.



Particle Filter Localization - Raycasting

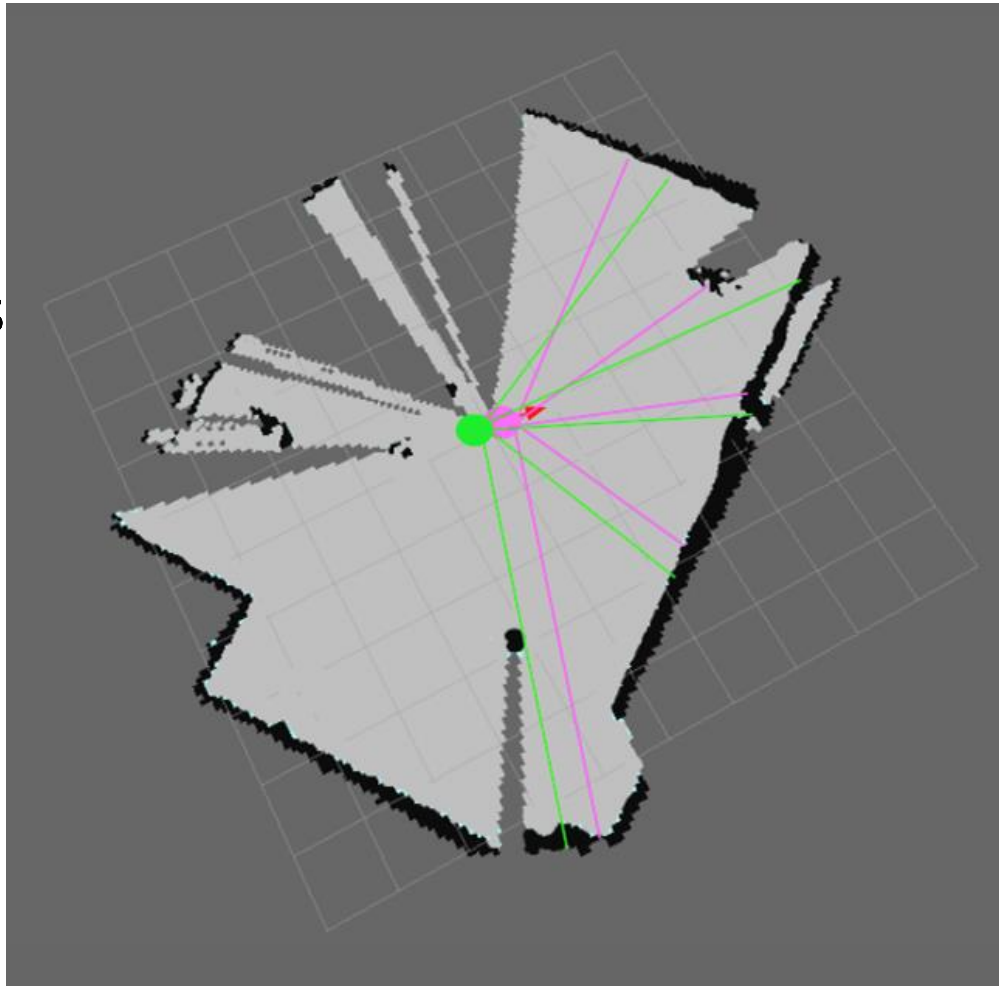
- Efficient methods for raycasting such as Breshenham’s Line method can be deployed.
- Alternately a signed distance field can be pre-computed.



Particle Filter Localization – Computing Particle Weights

- Recall, we have the 'real' laser scan which the car observed (shown in green).

