

# Mapping and Navigation

# OVERVIEW

---

## **Mapping and Navigation Concept**

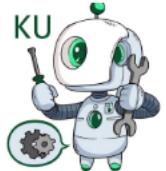
- Understanding the mapping and navigation concept
- Setup Software Environment

## **Creating a Map**

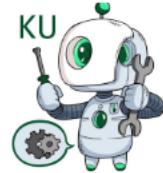
- SLAM
- Creating a map with slam gmapping package
- Save the map and Spawn the robot

## **Navigation and Autonomous driving**

- AMCL Localization
- Install and testing Kinect (3D sensor)
- Autonomous driving with a simple program

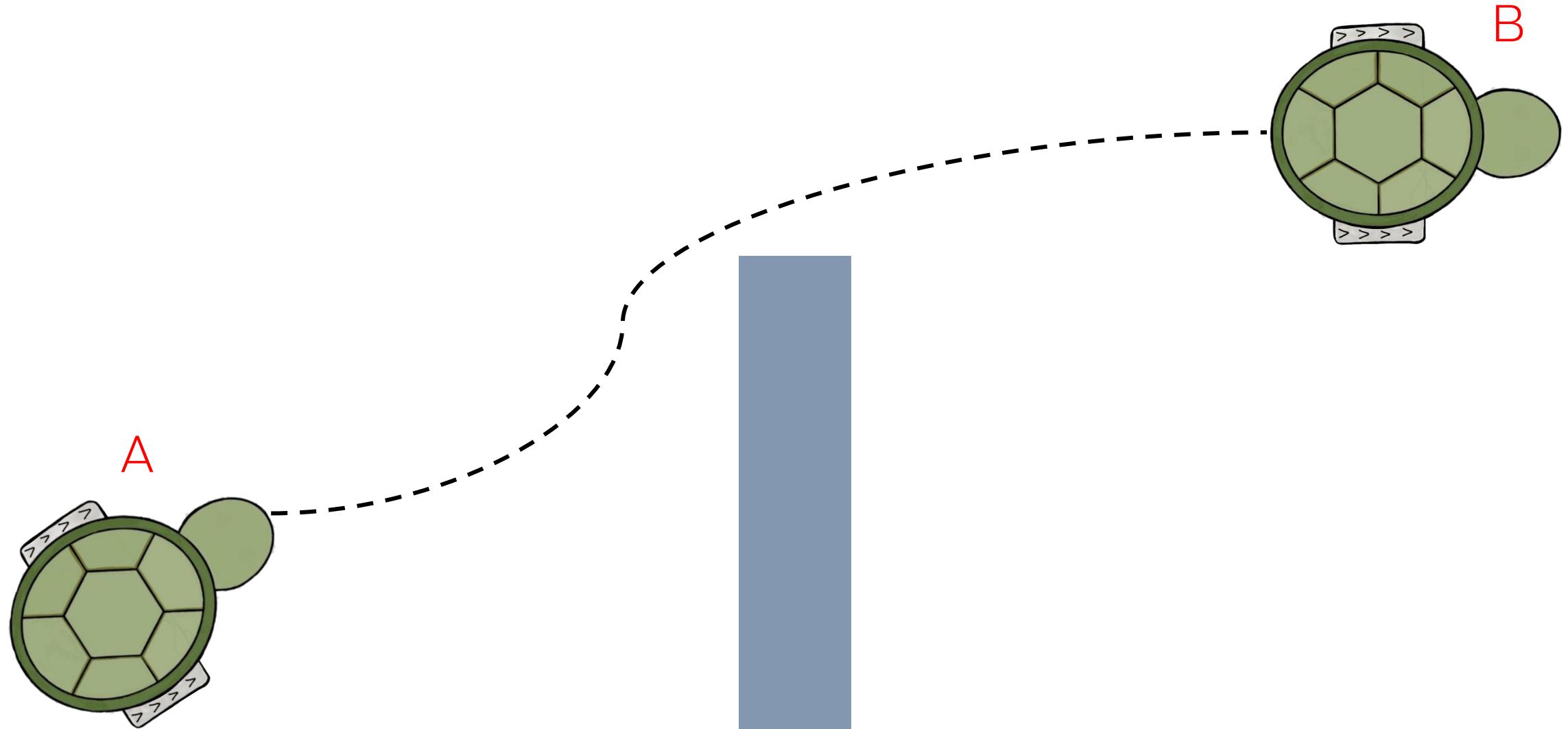


# **Mapping and Navigation Concept**



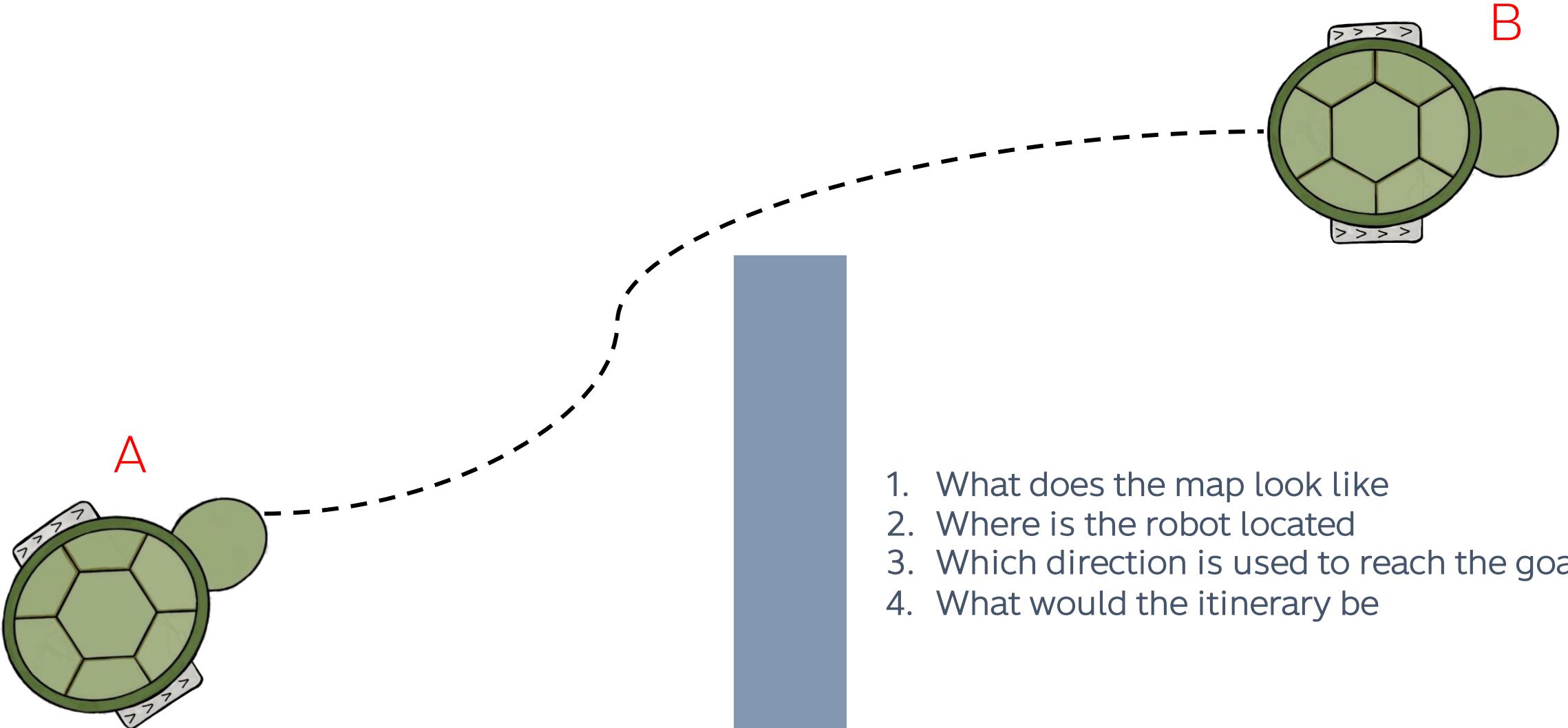


# Mapping and Navigation Concept





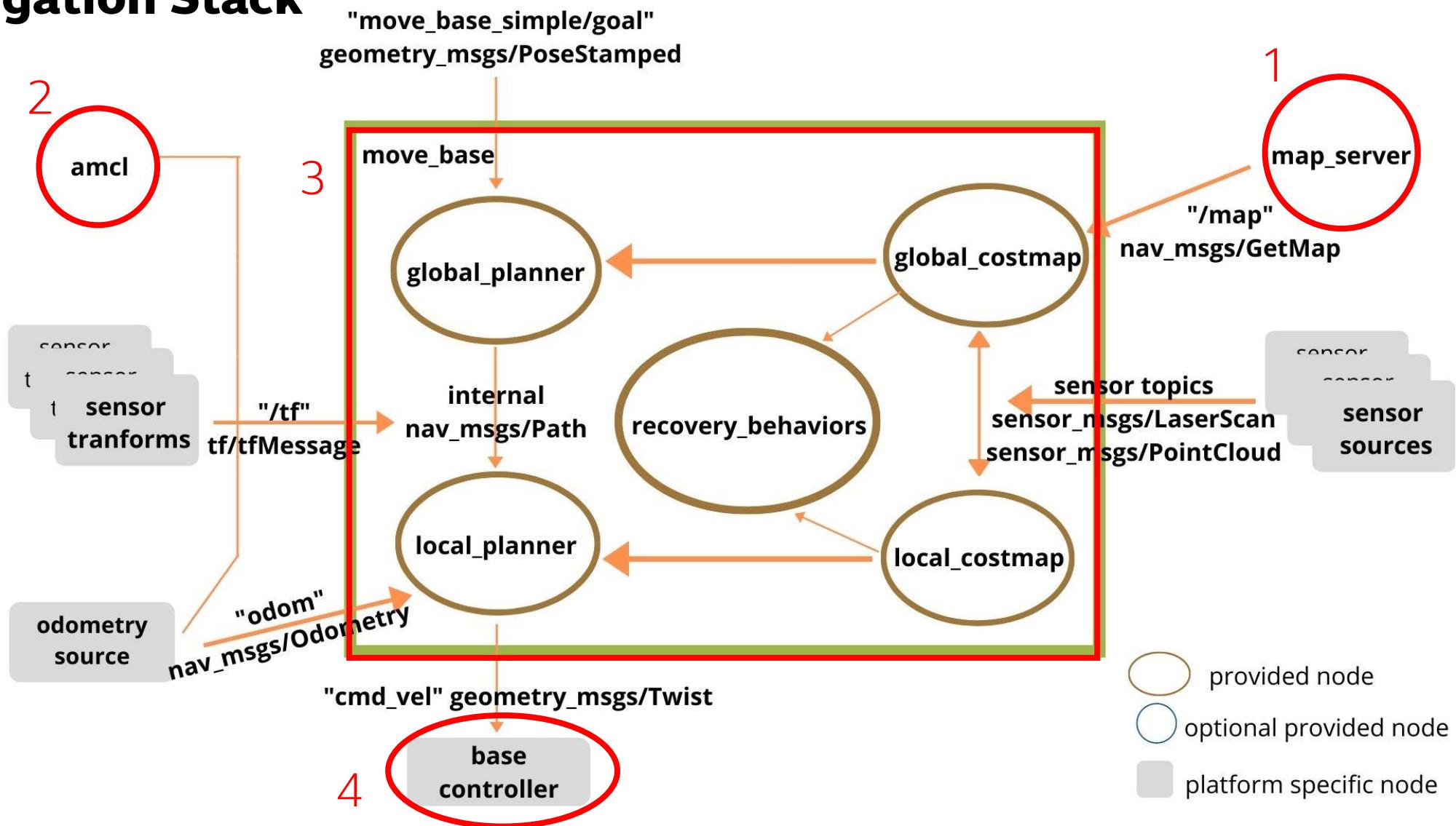
# Mapping and Navigation Concept



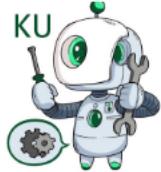
1. What does the map look like
2. Where is the robot located
3. Which direction is used to reach the goal
4. What would the itinerary be

# Mapping and Navigation Concept

## ROS Navigation Stack

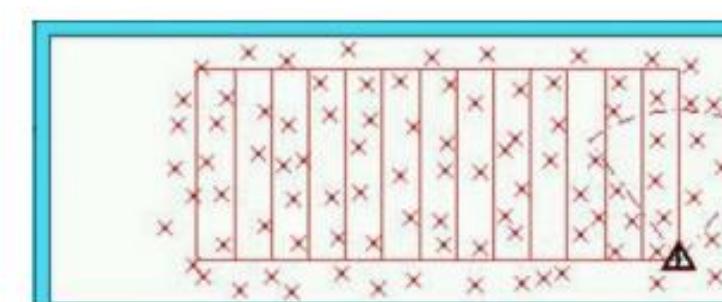
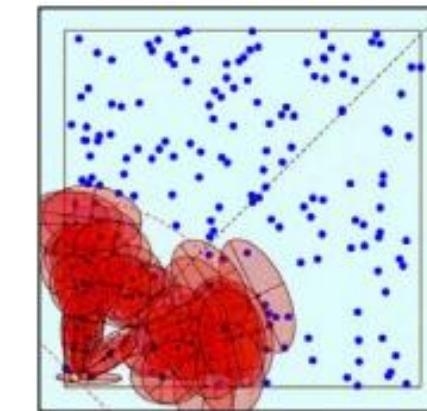
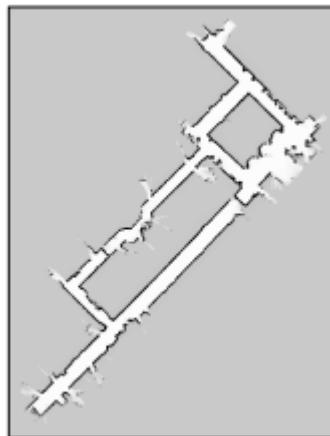


# Creating a Map



## Map Representations in Robotics

- Grid maps or scans, 2d, 3d
- Landmark-based



# Creating a Map (With Known Poses)

- Normally, mapping with known poses involves, given the measurements and the poses

$$d = \{x_1, z_1, x_2, z_2, \dots, x_t, z_t\}$$

- To calculate the most likely map

$$m^* = \operatorname{argmax}_m P(m \mid d)$$

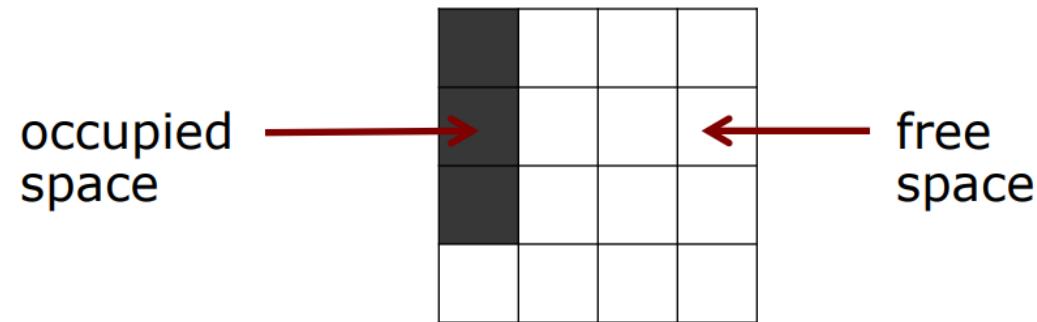
## Grid Maps

- We discretize the world into cells
- The grid structure is rigid
- Each cell is assumed to be occupied or free
- It is a non-parametric model
- It requires substantial memory resources
- It does not rely on a feature detector

# Creating a Map (With Known Poses)

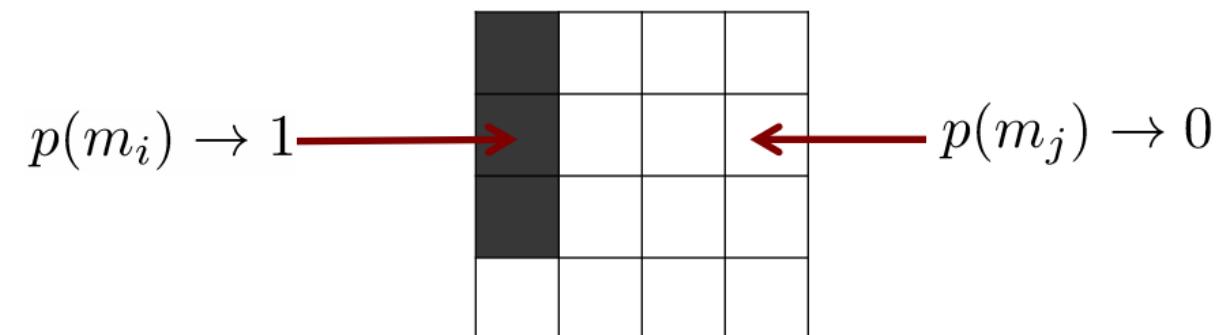
## Assumption 1

- The area that corresponds to a cell is either completely free or occupied



## Representation

- Each cell is a binary random variable that models the occupancy

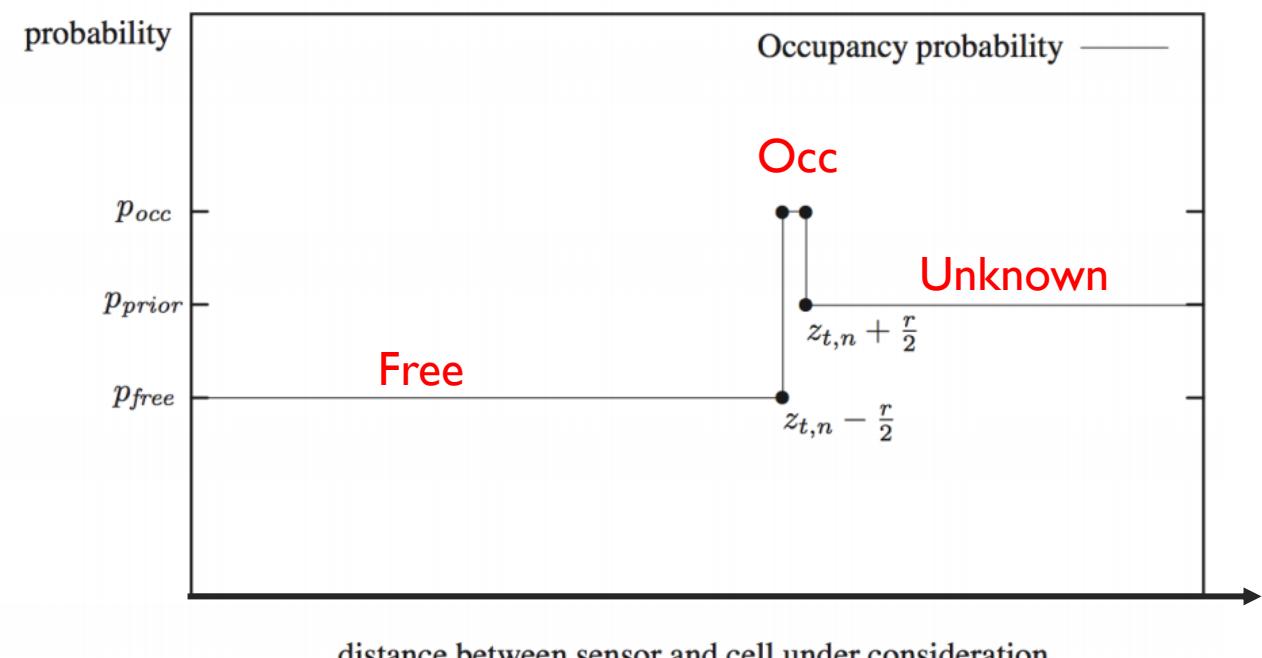


# Creating a Map (With Known Poses)

## Occupancy Probability

- Each cell is a binary random variable that models the occupancy
- Cell is occupied  $p(m_i) = 1$
- Cell is not occupied  $p(m_i) = 0$
- No information  $p(m_i) = 0.5$
- The environment is assumed to be static

## Inverse Sensor Model for Laser Range Finders

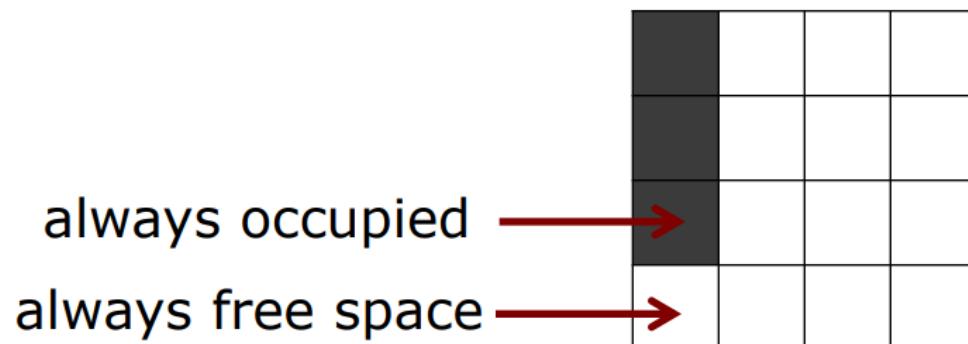




# Creating a Map (With Known Poses)

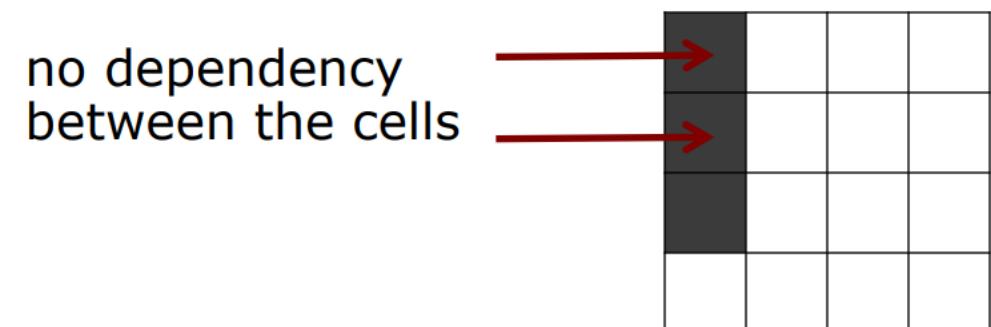
## Assumption 2

- The world is static (most mapping systems make this assumption)



## Assumption 3

- The cells (the random variables) are independent of each other



# Creating a Map (With Known Poses)

## Representation

- The probability distribution of the map is given by the product over the cells

$$p(m) = \prod_i p(m_i)$$





example map  
(4-dim state)