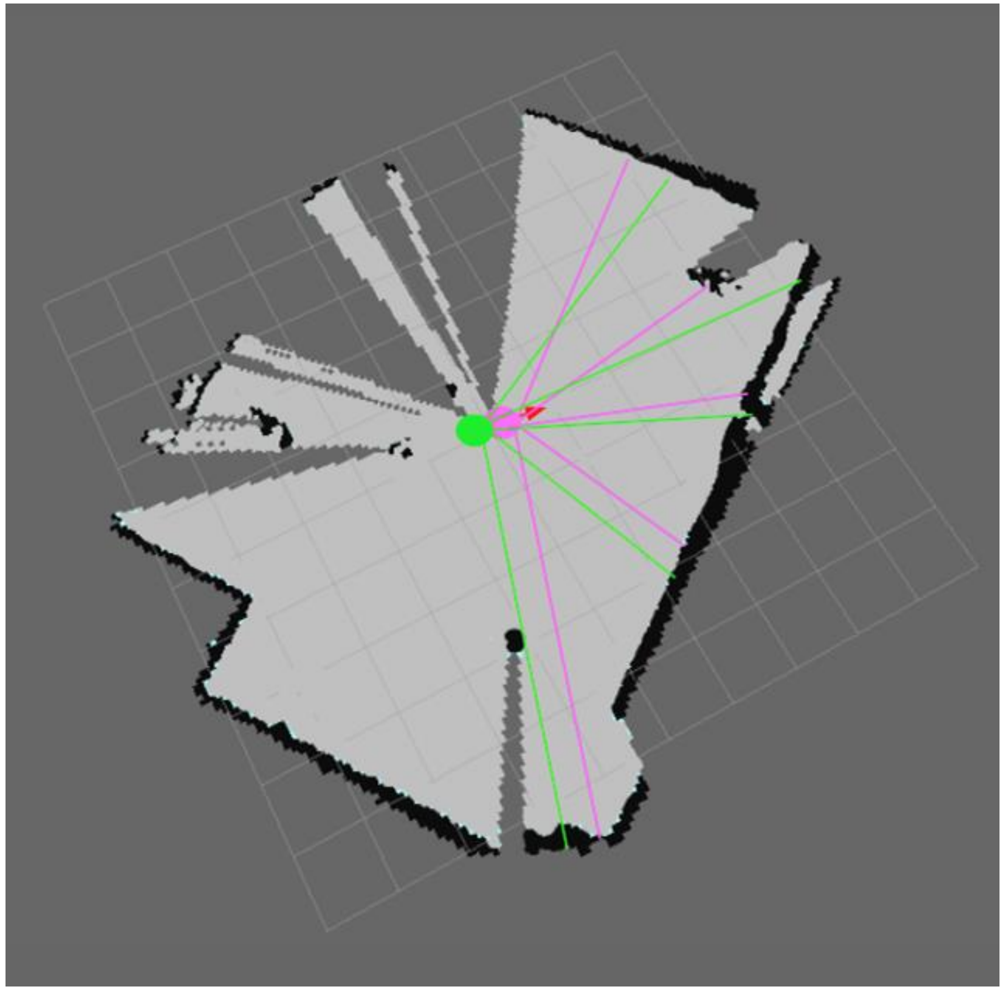
- Note we don't know where the green dot is but we do know the range measurements.



Particle Filter Localization – Computing Particle Weights

- We can compute a score for the fake laser scan:

$$p\left(o_t \middle| x_t^i\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(o_t - r^i)^2}{2\sigma^2}}$$