4 individual cells

## Estimating a Map From Data

- Given sensor data  $z_{1:t}$  and the poses  $x_{1:t}$  of the sensor, estimate the map

$$p(m | z_{1:t}, x_{1:t}) = \prod_i p(m_i | z_{1:t}, x_{1:t})$$



binary random variable

→ Binary Bayes filter  
(for a static state)

# Creating a Map (With Known Poses)

## Log Odds Notation

- The log odds notation computes the logarithm of the ratio of probabilities

$$\frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} = \underbrace{\frac{p(m_i | z_t, x_t)}{1 - p(m_i | z_t, x_t)}}_{\text{uses } z_t} \underbrace{\frac{p(m_i | z_{1:t-1}, x_{1:t-1})}{1 - p(m_i | z_{1:t-1}, x_{1:t-1})}}_{\text{recursive term}} \underbrace{\frac{1 - p(m_i)}{p(m_i)}}_{\text{prior}}$$

$$\rightarrow l(m_i | z_{1:t}, x_{1:t}) = \log \left( \frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} \right)$$

## Occupancy Mapping in Log Odds Form

- The product turns into a sum

$$l(m_i | z_{1:t}, x_{1:t}) = \underbrace{l(m_i | z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i | z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}}$$

- Or in short

$$l_{t,i} = \text{inv\_sensor\_model}(m_i, x_t, z_t) + l_{t-1,i} - l_0$$

Log odds ratio is defined as

$$l(x) = \log \frac{p(x)}{1 - p(x)}$$

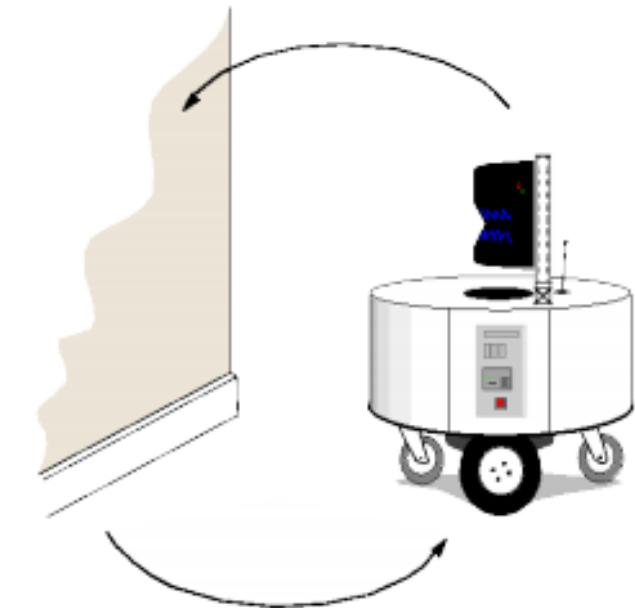
and with the ability to retrieve  $p(x)$

$$p(x) = 1 - \frac{1}{1 + \exp l(x)}$$

# Creating a Map (SLAM Concept)

SLAM has long been regarded as a **chicken-or-egg** problem:

- a **map** is needed for **localization** and
- a **pose estimate** is needed for **mapping**
- **Localization**: inferring location given a **map**
- **Mapping**: inferring a map given **locations**
- **SLAM**: learning a map and locating the robot simultaneously



# Creating a Map (SLAM Concept)

**Given:**

- The robot's controls

$$\mathbf{U}_{1:k} = \{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k\}$$

- Relative observations

$$\mathbf{Z}_{1:k} = \{z_1, z_2, \dots, z_k\}$$

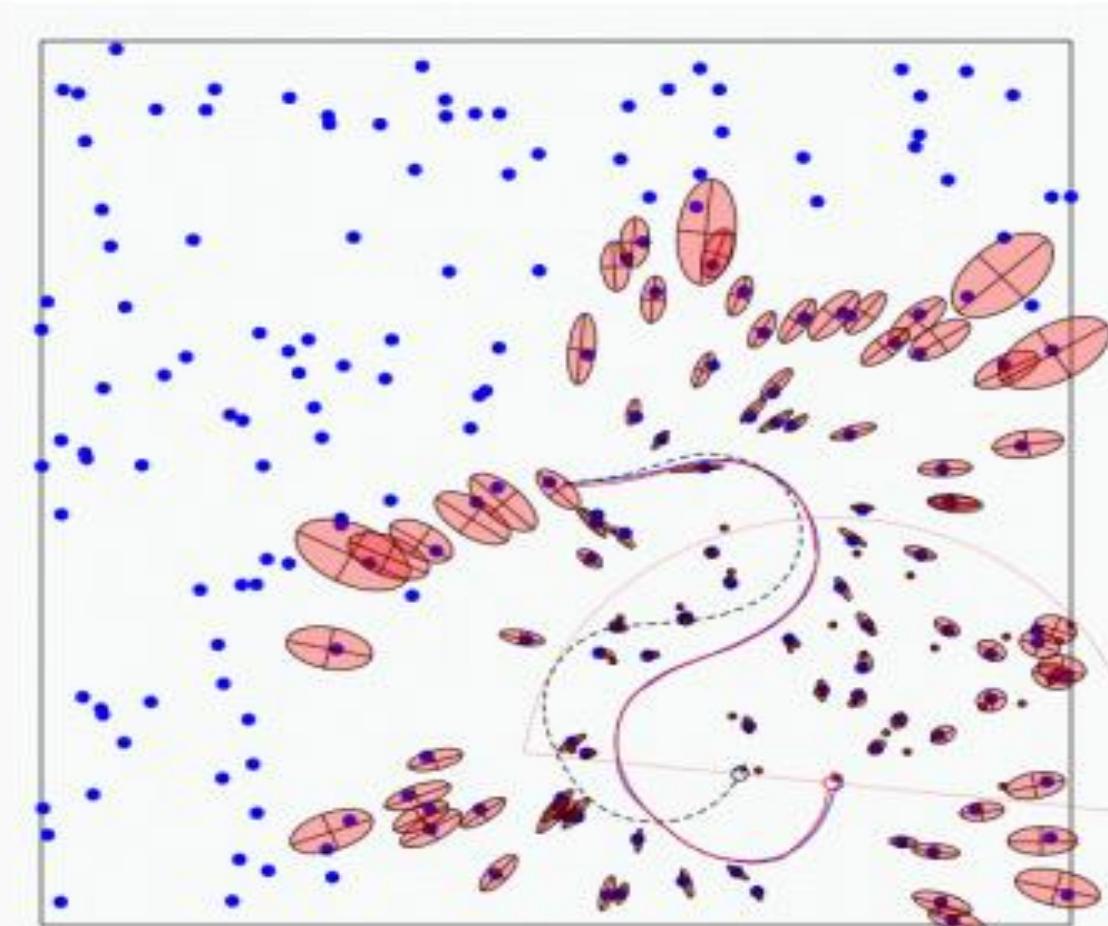
**Wanted:**

- Map of features

$$\mathbf{m} = \{m_1, m_2, \dots, m_n\}$$

- Path of the robot

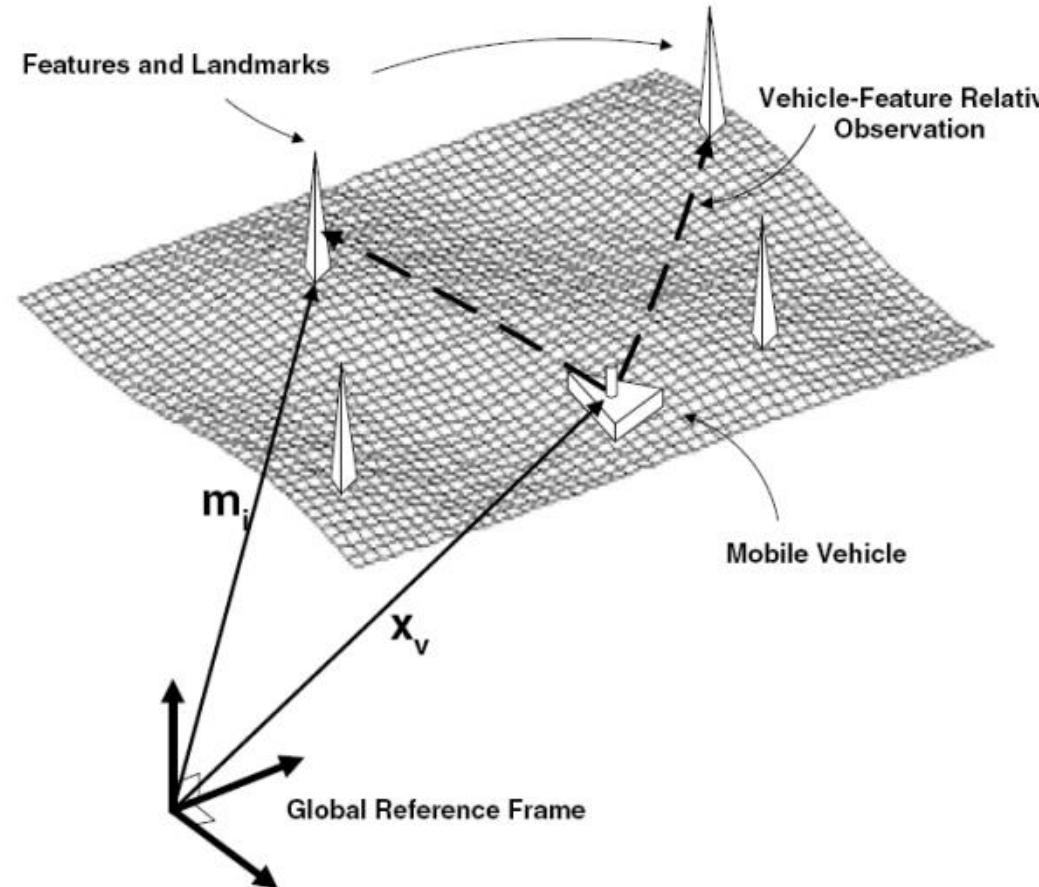
$$\mathbf{X}_{1:k} = \{x_1, x_2, \dots, x_k\}$$



# Creating a Map (SLAM Concept)

## Feature-Based SLAM

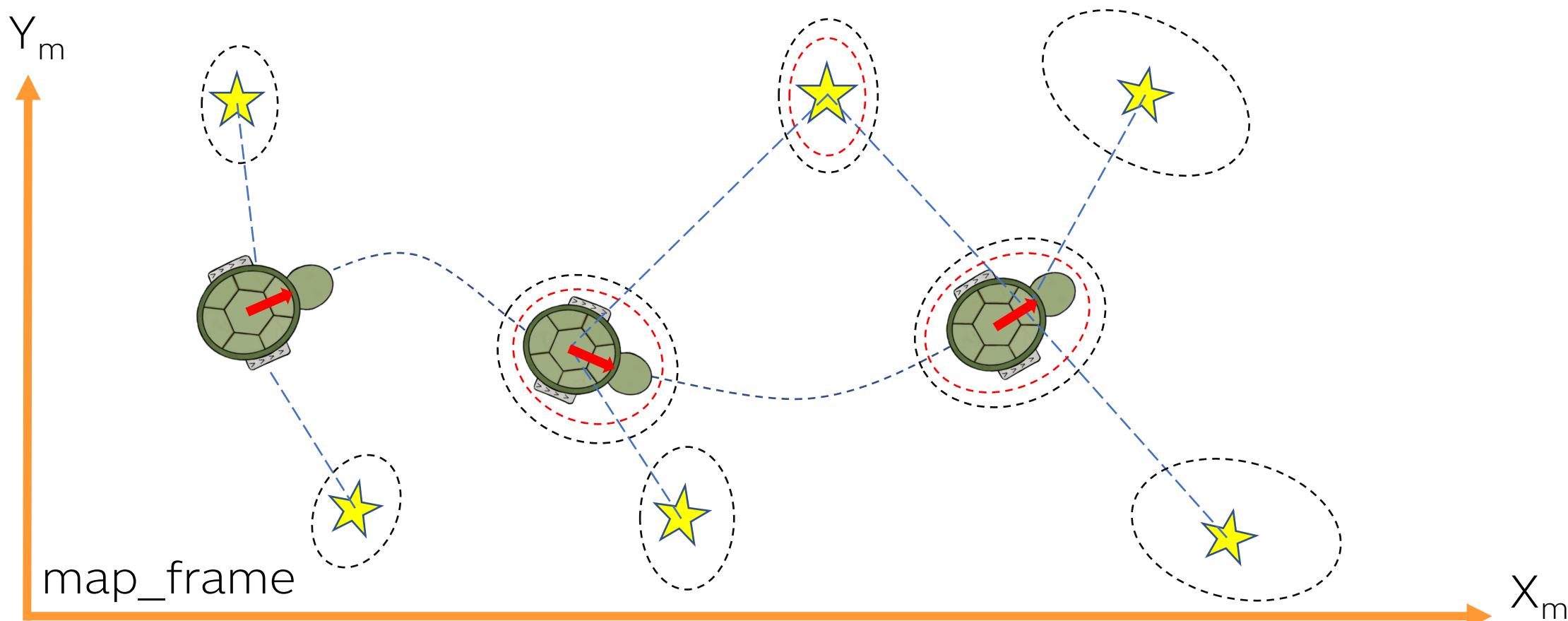
- Absolute robot poses
- Absolute landmark positions
- But only **relative** measurements of landmarks



# Creating a Map (SLAM Concept)

## Why is SLAM a hard problem ??

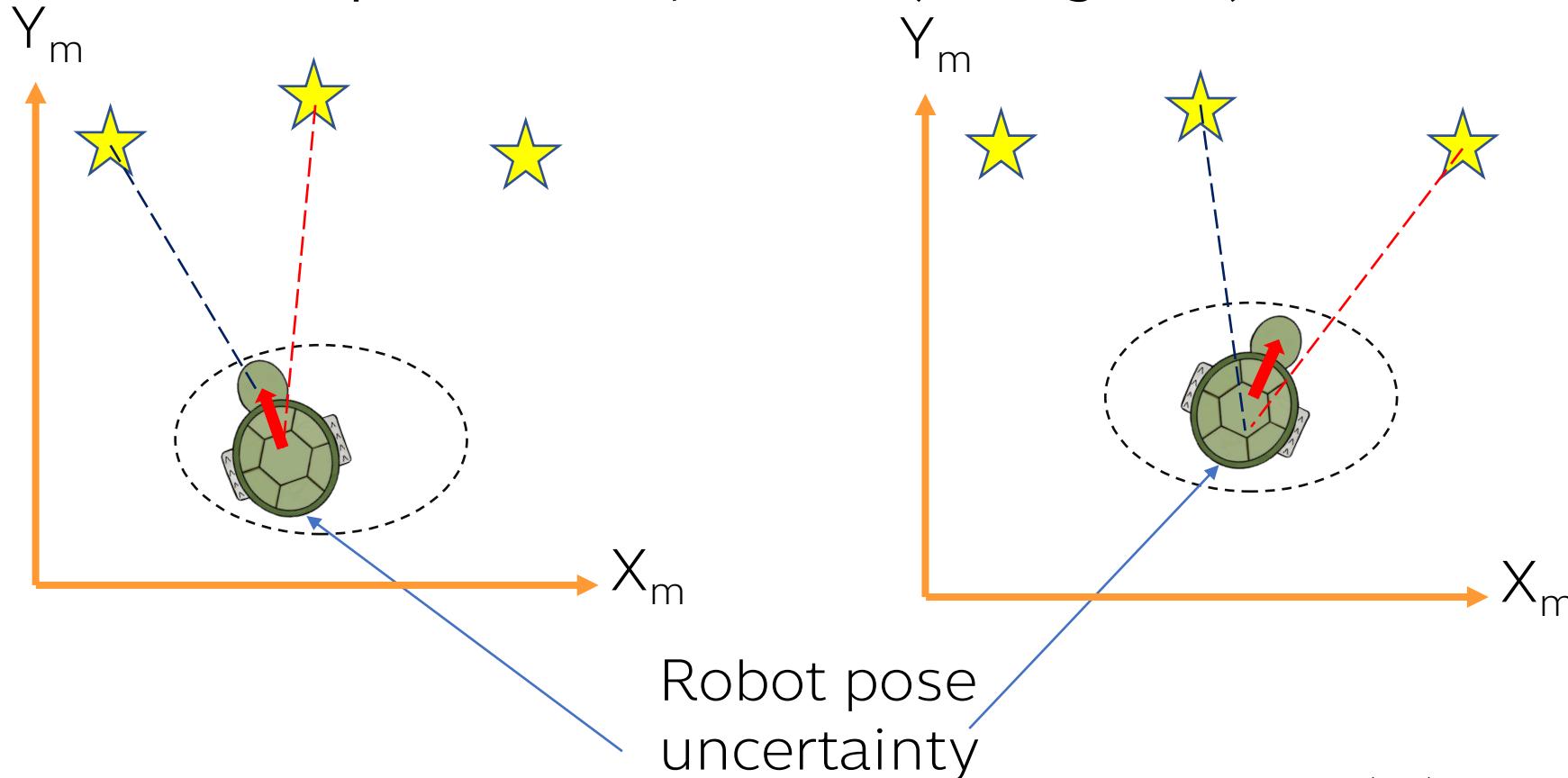
1. Robot path and map are both unknown
2. Errors in map and pose estimates correlated



# Creating a Map (SLAM Concept)

Why is SLAM a hard problem ??

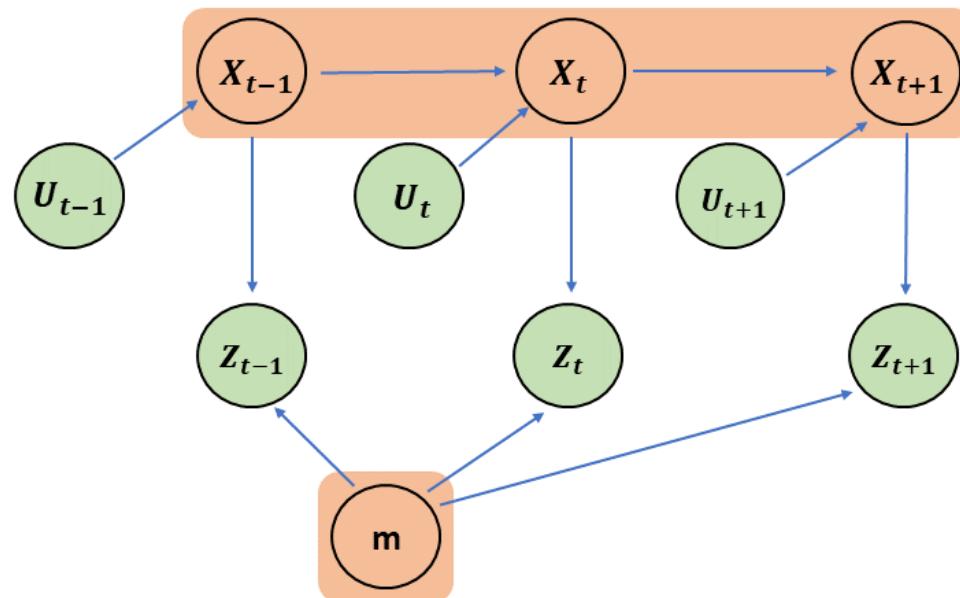
3. The mapping between observations and landmarks is unknown
4. Picking wrong data associations can have catastrophic consequences (divergence)



# Creating a Map (SLAM Concept)

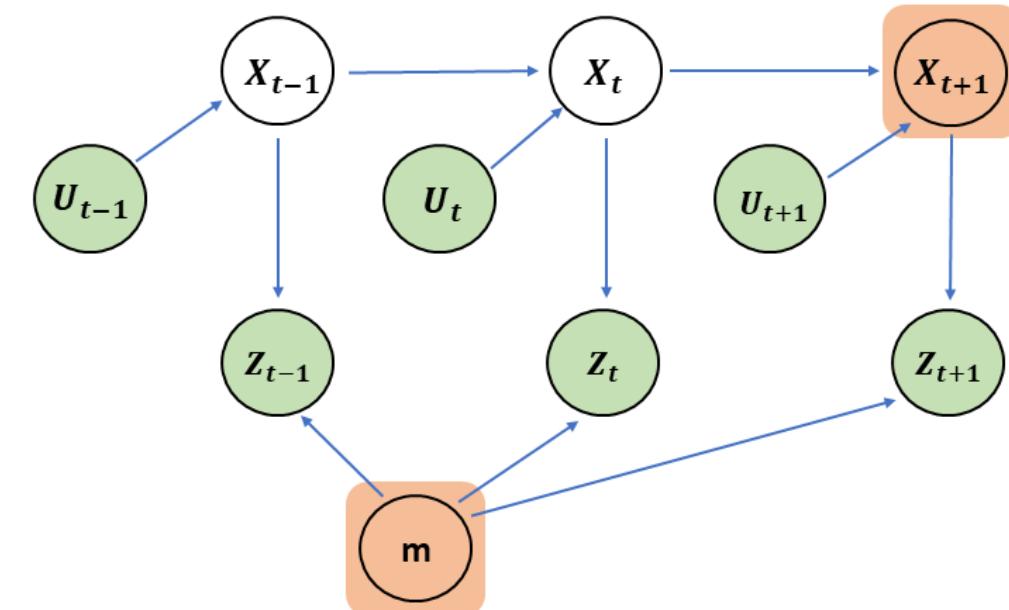
SLAM: Simultaneous Localization And Mapping

- Full SLAM:
- Online SLAM:



$$p(x_{0:t}, m | z_{1:t}, u_{1:t})$$

Estimate entire path and map

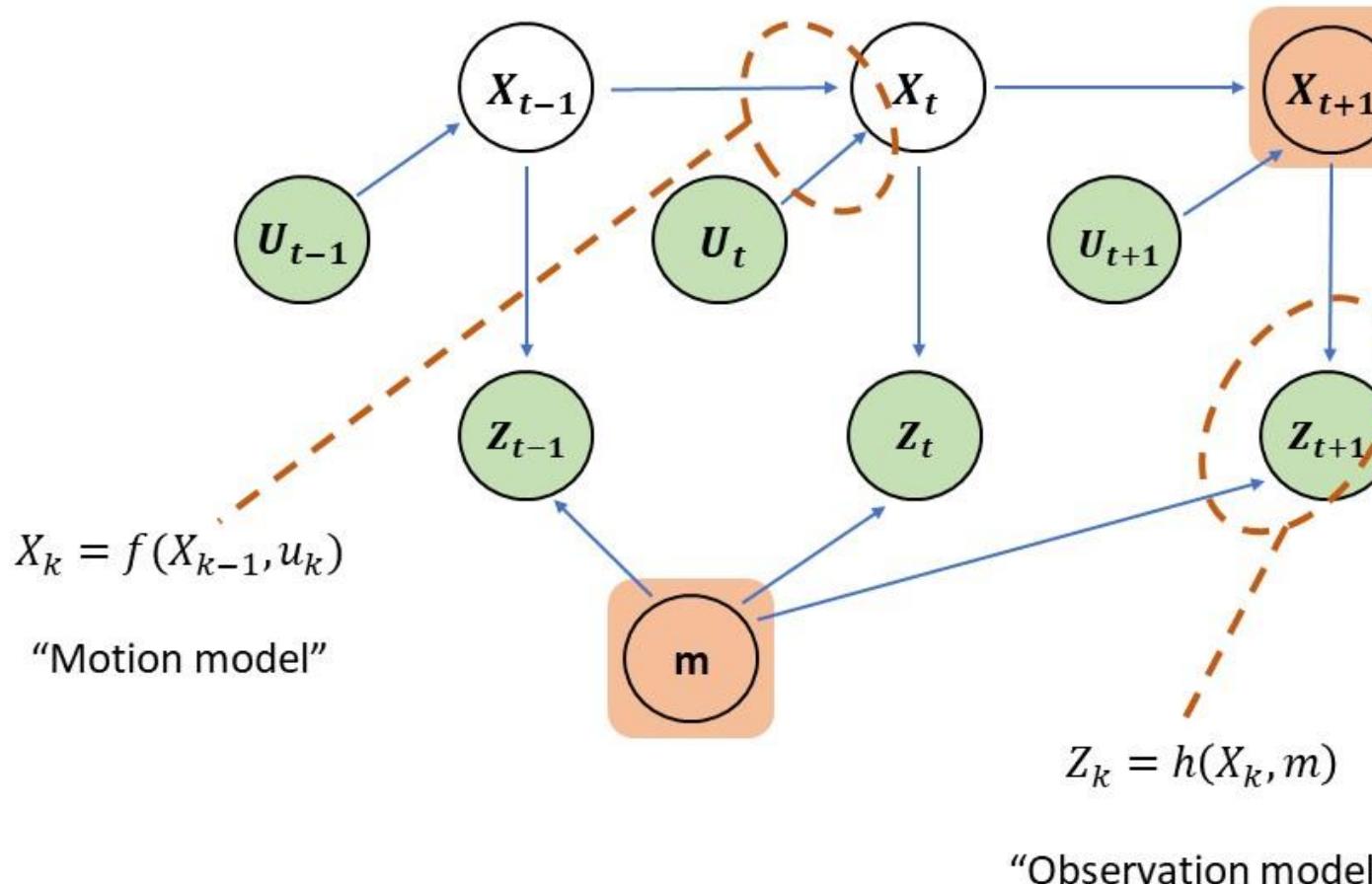


$$p(x_t, m | z_{1:t}, u_{1:t}) = \iint \dots \int p(x_{1:t}, m | z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1}$$

Estimates most recent pose and map

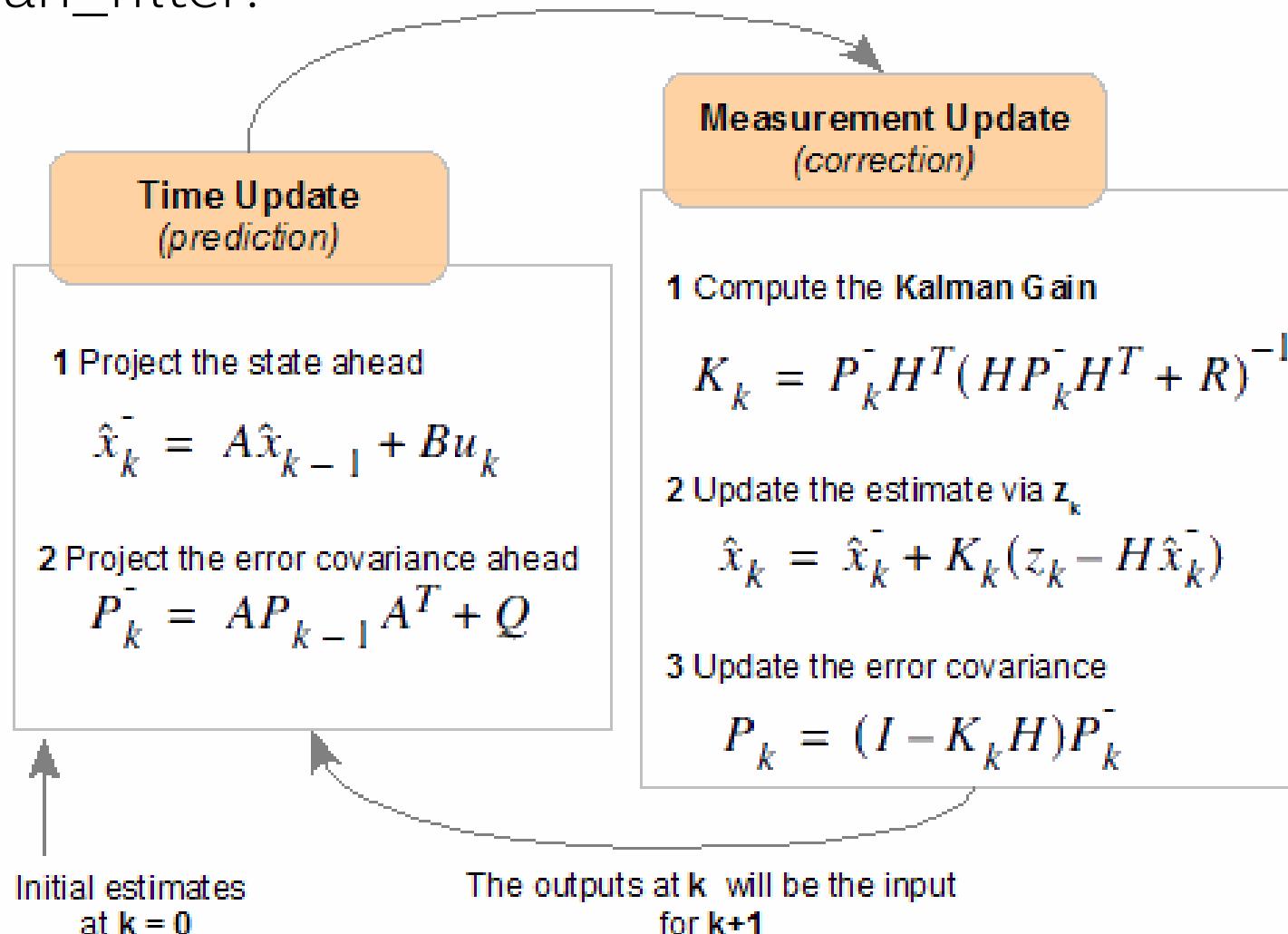
# Creating a Map (SLAM Concept)

## Motion and Observation Model



# Creating a Map (SLAM Concept)

Algorithm Kalman\_filter:



## EKF-SLAM:

### Localization

3x1 pose vector

3x3 cov. matrix

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \quad C_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 \end{bmatrix}$$

### SLAM

Landmarks simply extend the state. Growing state vector and covariance matrix

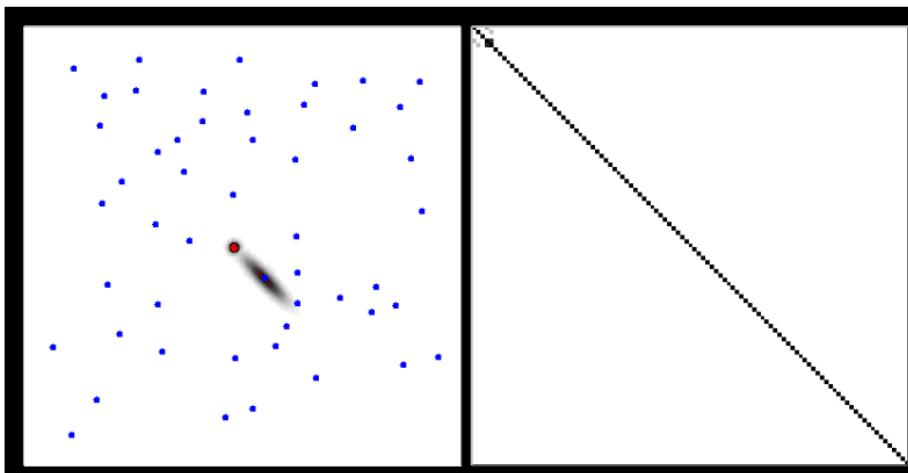
$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}_k \quad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} \\ C_{M_1 R} & C_{M_1} & C_{M_1 M_2} & \cdots & C_{M_1 M_n} \\ C_{M_2 R} & C_{M_2 M_1} & C_{M_2} & \cdots & C_{M_2 M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{M_n R} & C_{M_n M_1} & C_{M_n M_2} & \cdots & C_{M_n} \end{bmatrix}_k$$

# Creating a Map (SLAM Concept)

Integrating New Landmarks

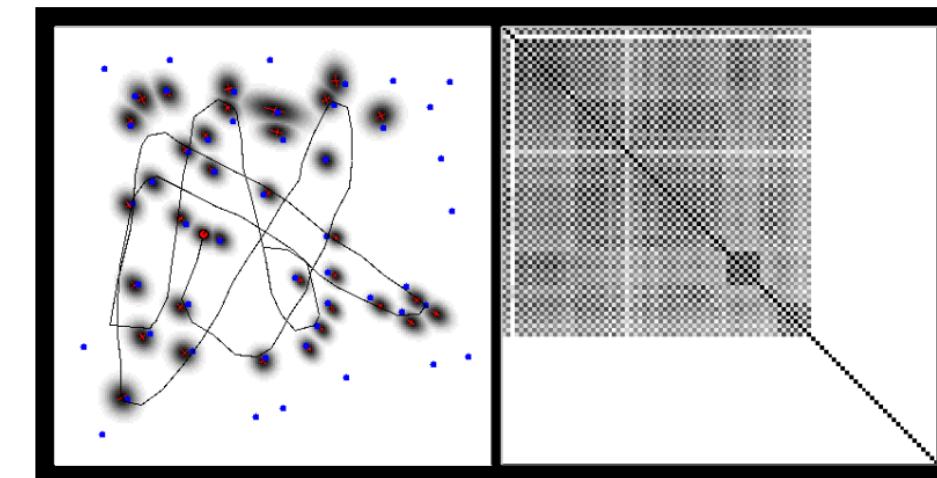
$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \\ \mathbf{m}_{n+1} \end{bmatrix}_k \quad C_k = \begin{bmatrix} C_R & C_{RM_1} & C_{RM_2} & \cdots & C_{RM_n} & C_{RM_{n+1}} \\ C_{M_1 R} & C_{M_1} & C_{M_1 M_2} & \cdots & C_{M_1 M_n} & C_{M_1 M_{n+1}} \\ C_{M_2 R} & C_{M_2 M_1} & C_{M_2} & \cdots & C_{M_2 M_n} & C_{M_2 M_{n+1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{M_n R} & C_{M_n M_1} & C_{M_n M_2} & \cdots & C_{M_n} & C_{M_n M_{n+1}} \\ C_{M_{n+1} R} & C_{M_{n+1} M_1} & C_{M_{n+1} M_2} & \cdots & C_{M_{n+1} M_n} & C_{M_{n+1}} \end{bmatrix}_k$$

Initial state (number of landmark = 0)



Map

State N (number of landmark >> 0)



Map

Correlation matrix

## EKF-SLAM: Summary

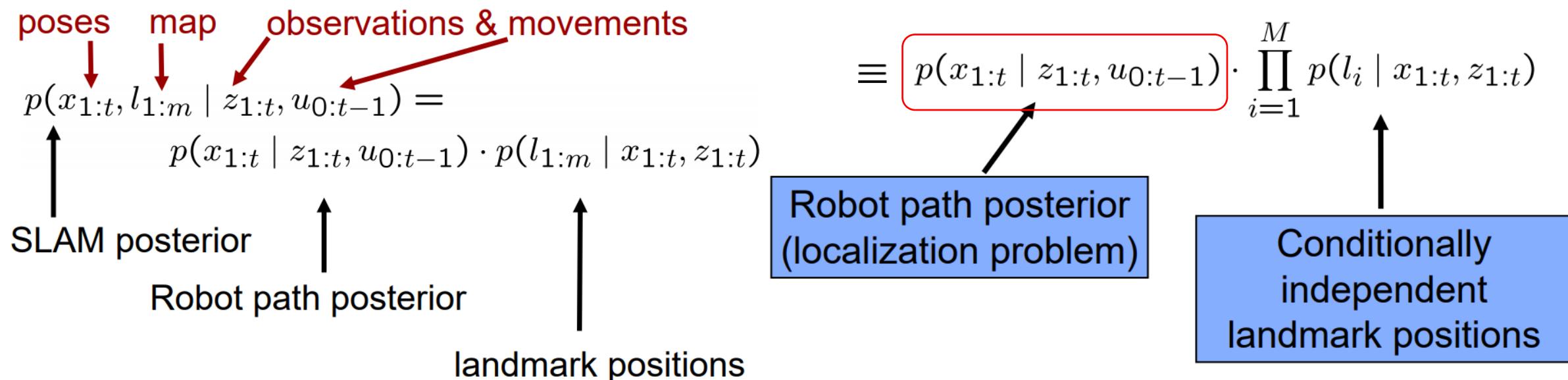
- The first SLAM solution
- Convergence proof for **linear Gaussian case**
- Can diverge if nonlinearities are large (and the reality is nonlinear...)
- Problem: becomes computationally intractable for large maps!
- Cost per step: quadratic in  $n$ , the number of landmarks:  $O(n^2)$
- Total cost to build a map with  $n$  landmarks:  $O(n^3)$
- Memory consumption:  $O(n^2)$

## SLAM Techniques

- EKF SLAM
- FastSLAM
- Graph-based SLAM
- ETC.

# Creating a Map (SLAM Concept)

ROS slam\_gmapping package  
based on Grid Mapping with Particle Filter (PF)



# Creating a Map (SLAM Concept)

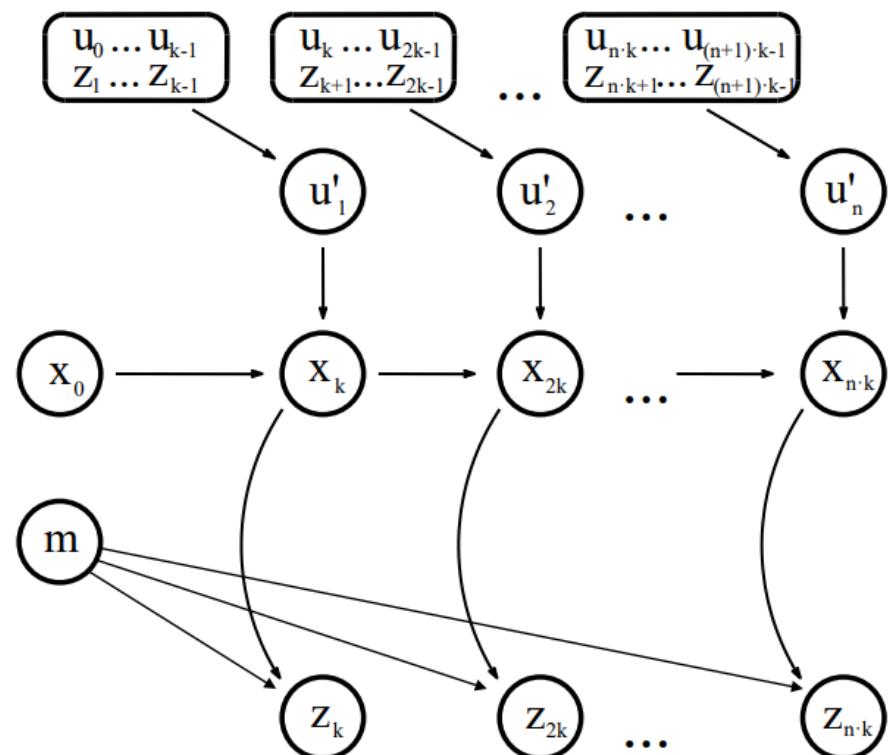
## Rao-Blackwellization

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}) = \\ p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(m \mid x_{1:t}, z_{1:t})$$

This is localization, use MCL

Use the pose estimate  
from the MCL and apply  
mapping with known poses

## With Scan-Matching



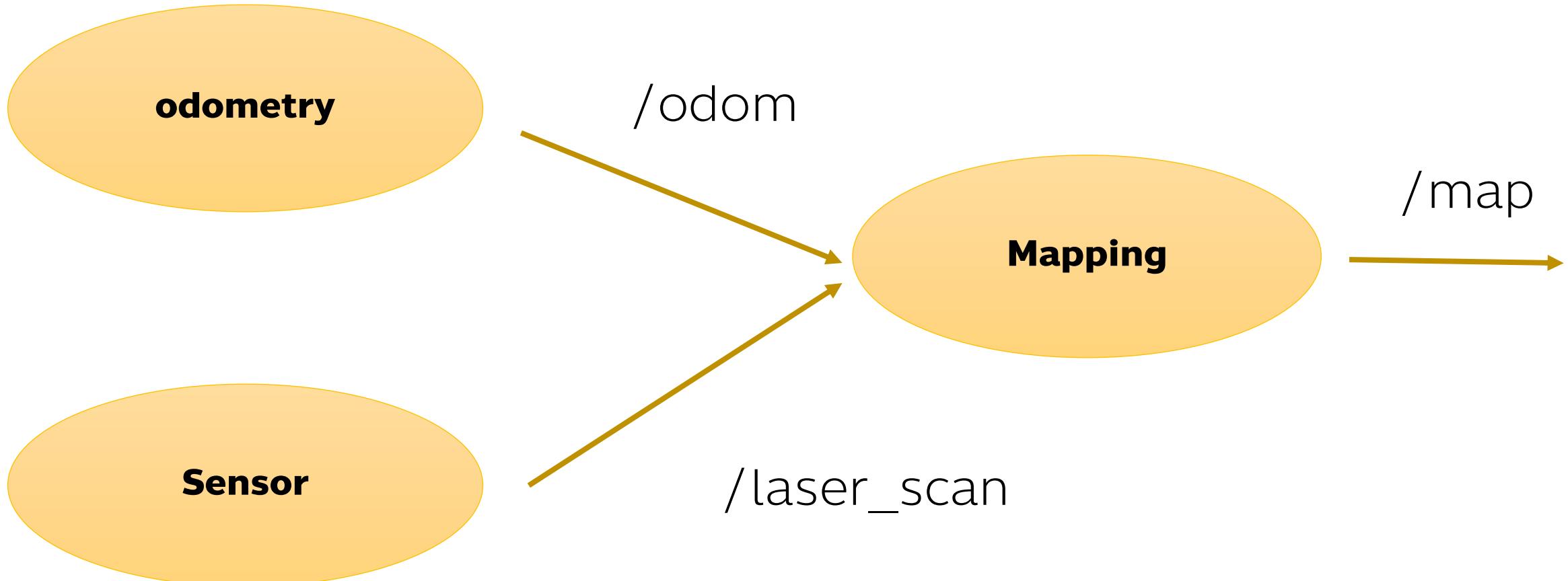
for more information please see:

<http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf>  
<http://wiki.ros.org/gmapping>



# Mapping and Navigation Concept

## Mapping



# Localization



KU

KASETSART  
UNIVERSITY

# Localization

## Adaptive Monte Carlo localization (AMCL)

Monte Carlo localization (MCL), also known as **particle filter localization**, is an algorithm for robots to localize using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment

# Localization

AMCL dynamically adjusts the number of particles based on KL-distance to ensure that the particle distribution converge to the true distribution of robot state based on all past sensor and motion measurements with high probability.