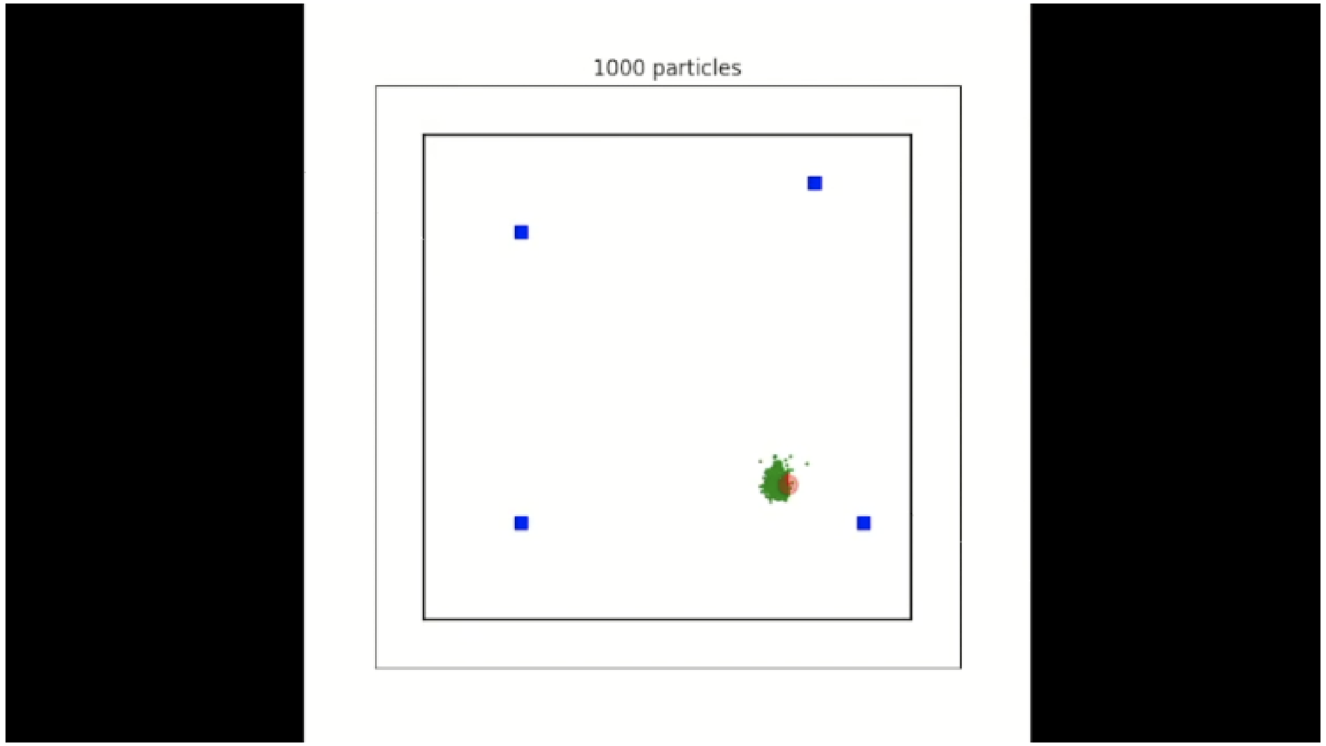
Particle Filter Localization – The Algorithm

```
# a single recursive update step of the MCL algorithm
def MCL ( $X_{t-1}, a_{t-1}, o$ ) :
    $X_t$ = { }
    for i in range(m):
        # sample a pose from the old particles according to old weights. Samples
        # implicitly represent the prior prob. Dist. Bel(x_{t_1}) in eqn.
        $x_{t-1}^i \sim X_{t-1}$
        # update the samples pose according to the motion model
        $x_{t-1}^i \sim p(x_t|x_{t-1}, a_{t-1})$
        # weight the updated pose according to the sensor model
        $w_t^i \sim p(o|x_t^i)$
        # add the new pose and weight to the new distribution
        $X_t \sim X_t \cup \{(x_t^i, w_t^i)\}$
    # normalize weights, should sum to 1
    $X_t = normalize(X_t)$
    return $X_t$
```

Particle Filter Localization – The Algorithm

```
# iterative application of the MCL algorithm
def particle_filter ( ) :
    $X = \mathrm{Bel}(x_0)$  $\leftarrow$ *initial particles*
    while true :
        a = get_last_odometry( )
        o = get_last_sensor_readings()
        X = MCL(X, a, o)
        # inferred pose ← expected value over particle distribution
        pose = Ex[X]
```

Practice: Particle Filter



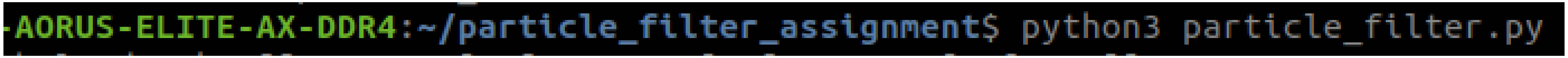
- The goal of this lab is to implement the Particle Filter algorithm.
- In this lab, we will implement a Particle filter is to estimate the probability distribution of the system state using the model's predicted values.

Overview

- For this lab, we will implement a ‘**Particle Filter**’ in the simulation environment.
- We need to **implement Particle Filter algorithm** to use predictions from the motion model.

Environment Setup

- In the file, you can find 2 folders and 1 file, (core, simulator, particle_filter.py).
- The "**particle_filter.py**" is an executable file that implements the particle filter algorithm.
- By running the "particle_filter.py" file, you can run the code and observe the particle filter localization.



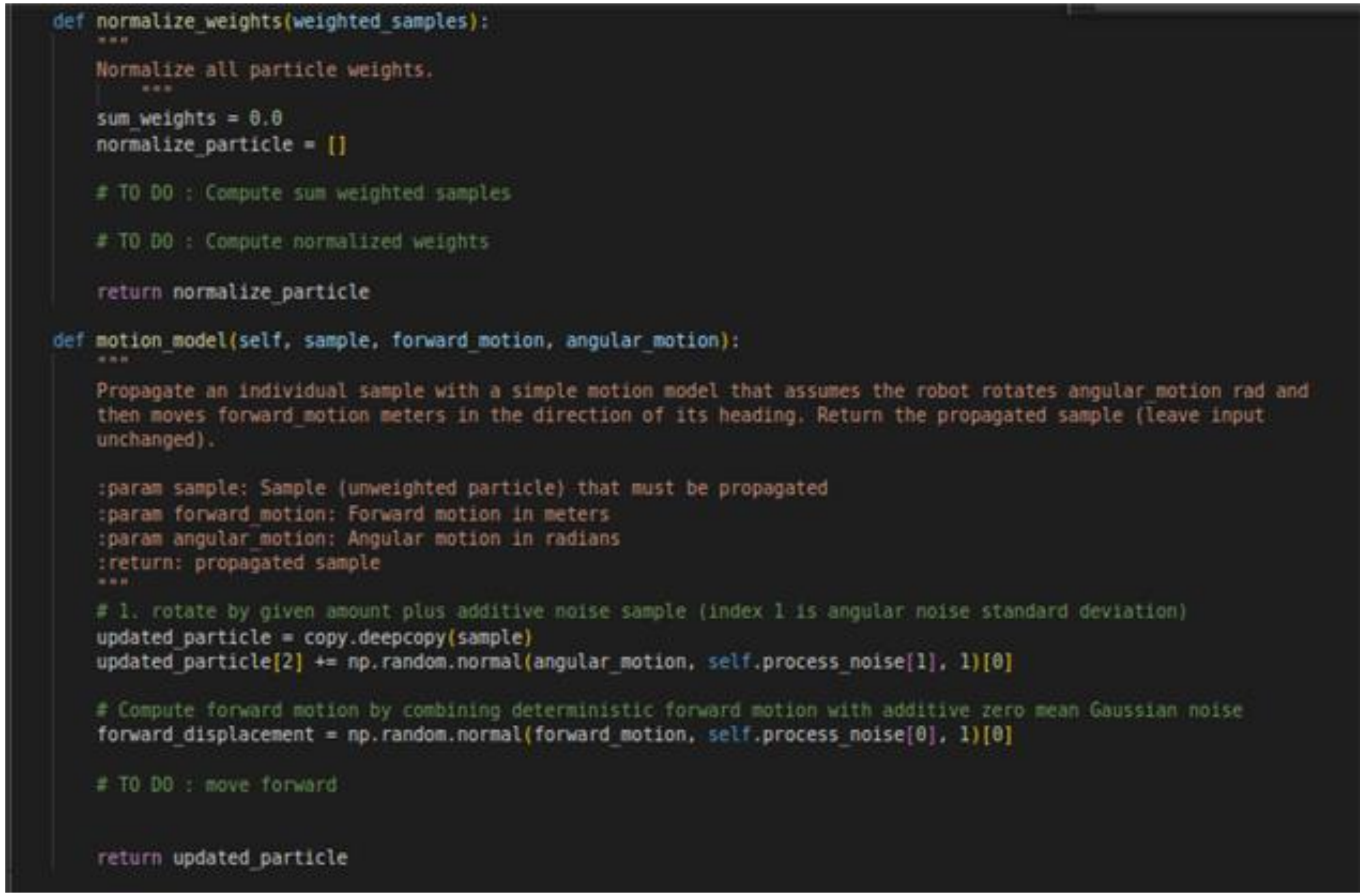
```
-AORUS-ELITE-AX-DDR4:~/particle_filter_assignment$ python3 particle_filter.py
```

particle_filter_sir.py

- To implement a particle filter, you need to write the code based on the file located at "**YOURWORKSPACE**/particle_filter_assignment/core/particle_filters/particle_filter_sir.py".