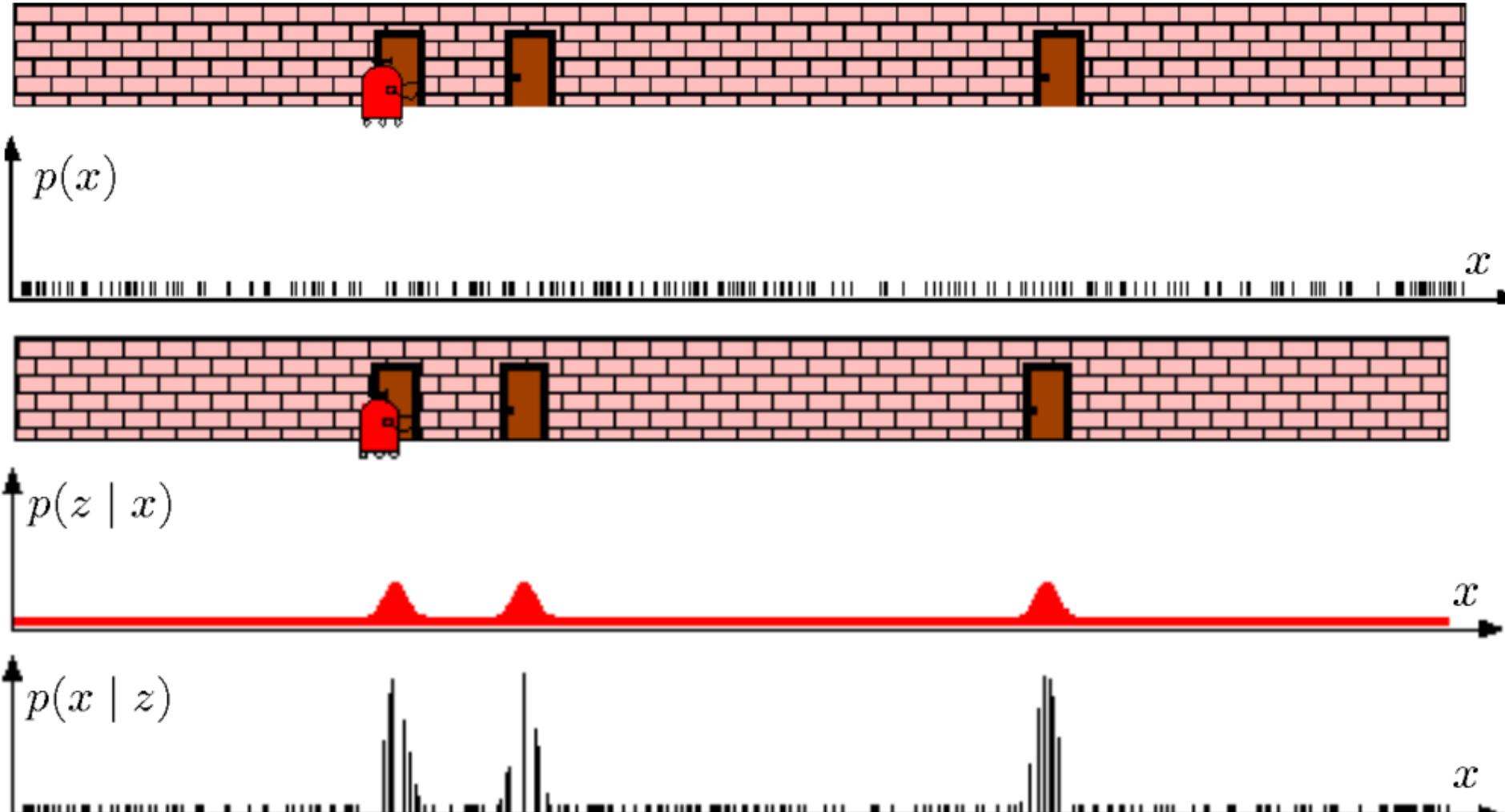
# Localization

## Particle Filters (PF)

- Sample the next generation for particles using the proposal distribution (Often use the motion model)
- Compute the importance weights
- Resampling: “Replace unlikely samples by more likely ones”