particle_filter_sir.py

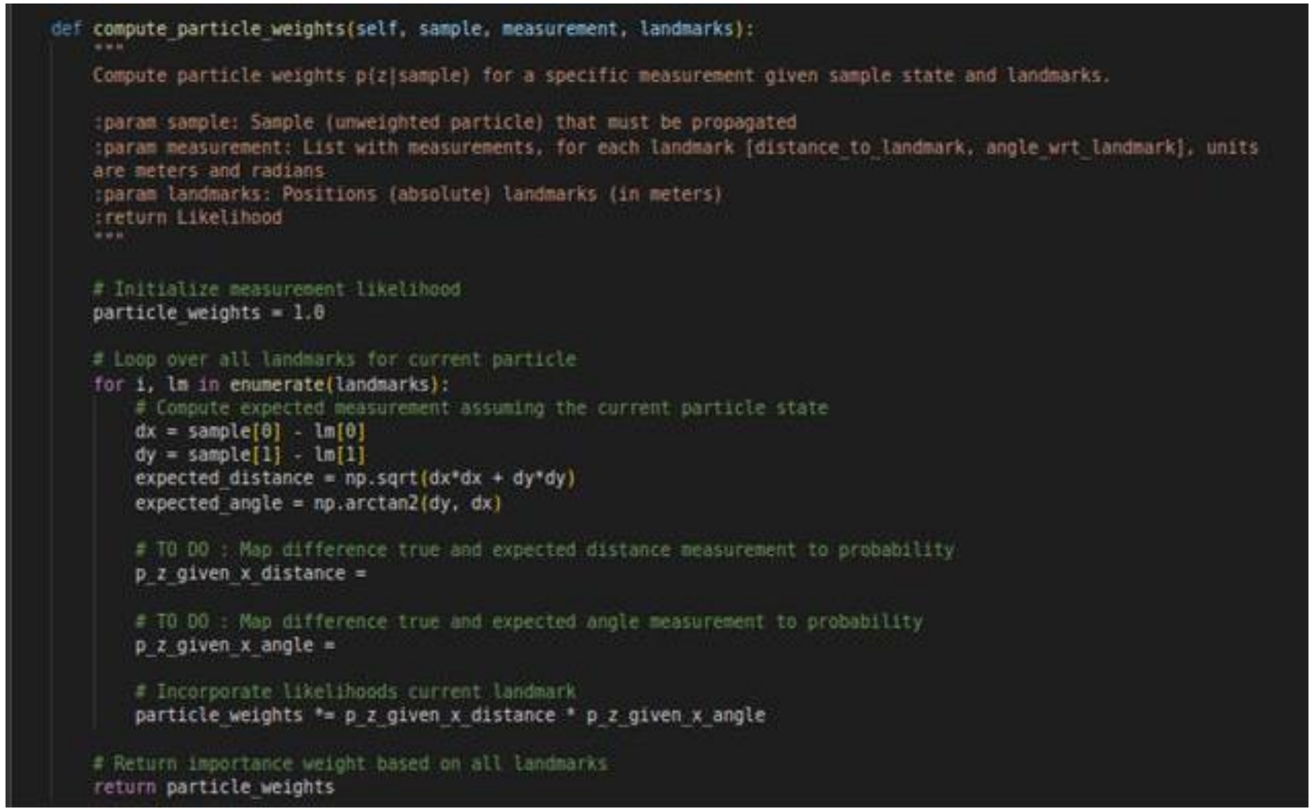
- The **normalize_weights** function takes a list of weighted samples as input and returns a list of normalized particles.
- The **motion_model** function propagates an individual sample by applying a simple motion model.



```
def normalize_weights(weighted_samples):
    """
    Normalize all particle weights.
        """
    sum_weights = 0.0
    normalize_particle = []

    # TO DO : Compute sum weighted samples

    # TO DO : Compute normalized weights

    return normalize_particle

def motion_model(self, sample, forward_motion, angular_motion):
    """
    Propagate an individual sample with a simple motion model that assumes the robot rotates angular_motion rad and
    then moves forward_motion meters in the direction of its heading. Return the propagated sample (leave input
    unchanged).

    :param sample: Sample (unweighted particle) that must be propagated
    :param forward_motion: Forward motion in meters
    :param angular_motion: Angular motion in radians
    :return: propagated sample
    """
    # 1. rotate by given amount plus additive noise sample (index 1 is angular noise standard deviation)
    updated_particle = copy.deepcopy(sample)
    updated_particle[2] += np.random.normal(angular_motion, self.process_noise[1], 1)[0]

    # Compute forward motion by combining deterministic forward motion with additive zero mean Gaussian noise
    forward_displacement = np.random.normal(forward_motion, self.process_noise[0], 1)[0]

    # TO DO : move forward


    return updated_particle
```