# Localization

Sensor Information: Importance Sampling

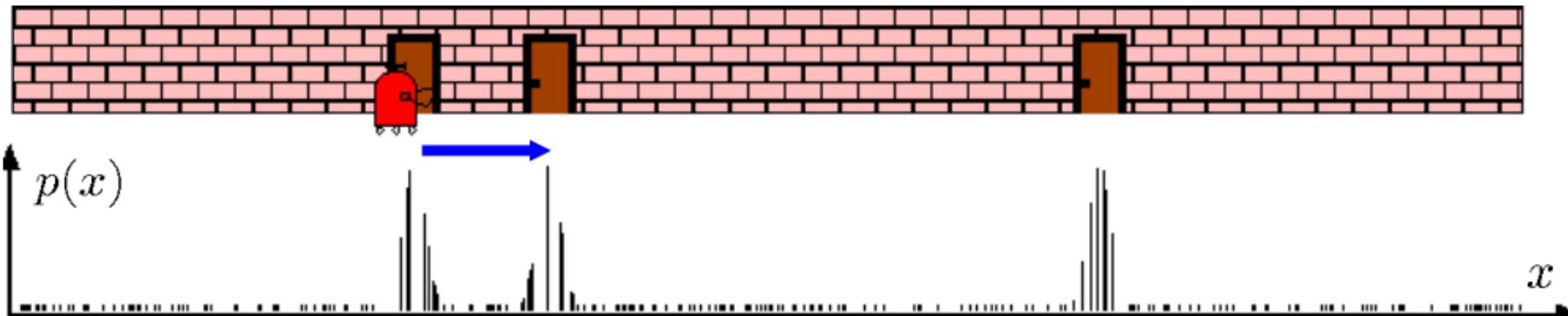


Sensor Model

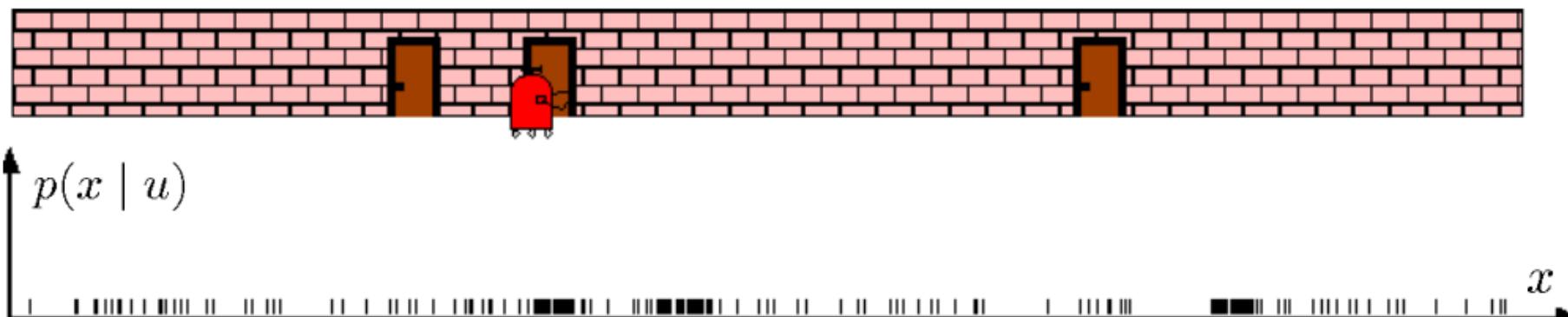
Weighting

# Localization

## Robot Motion



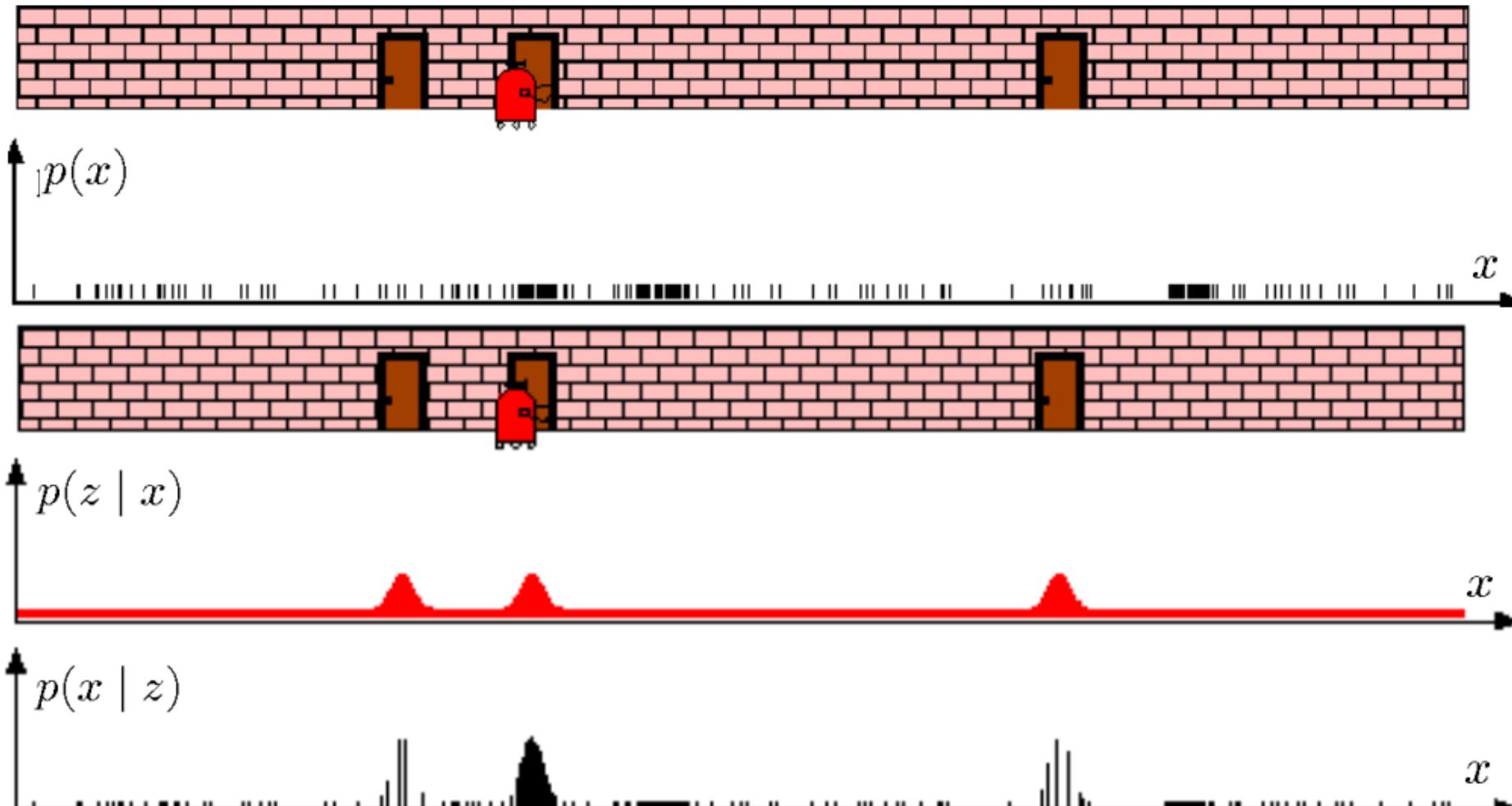
Sampling  
Motion Model



Resampling

# Localization

Sensor Information: Importance Sampling

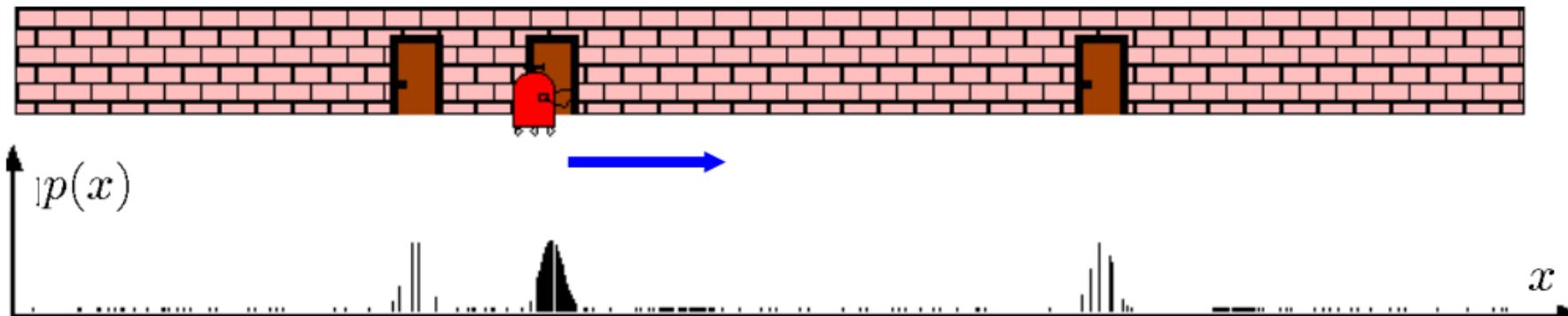


Sensor Model

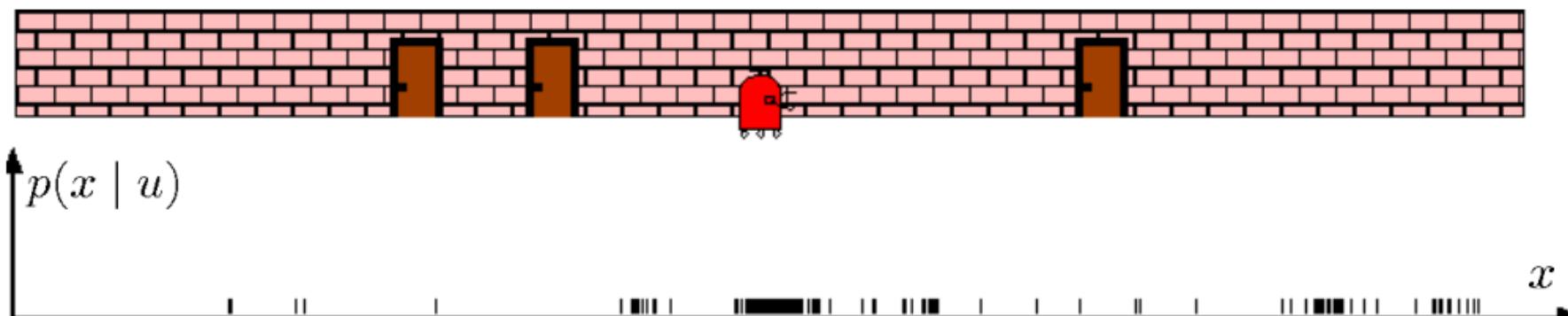
Weighting

# Localization

## Robot Motion



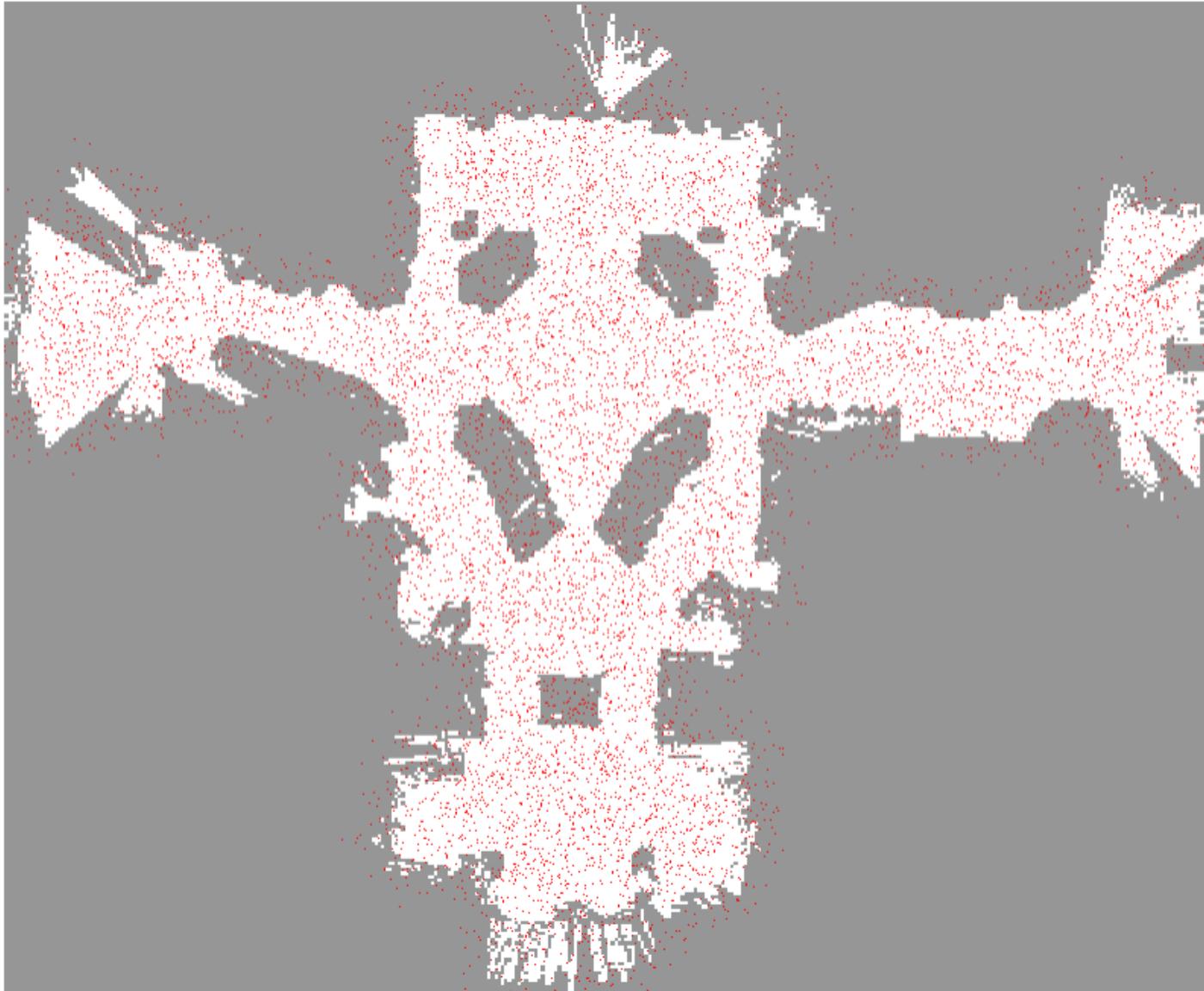
Sampling  
Motion Model

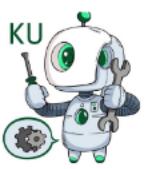


Resampling

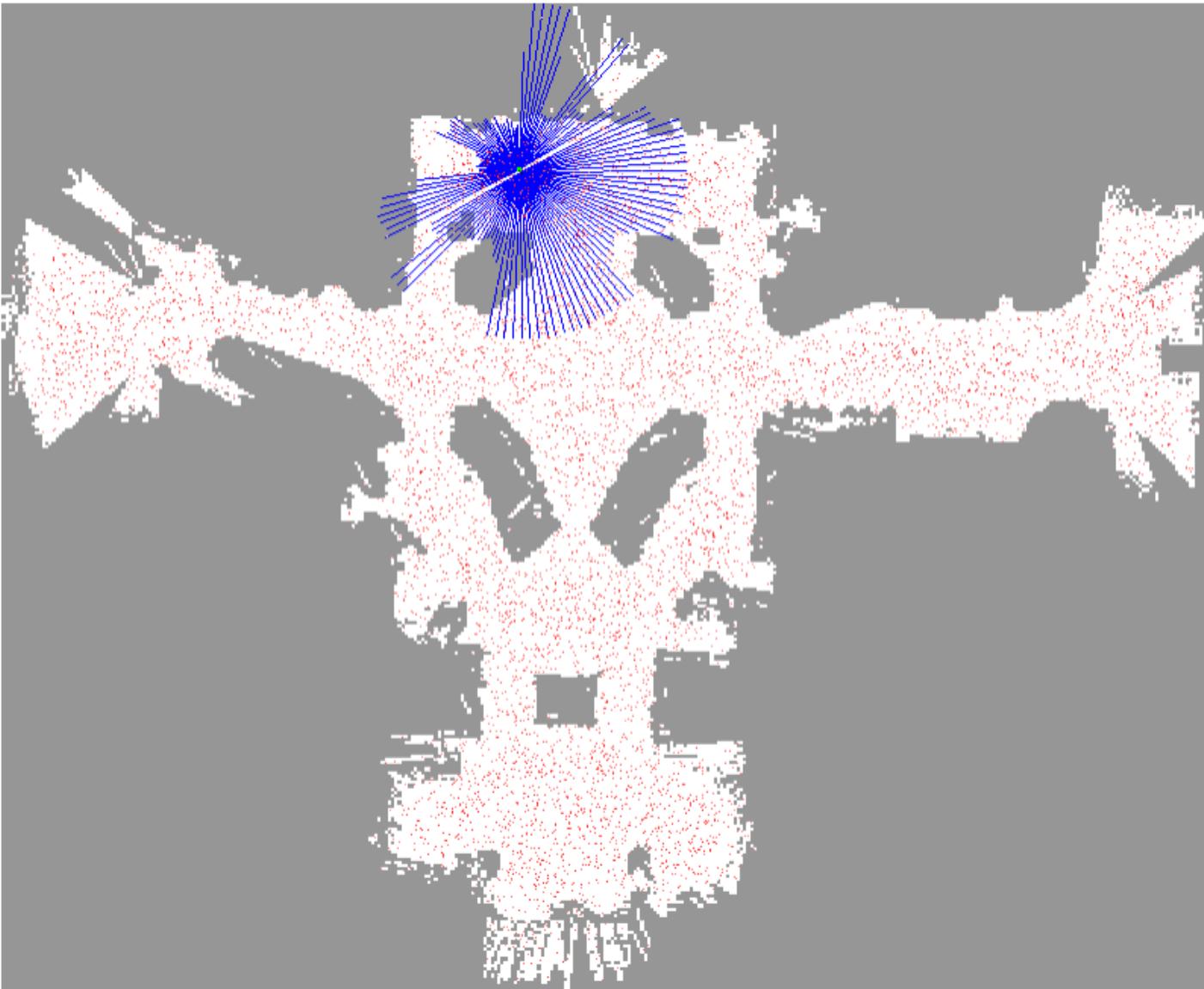


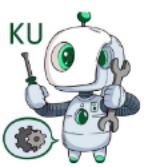
# Localization



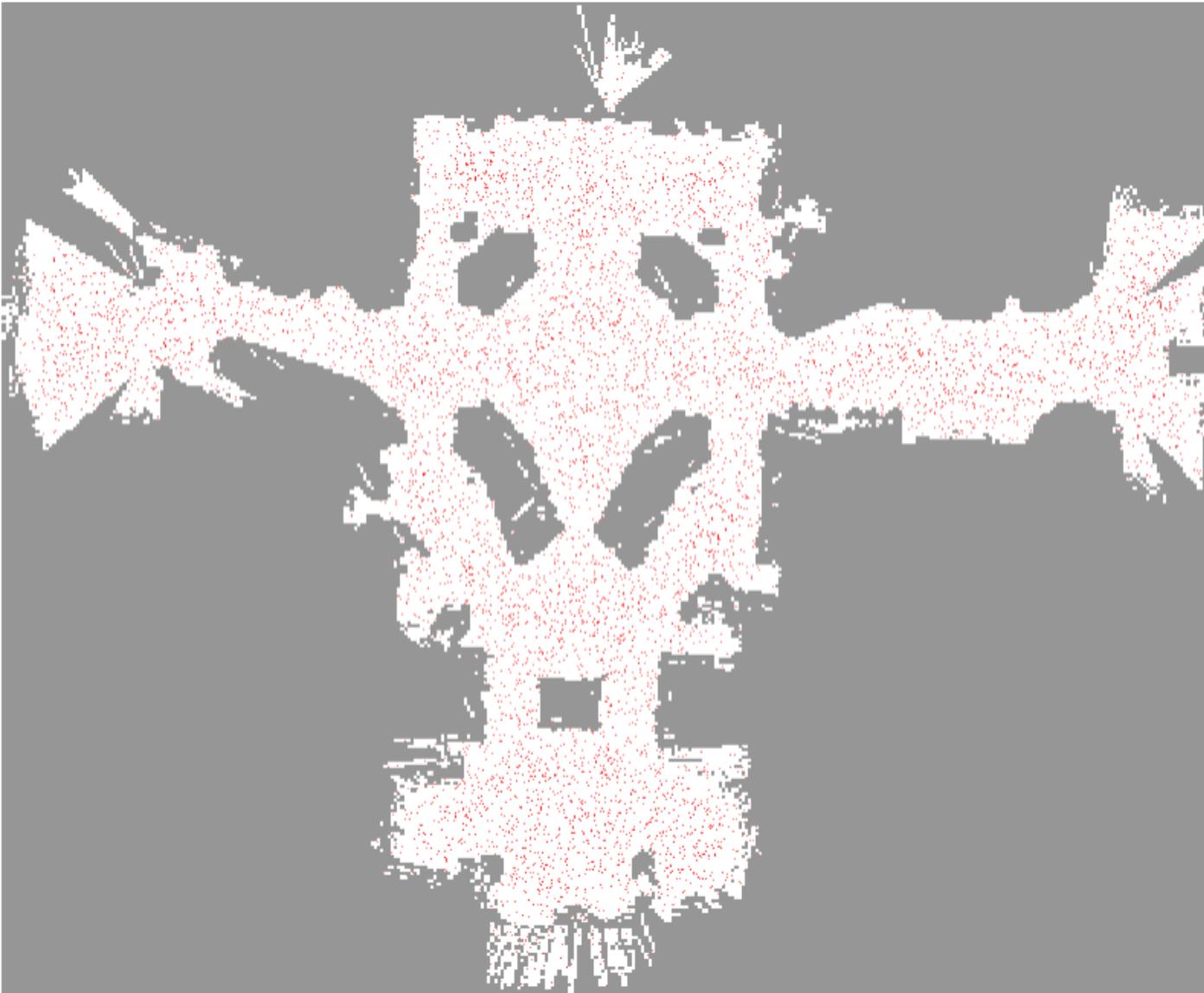


# Localization

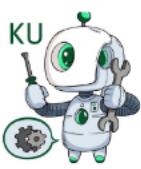




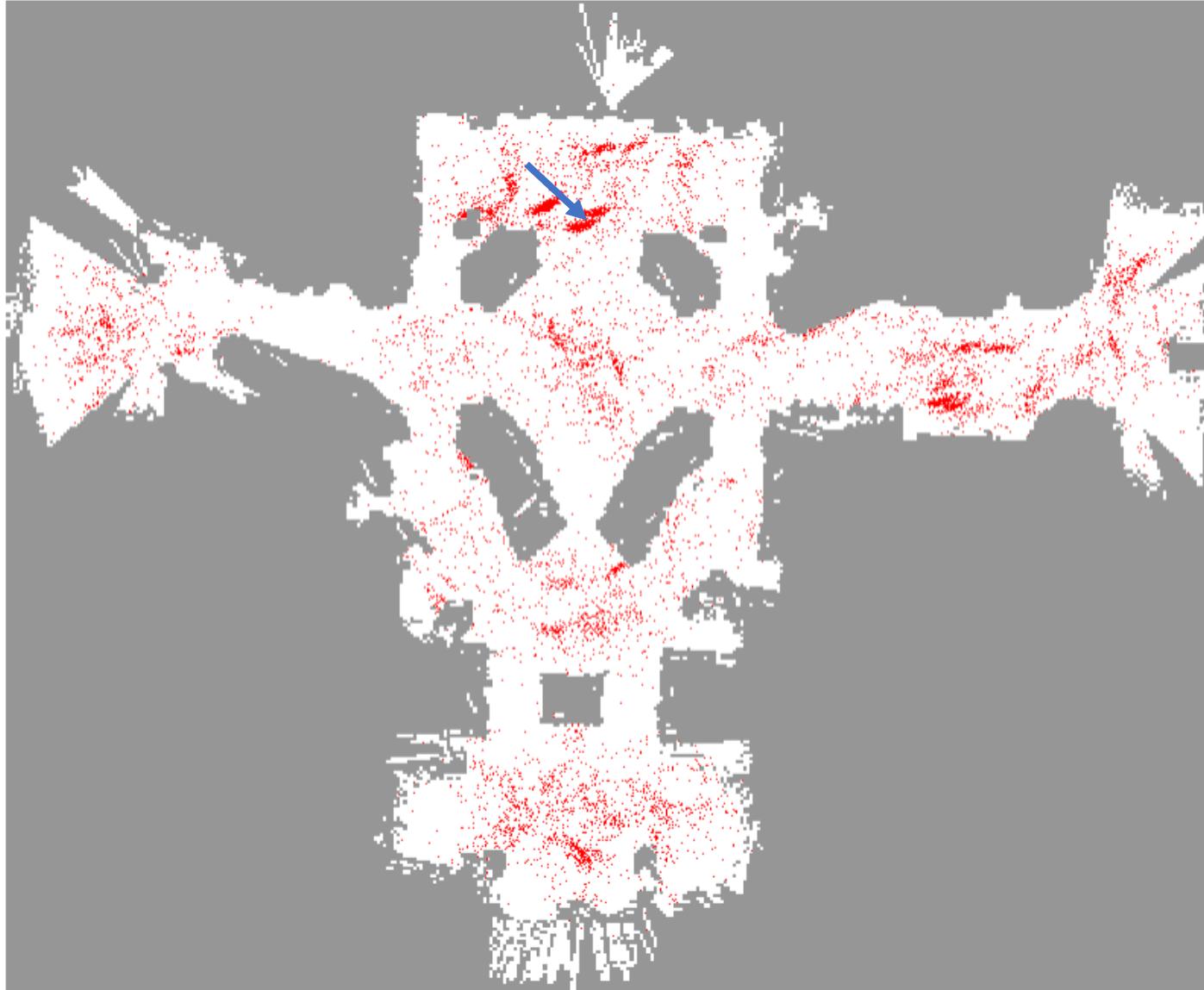
# Localization



Update  
Weight



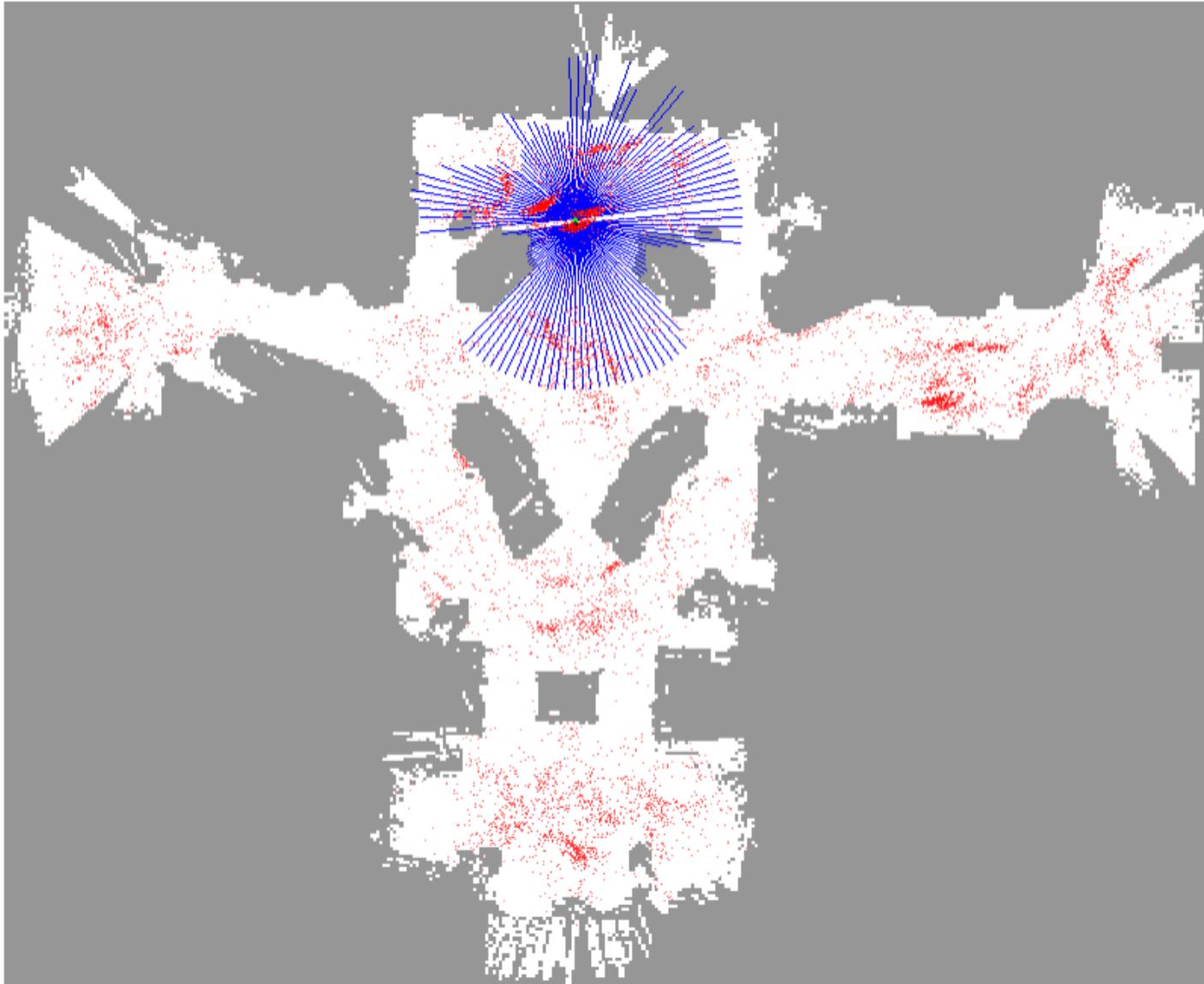
# Localization



Sampling  
Motion Model

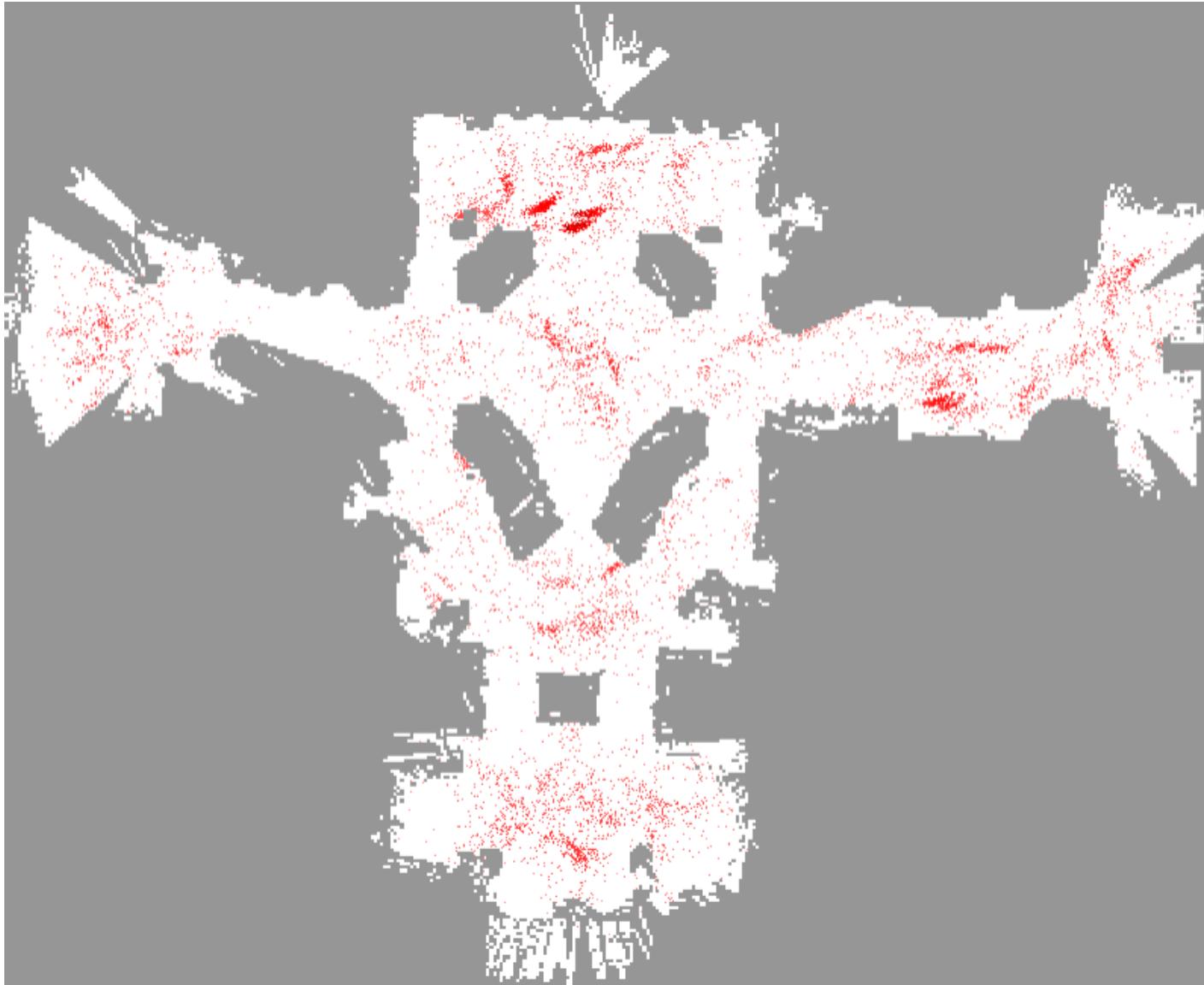


# Localization

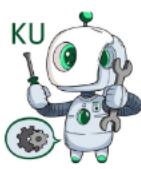




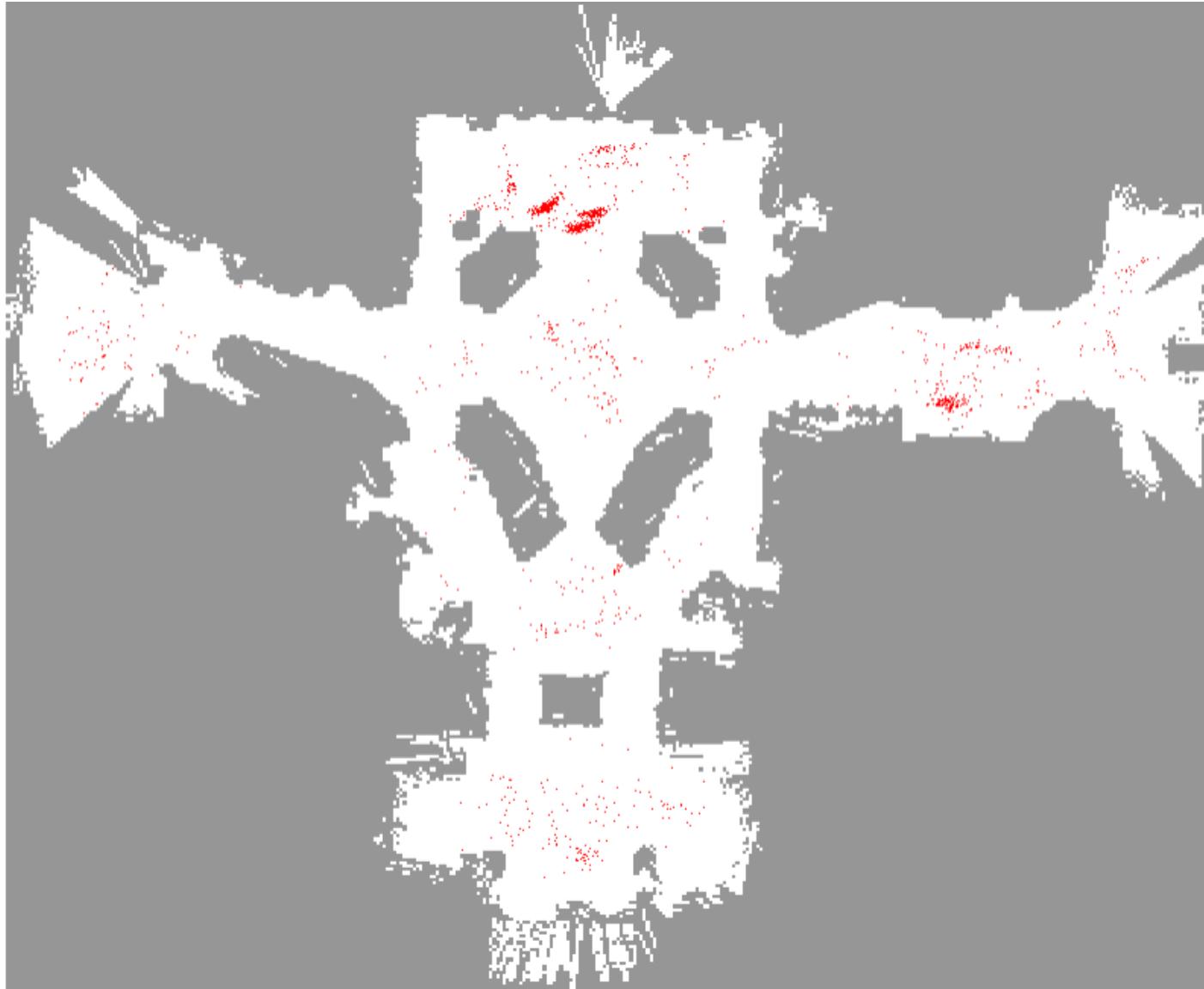
# Localization



Update  
Weight

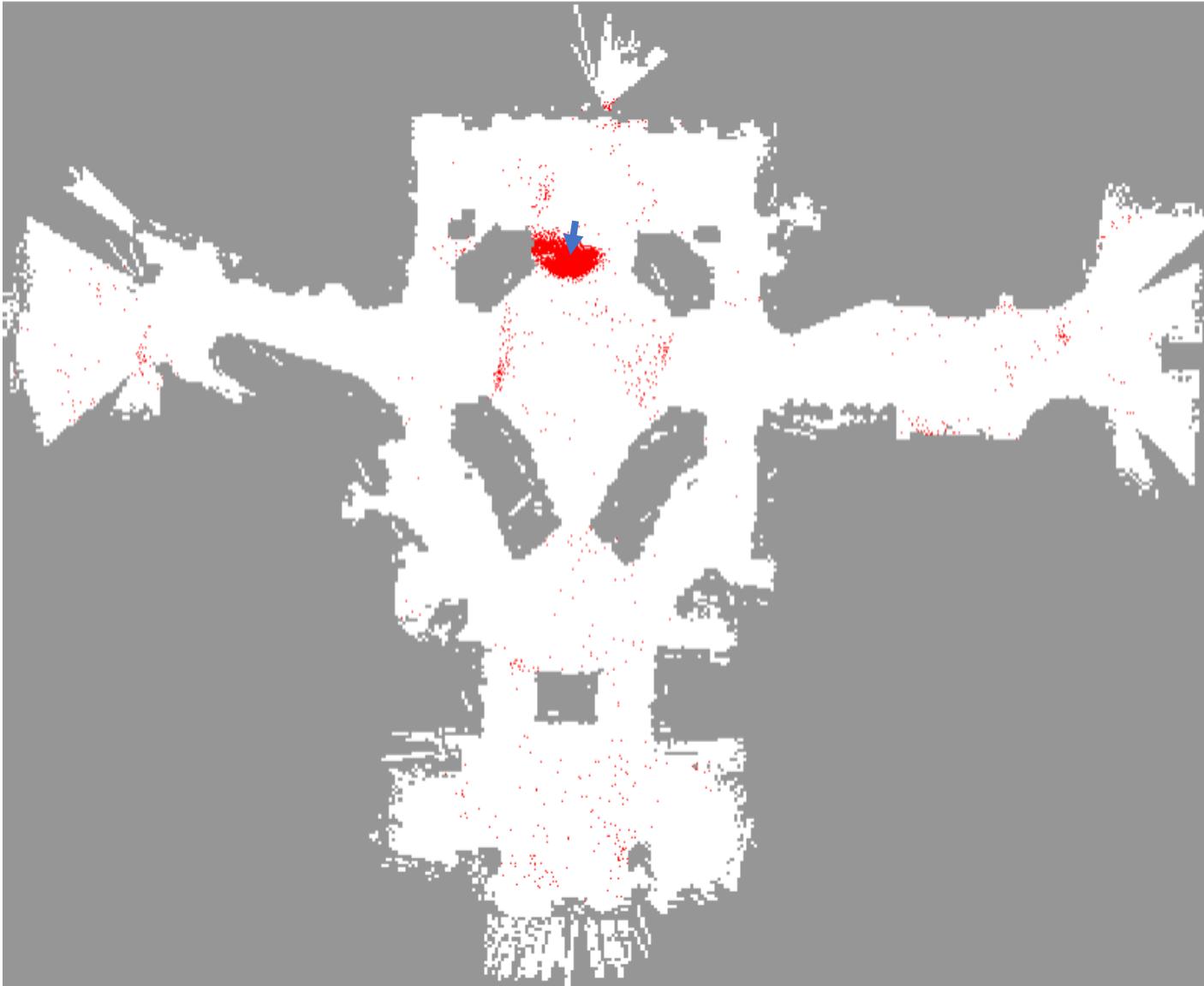


# Localization



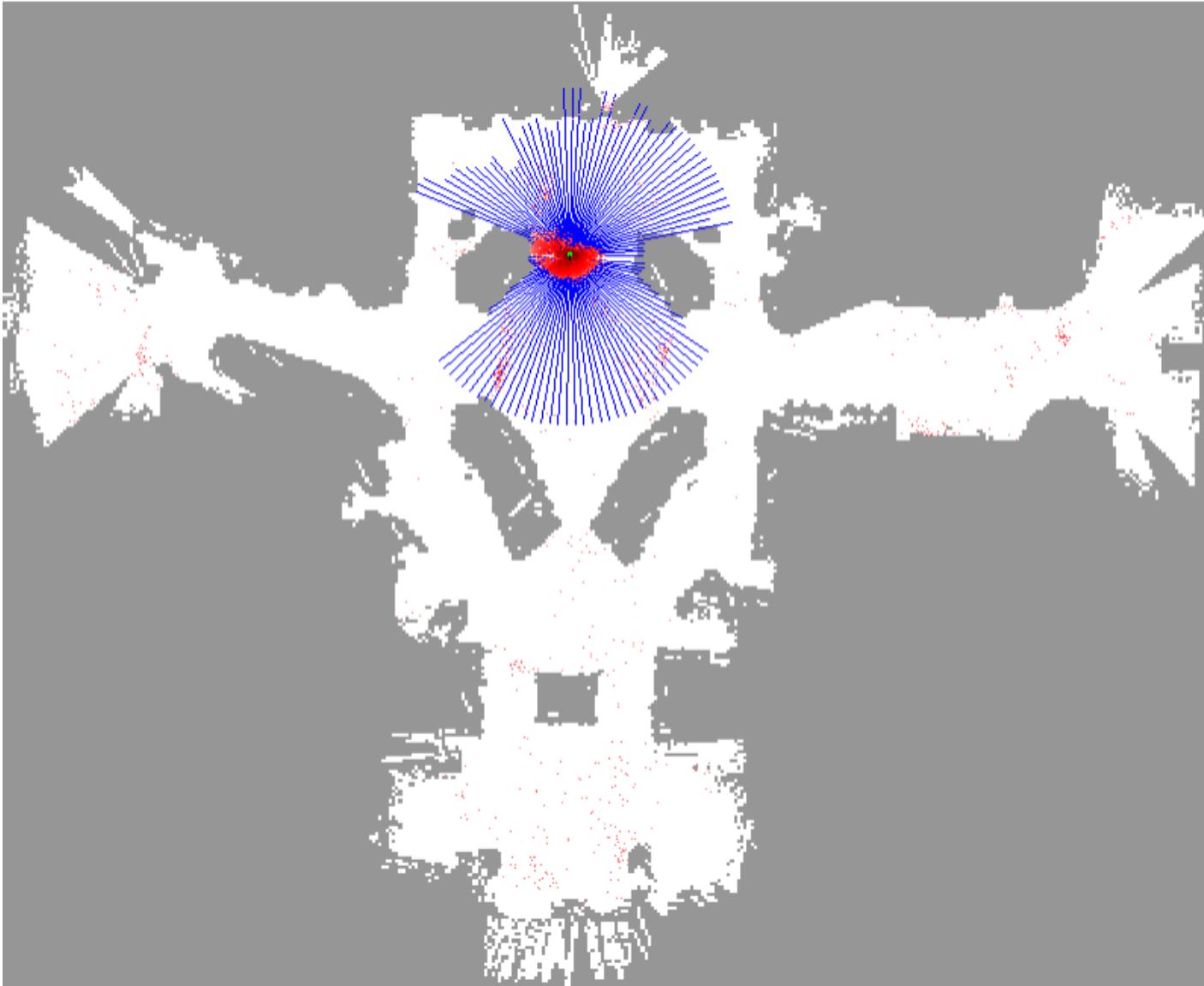
Resampling  
Reset Weight

# Localization

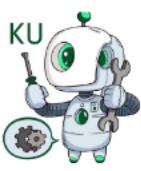


Sampling  
Motion Model

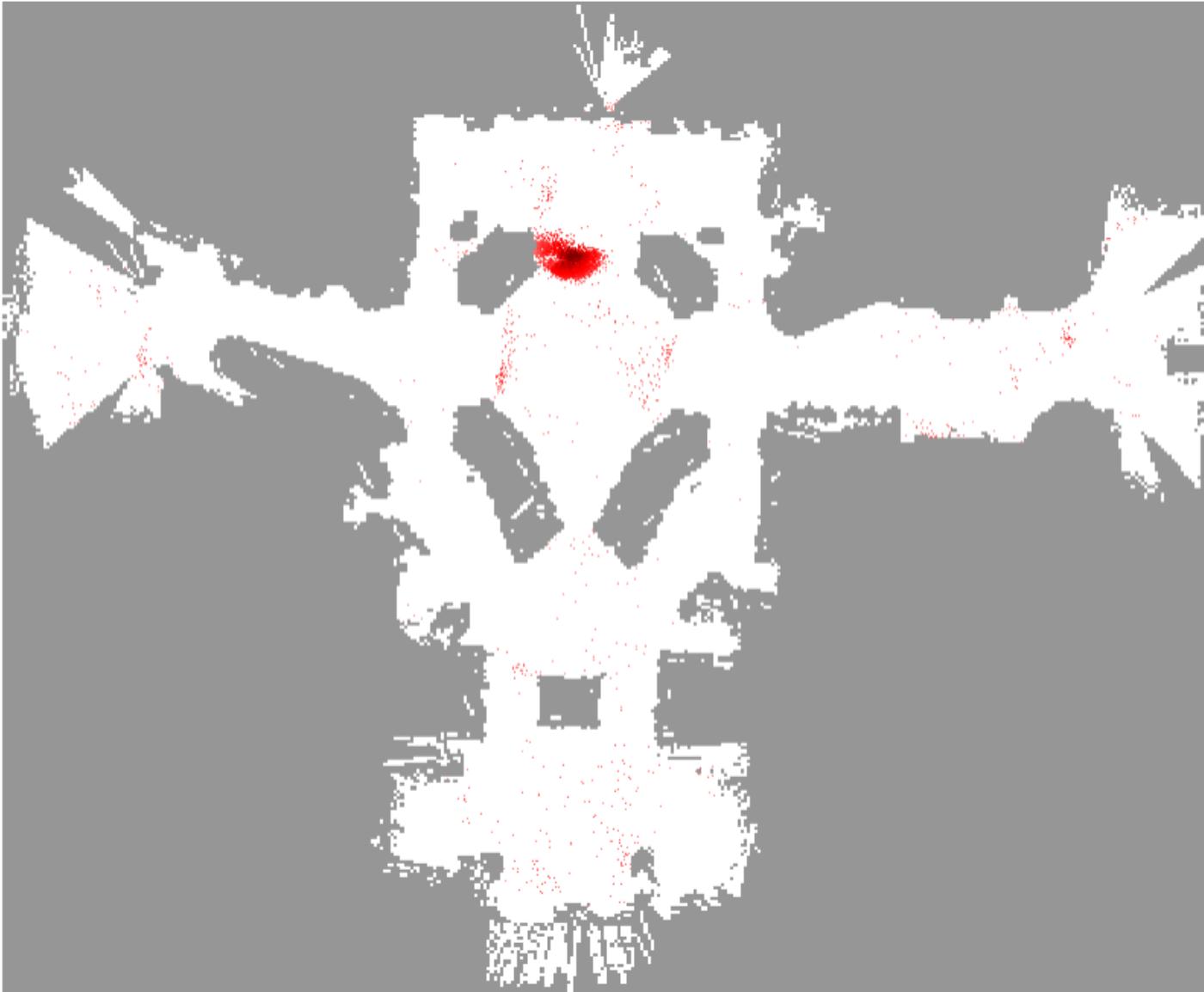
# Localization



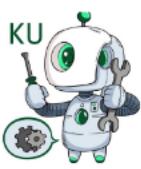
Sensor Model



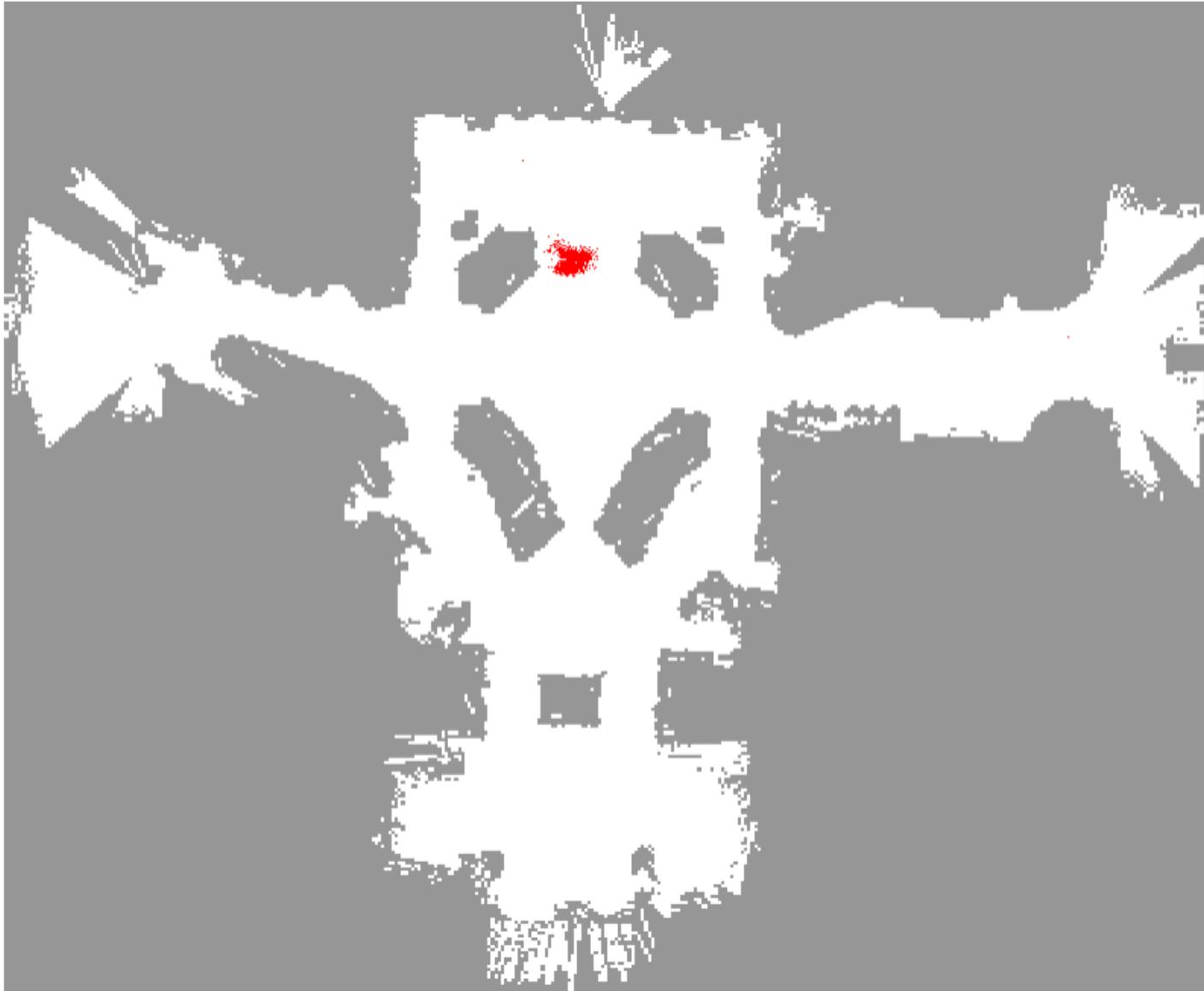
# Localization



Update  
Weight



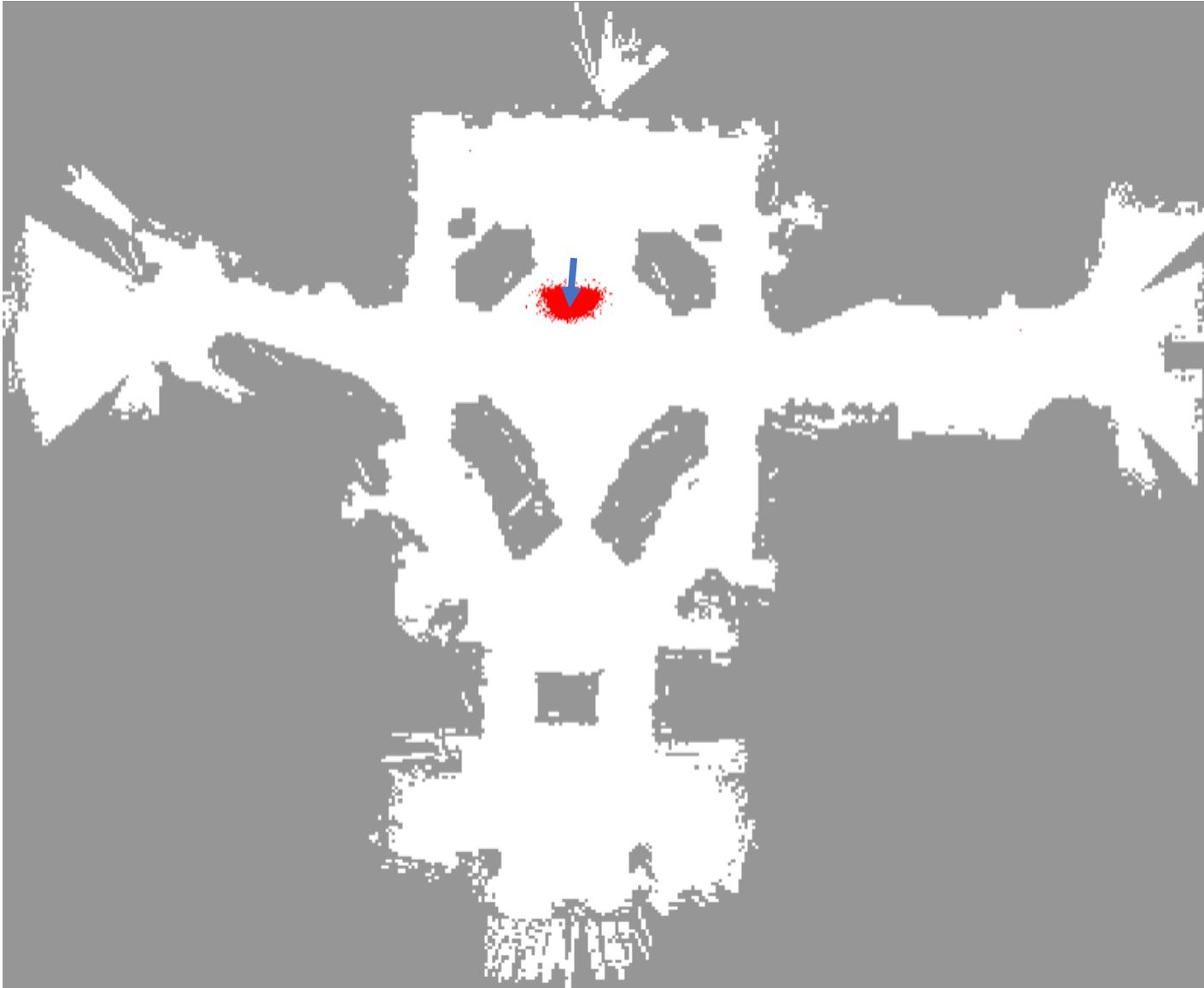
# Localization



Resampling  
Reset Weight



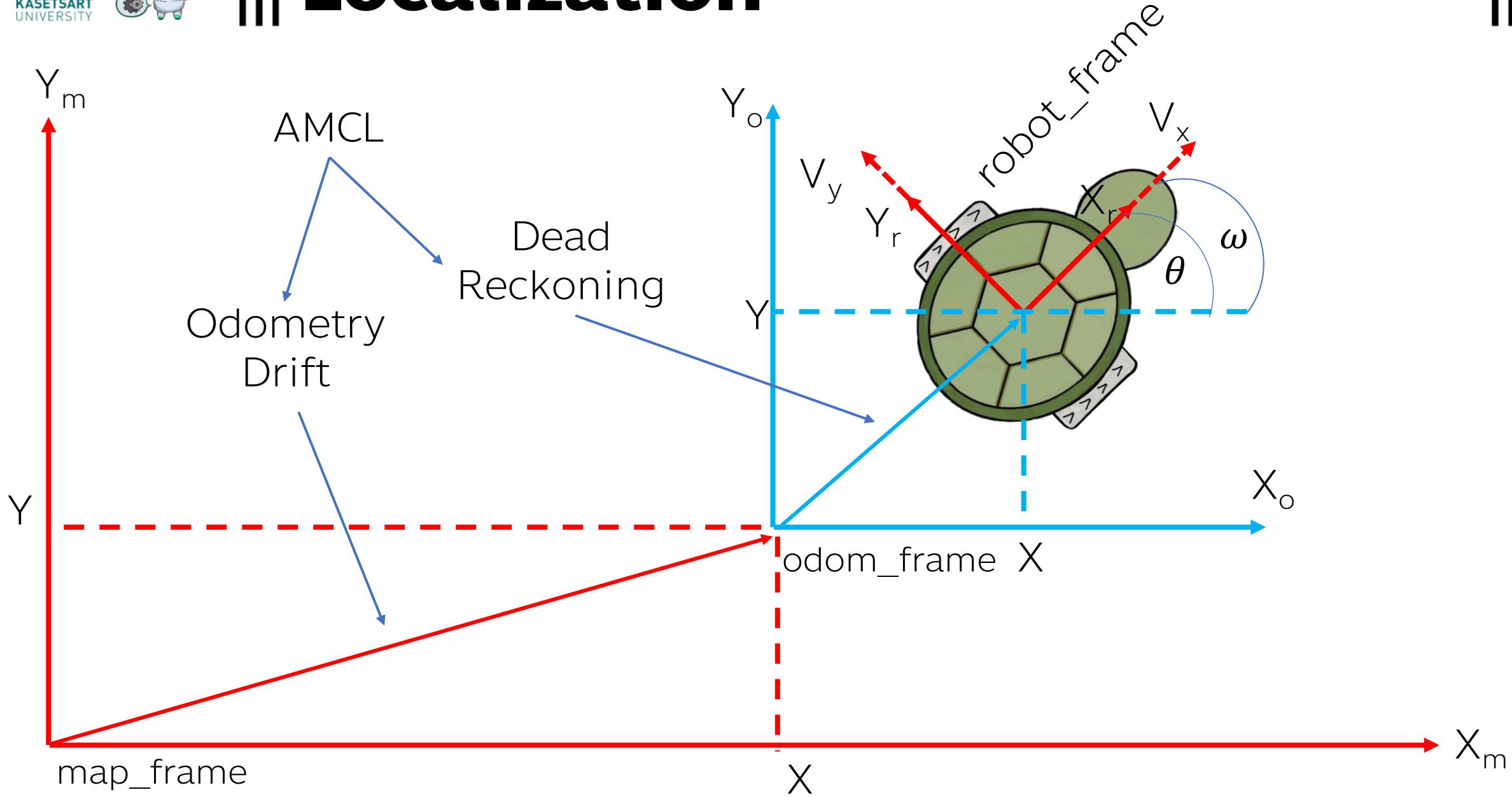
# Localization



Sampling  
Motion Model

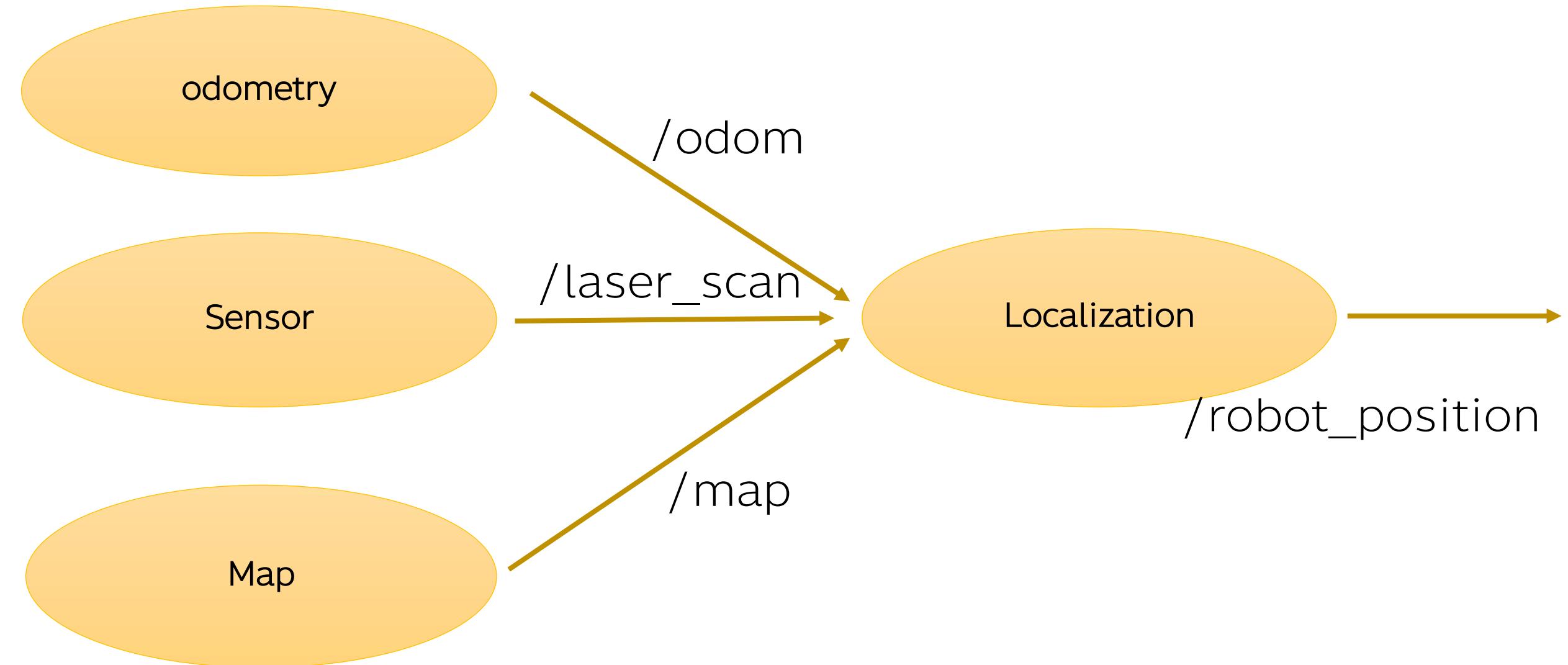


# Localization

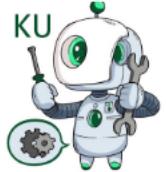




# Localization



# Mapping and Localization Example



# Creating a Map

Run following command On the Turtlebot:

Bring up the robot

```
$ roslaunch turtlebot_bringup minimal.launch
```

Run the gmapping demo app

```
$ roslaunch turtlebot_navigation gmapping_demo.launch
```

```
Registering First Scan
[ INFO] [1589187963.541649911]: Resizing costmap to 32 X 32 at 0.050000 m/pix
[ INFO] [1589187963.641544626]: Received a 32 X 32 map at 0.050000 m/pix
[ INFO] [1589187963.652621787]: Using plugin "obstacle_layer"
[ INFO] [1589187963.659199401]: Subscribed to Topics: scan bump
[ INFO] [1589187963.732045008]: Using plugin "inflation_layer"
[ INFO] [1589187963.822148758]: Using plugin "obstacle_layer"
[ INFO] [1589187963.826630389]: Subscribed to Topics: scan bump
[ INFO] [1589187963.902166812]: Using plugin "inflation_layer"
[ INFO] [1589187963.974495217]: Created local_planner dwa_local_planner/DWAPlannerROS
[ INFO] [1589187963.978729699]: Sim period is set to 0.20
[ INFO] [1589187964.792233685]: Stopping device RGB and Depth stream flush.
[ INFO] [1589187964.979560588]: Recovery behavior will clear layer obstacles
[ INFO] [1589187964.988219664]: Recovery behavior will clear layer obstacles
[ INFO] [1589187965.050636841]: odom received!
```