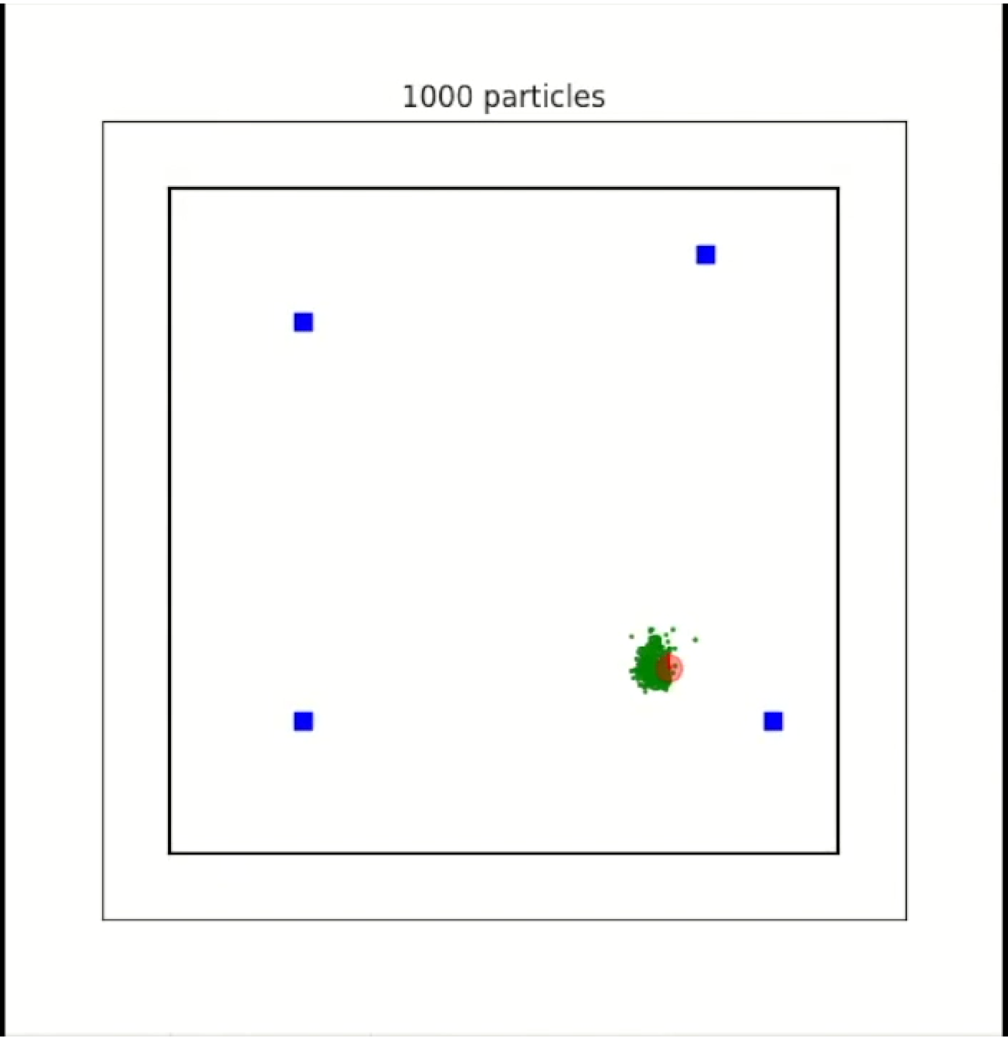
particle_filter_sir.py

- The **compute_particle_weights** function calculates the particle weights p(z|sample) for a specific measurement, given the sample state and landmarks.
- You need to complete the code by implementing the probability mapping for the distance and angle differences in order to obtain accurate particle weights.