# Creating a Map

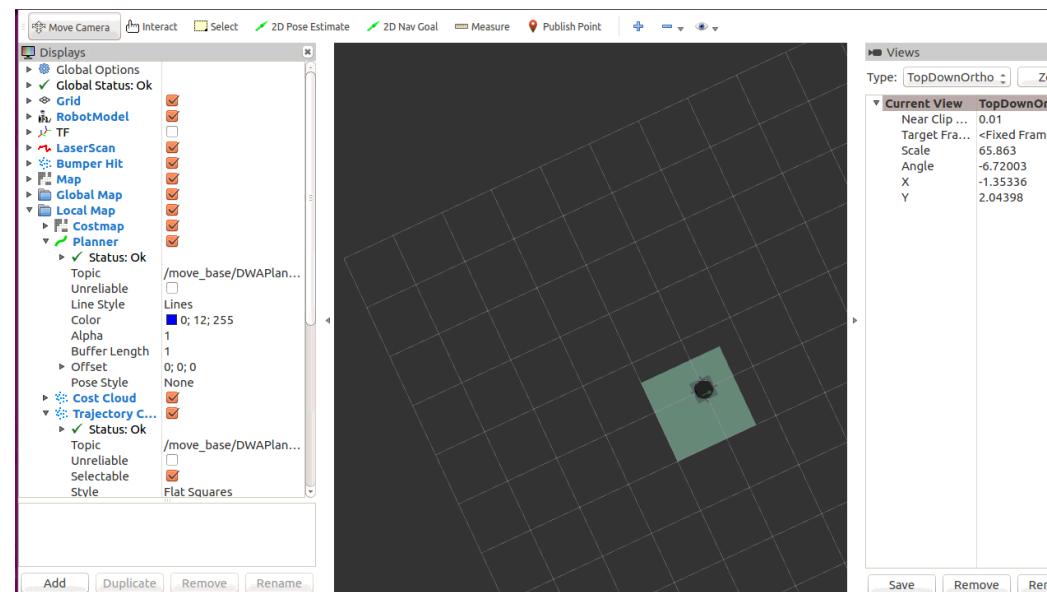
Run following command On Workstation:

Run the robot visualization to see the collecting map:

```
$ rosrun turtlebot_rviz_launchers view_navigation.launch
```

Run controller node

```
$ rosrun turtlebot_teleop logitech.launch
```



# Creating a Map

Save the map to file:

```
$ rosrun map_server map_saver -f /tmp/my_map
```

```
$ cp /tmp/my_map* ~
```

```
user@ros-workshop:~$ rosrun map_server map_saver -f /tmp/my_map
[ INFO] [1589188055.299165053]: Waiting for the map
[ INFO] [1589188055.529010694]: Received a 32 X 32 map @ 0.050 m/pix
[ INFO] [1589188055.529054573]: Writing map occupancy data to /tmp/my_map.pgm
[ INFO] [1589188055.529200605]: Writing map occupancy data to /tmp/my_map.yaml
[ INFO] [1589188055.529301794]: Done
```

# Localization

Run following command On Turtlebot:

Run the AMCL:

```
$ rosrun turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml
```

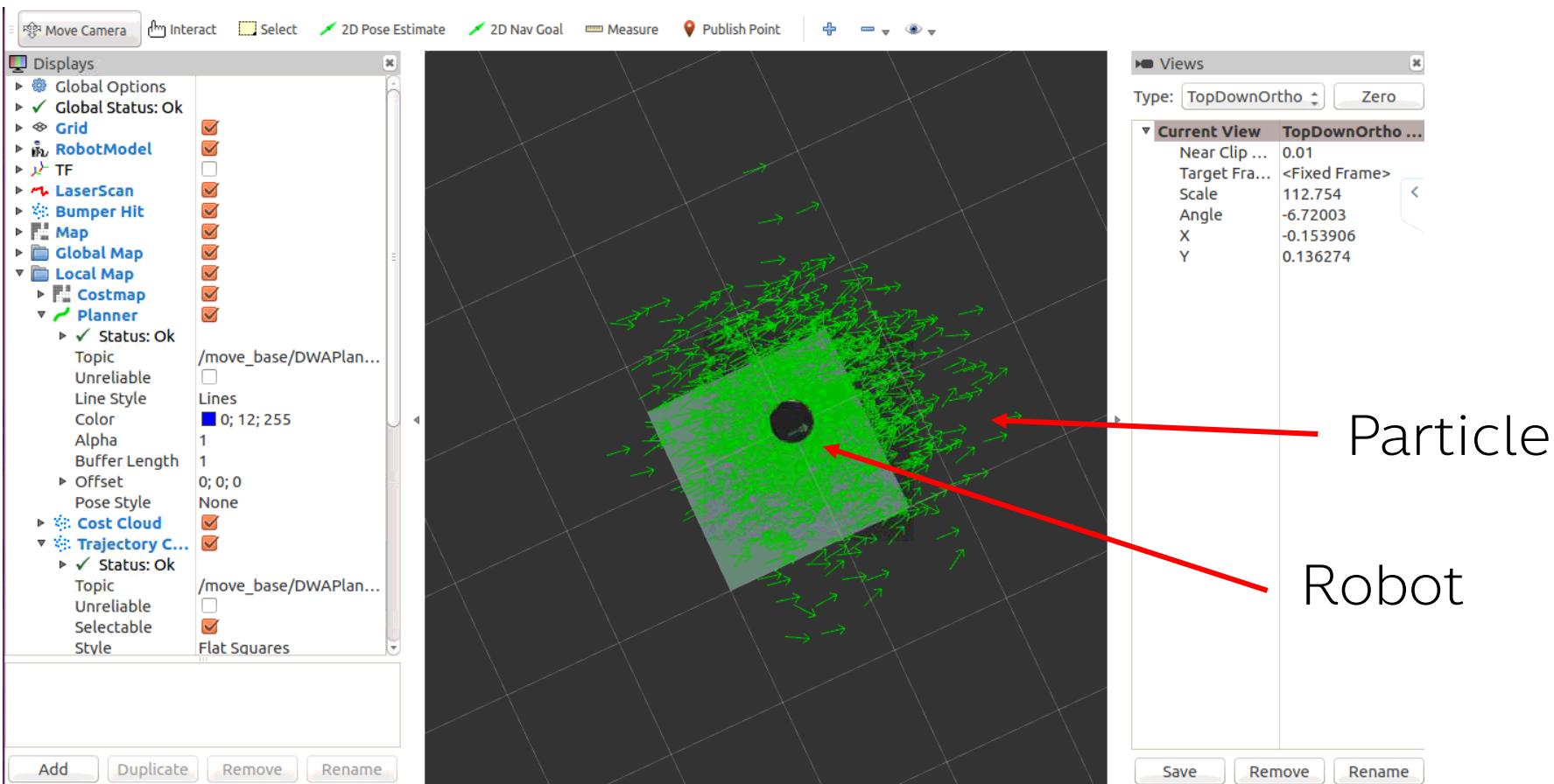
```
[ INFO] [1589188330.345375072]: Using plugin "static_layer"
[ INFO] [1589188330.353272344]: Requesting the map...
[ INFO] [1589188330.558622889]: Resizing costmap to 32 X 32 at 0.050000 m/pix
[ INFO] [1589188330.658583904]: Received a 32 X 32 map at 0.050000 m/pix
[ INFO] [1589188330.669448744]: Using plugin "obstacle_layer"
[ INFO] [1589188330.680303584]: Subscribed to Topics: scan bump
[ INFO] [1589188330.750137639]: Using plugin "inflation_layer"
[ INFO] [1589188330.801454276]: Stopping device RGB and Depth stream flush.
[ INFO] [1589188330.849969357]: Using plugin "obstacle_layer"
[ INFO] [1589188330.854517455]: Subscribed to Topics: scan bump
[ INFO] [1589188330.931099651]: Using plugin "inflation_layer"
[ INFO] [1589188331.005866802]: Created local_planner dwa_local_planner/DWAPlannerROS
[ INFO] [1589188331.011846625]: Sim period is set to 0.20
[ INFO] [1589188332.010352631]: Recovery behavior will clear layer obstacles
[ INFO] [1589188332.016616784]: Recovery behavior will clear layer obstacles
[ INFO] [1589188332.079586595]: odom received!
```

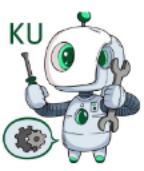
# Localization

Run following command On Workstation:

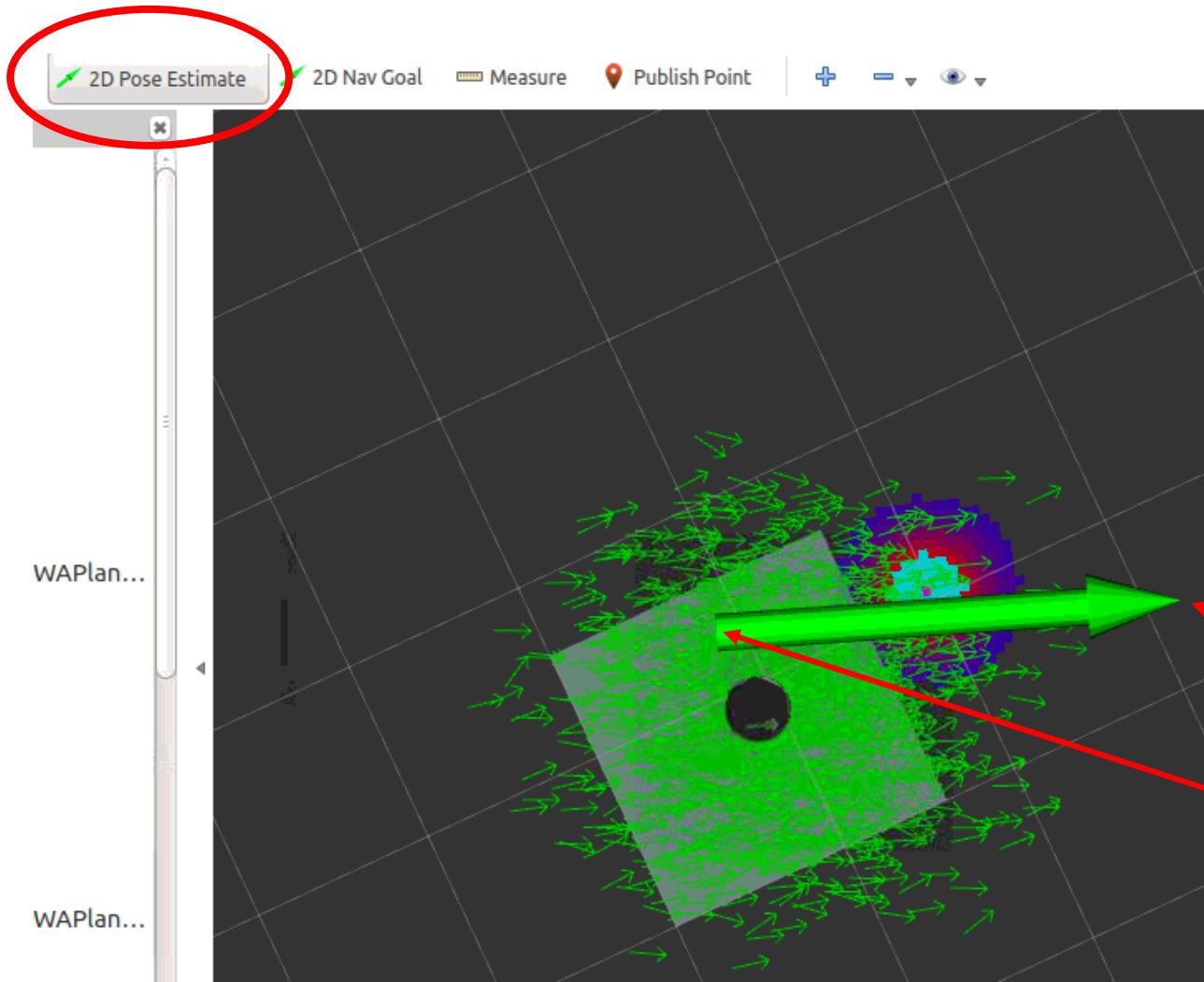
Run the robot visualization to show the map and position of robot:

```
$ rosrun turtlebot_rviz_launchers view_navigation.launch
```





# Localization



1) Initial the position of the robot by click "2D Pose Estimate" button

Then click and hold at guest position.

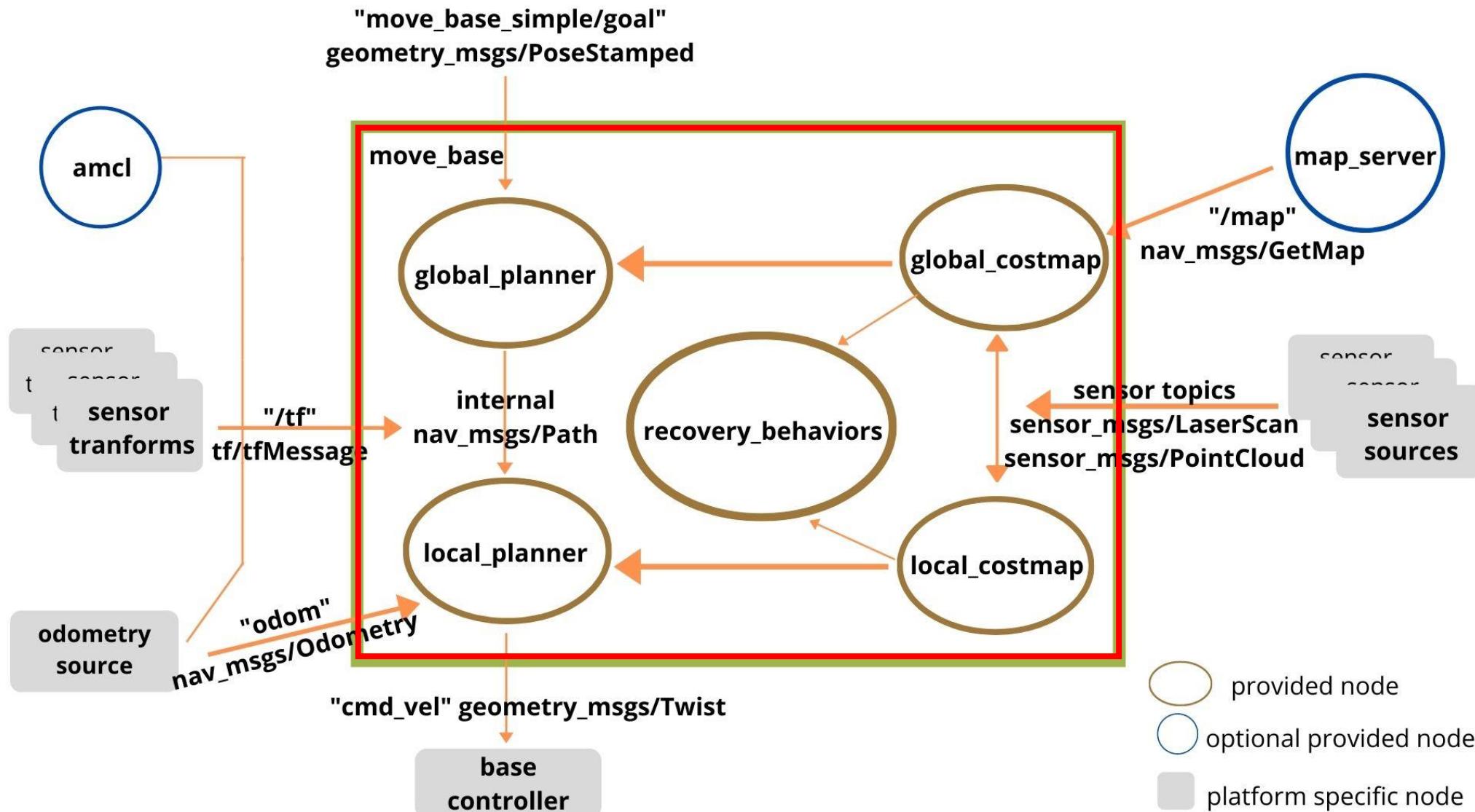
Turn around the position to give heading of robot.

Heading  
Position

# **Navigation and Autonomous Driving**



# Navigation and Autonomous Driving



# Navigation and Autonomous Driving

Goals:

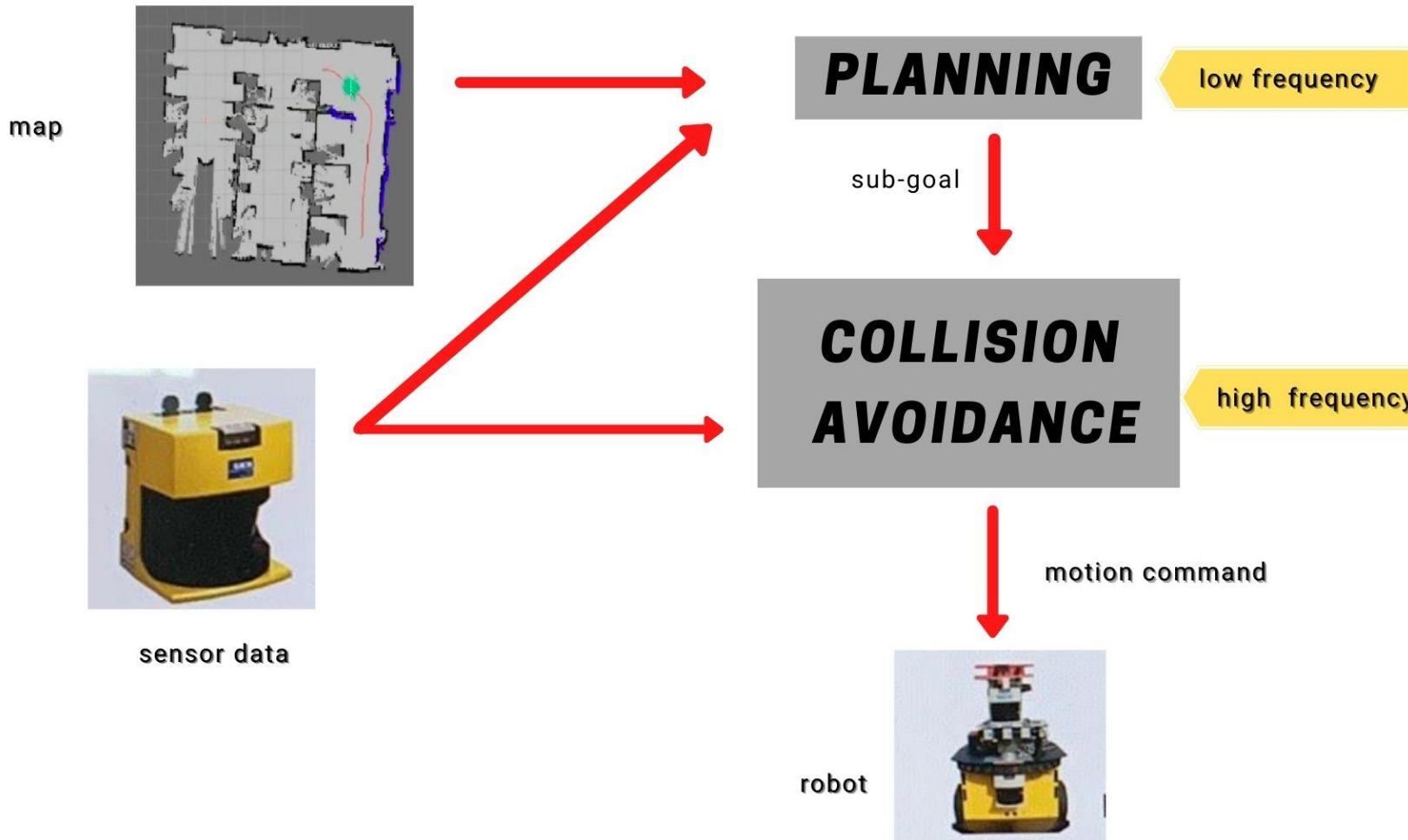
- Collision – free trajectory
- Robot should reach the goal location as quickly

Two Challenges:

- Calculate the optimal path taking potential uncertainties in the actions into account
- Quickly generate actions in the case of unforeseen objects