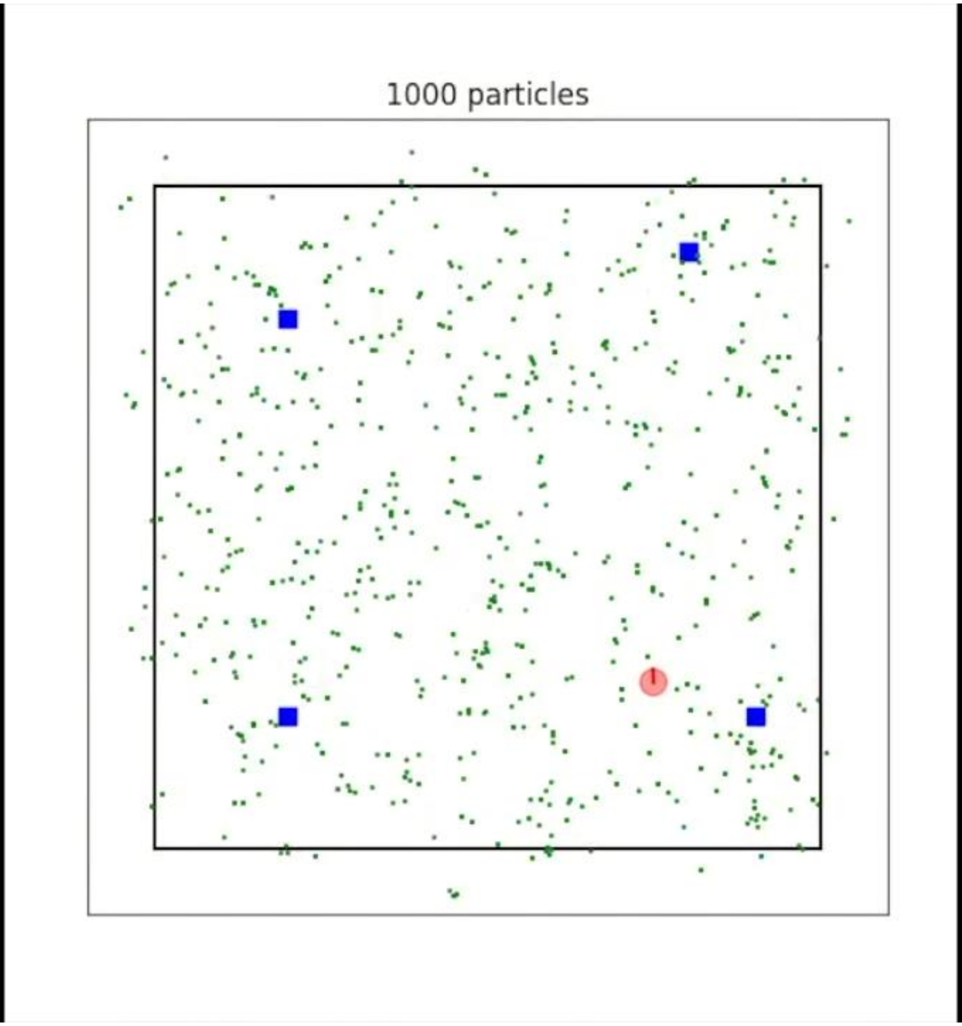


```python
def compute_particle_weights(self, sample, measurement, landmarks):
    """
    Compute particle weights p(z|sample) for a specific measurement given sample state and landmarks.

    :param sample: Sample (unweighted particle) that must be propagated
    :param measurement: List with measurements, for each landmark [distance_to_landmark, angle_wrt_landmark], units
    are meters and radians
    :param landmarks: Positions (absolute) landmarks (in meters)
    :return Likelihood
    """

    # Initialize measurement likelihood
    particle_weights = 1.0

    # Loop over all landmarks for current particle
    for i, lm in enumerate(landmarks):
        # Compute expected measurement assuming the current particle state
        dx = sample[0] - lm[0]
        dy = sample[1] - lm[1]
        expected_distance = np.sqrt(dx*dx + dy*dy)
        expected_angle = np.arctan2(dy, dx)

        # TO DO : Map difference true and expected distance measurement to probability
        p_z_given_x_distance =

        # TO DO : Map difference true and expected angle measurement to probability
        p_z_given_x_angle =

        # Incorporate likelihoods current landmark
        particle_weights *= p_z_given_x_distance * p_z_given_x_angle

    # Return importance weight based on all landmarks
    return particle_weights
```

Successful Example



Failed Example



Using Particle Filter

```
# build ‘particle filter’
mkdir –p particle_ws/src
cd ..
caktin_make

# Edit ‘bashrc’
gedit ~/.bashrc
alias particle=‘source ~/particle_ws/devel/setup.bash && roslaunch particle_filter localize.launch’


# Run ‘particle filter’
particle
```