# Navigation and Autonomous Driving

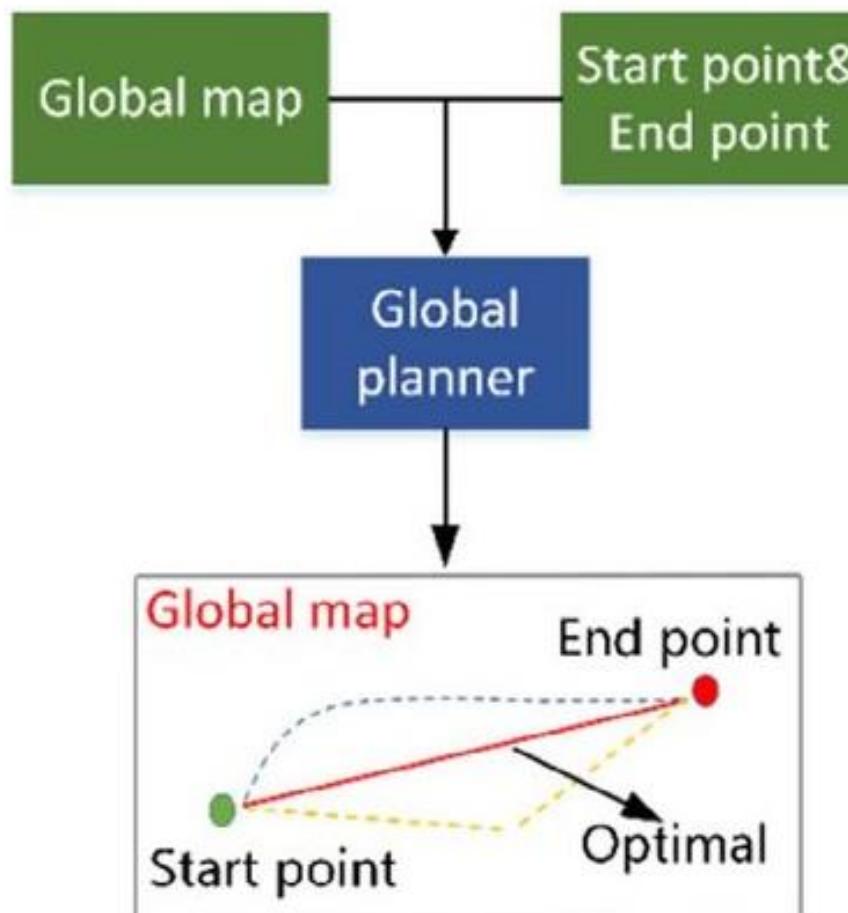
## Classic Two-layered Architecture



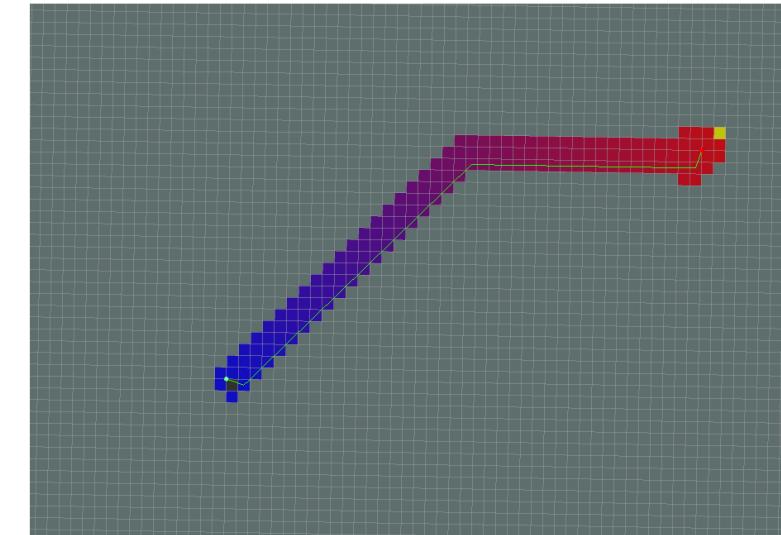


# Navigation and Autonomous Driving

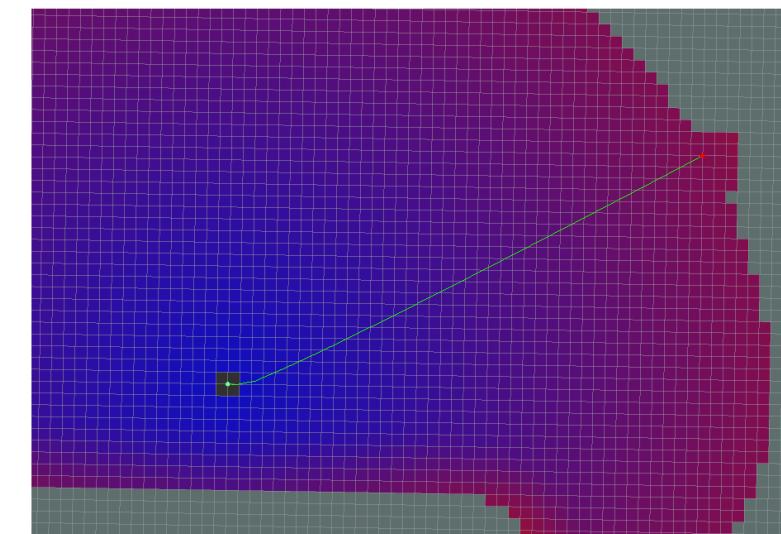
## Path Planning – Global Planner

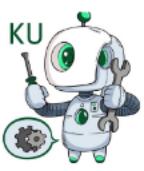


A\*

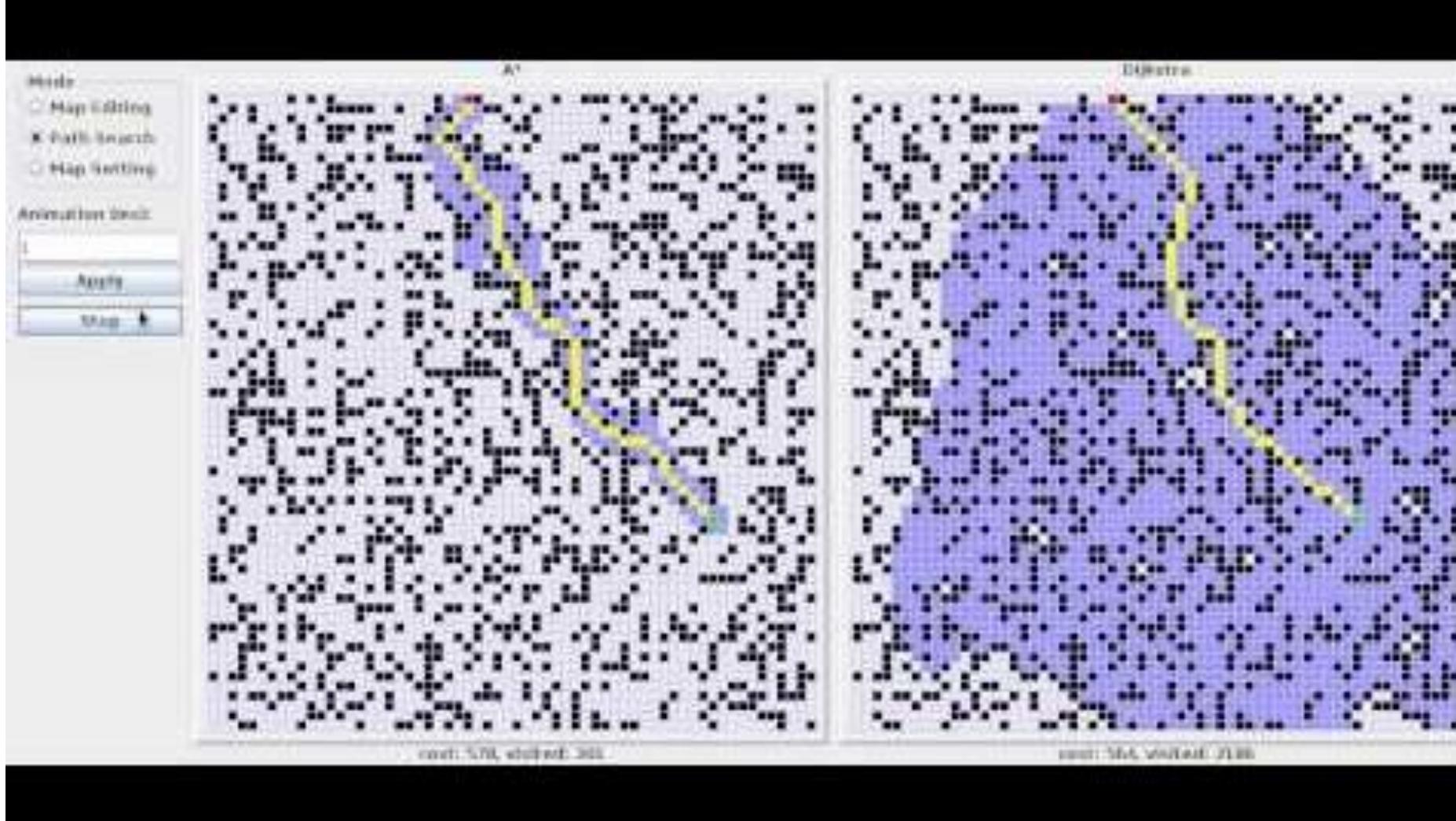


Dijkstra's



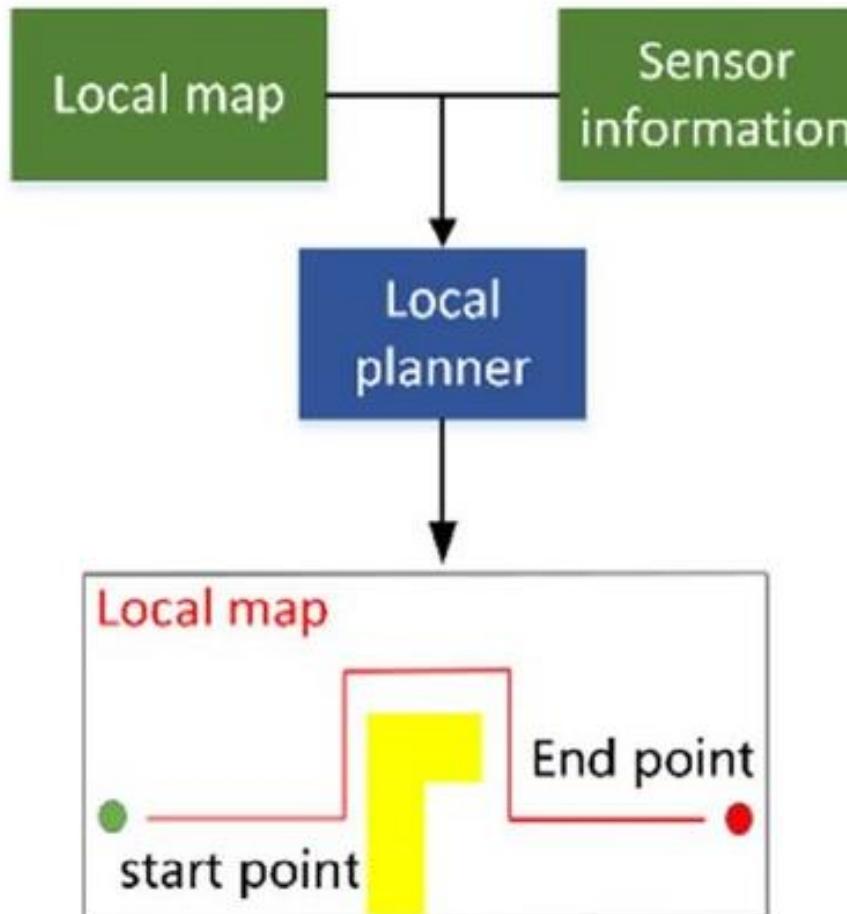


# Navigation and Autonomous Driving

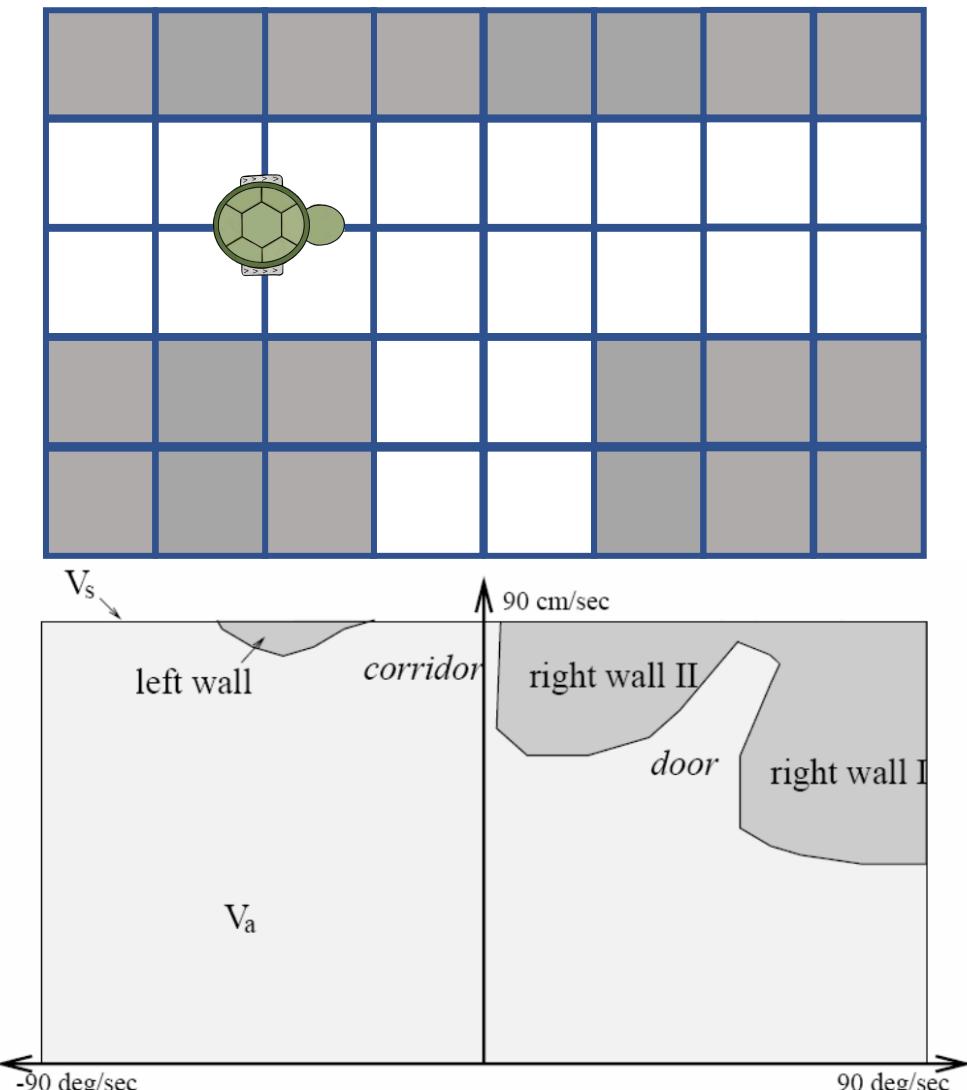


# Navigation and Autonomous Driving

## Path Planning – Local Planner

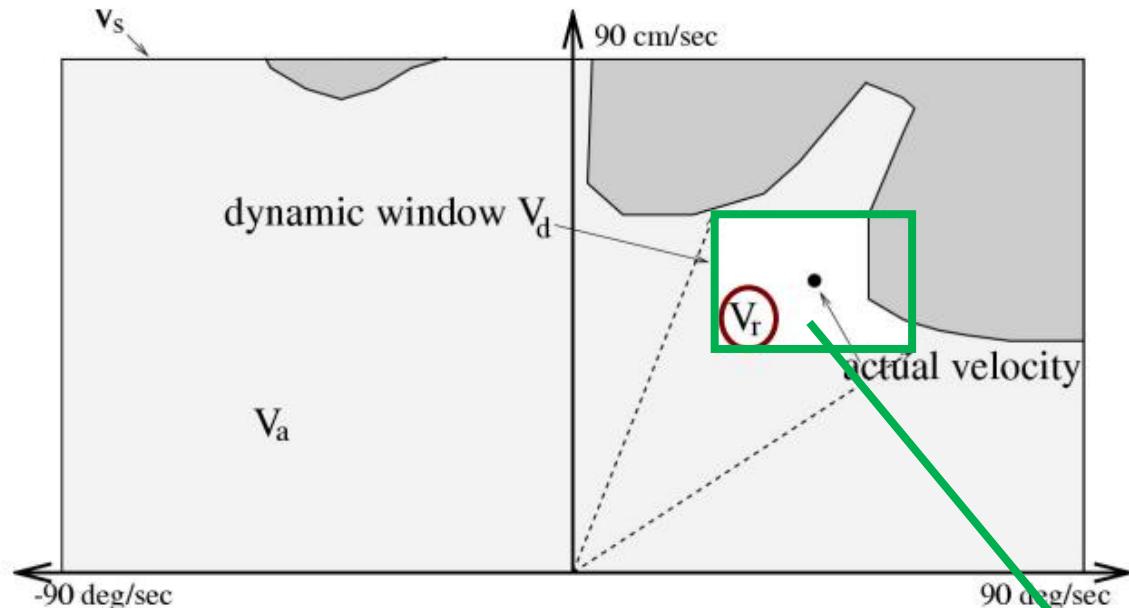


DWA local planer search



# Navigation and Autonomous Driving

DWA local planer search

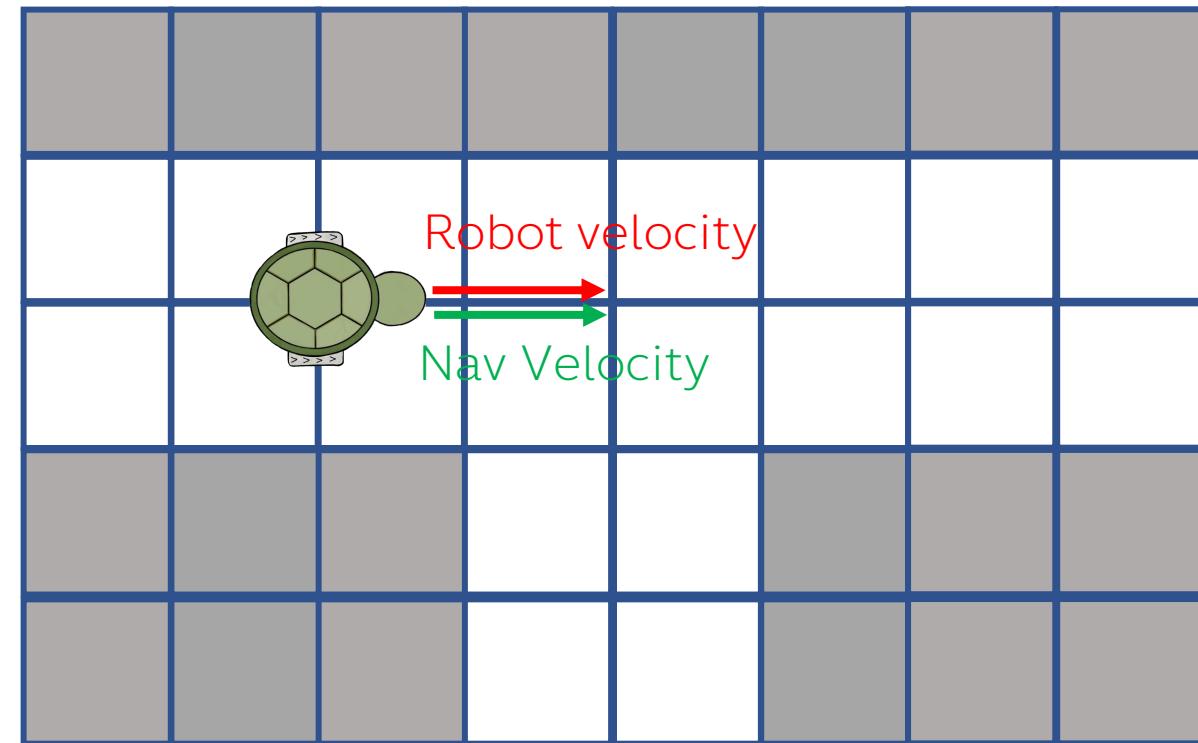


$V_s$  = all possible speeds of the robot.

$V_a$  = obstacle free area.

$V_d$  = speeds reachable within a certain time frame based on possible accelerations.

$$V_r = V_s \cap V_a \cap V_d$$



Maximize  
velocity

Follows Grid based path  
(Global)

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Cost to reach  
the goal

Goal  
nearest

# Navigation and Autonomous Driving

Run following command On Turtlebot:

Run the AMCL:

```
$ roslaunch turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml
```

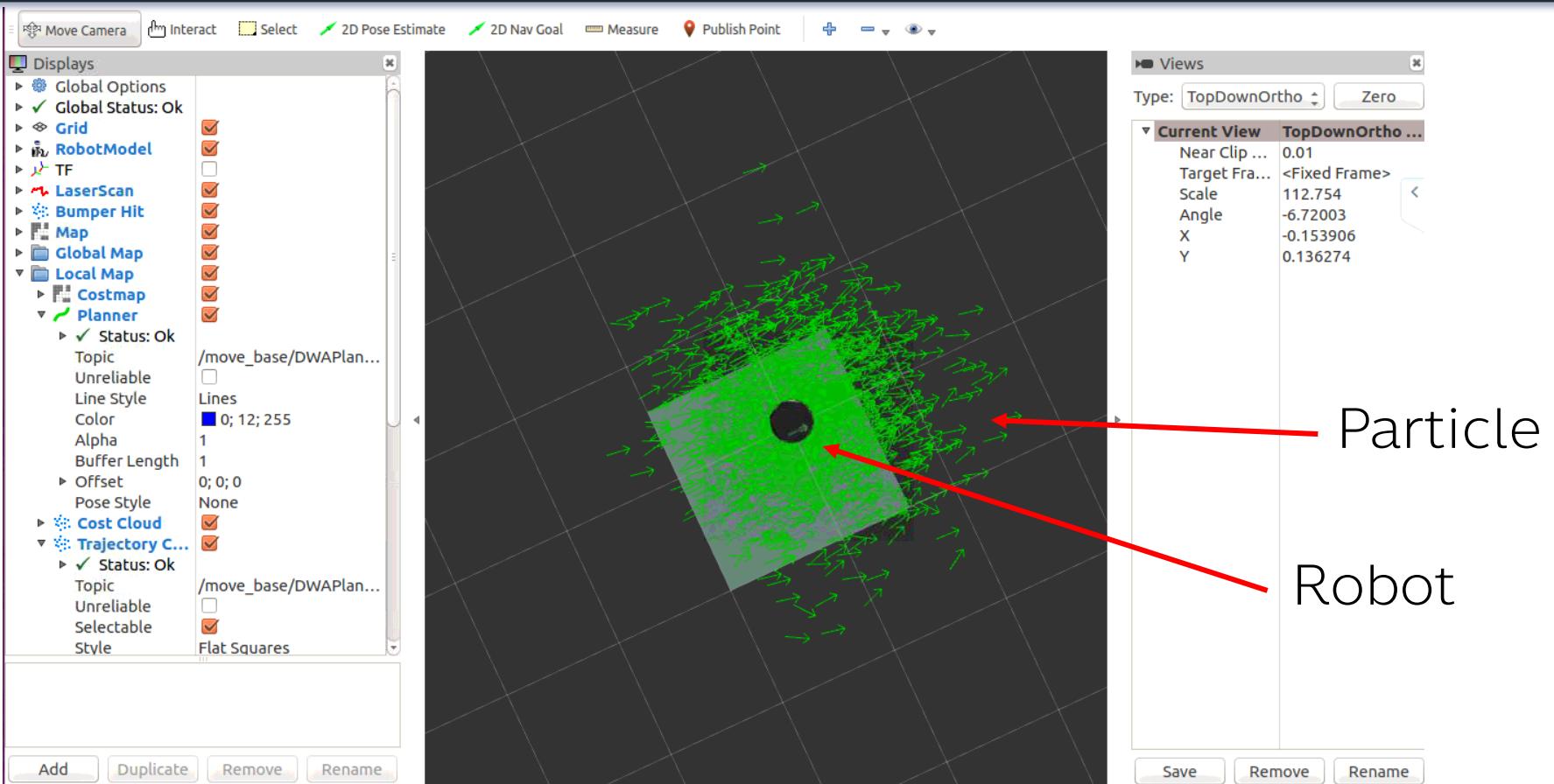
```
[ INFO] [1589188330.345375072]: Using plugin "static_layer"
[ INFO] [1589188330.353272344]: Requesting the map...
[ INFO] [1589188330.558622889]: Resizing costmap to 32 X 32 at 0.050000 m/pix
[ INFO] [1589188330.658583904]: Received a 32 X 32 map at 0.050000 m/pix
[ INFO] [1589188330.669448744]: Using plugin "obstacle_layer"
[ INFO] [1589188330.680303584]: Subscribed to Topics: scan bump
[ INFO] [1589188330.750137639]: Using plugin "inflation_layer"
[ INFO] [1589188330.801454276]: Stopping device RGB and Depth stream flush.
[ INFO] [1589188330.849969357]: Using plugin "obstacle_layer"
[ INFO] [1589188330.854517455]: Subscribed to Topics: scan bump
[ INFO] [1589188330.931099651]: Using plugin "inflation_layer"
[ INFO] [1589188331.005866802]: Created local_planner dwa_local_planner/DWAPlannerROS
[ INFO] [1589188331.011846625]: Sim period is set to 0.20
[ INFO] [1589188332.010352631]: Recovery behavior will clear layer obstacles
[ INFO] [1589188332.016616784]: Recovery behavior will clear layer obstacles
[ INFO] [1589188332.079586595]: odom received!
```

# Navigation and Autonomous Driving

Run following command On Workstation:

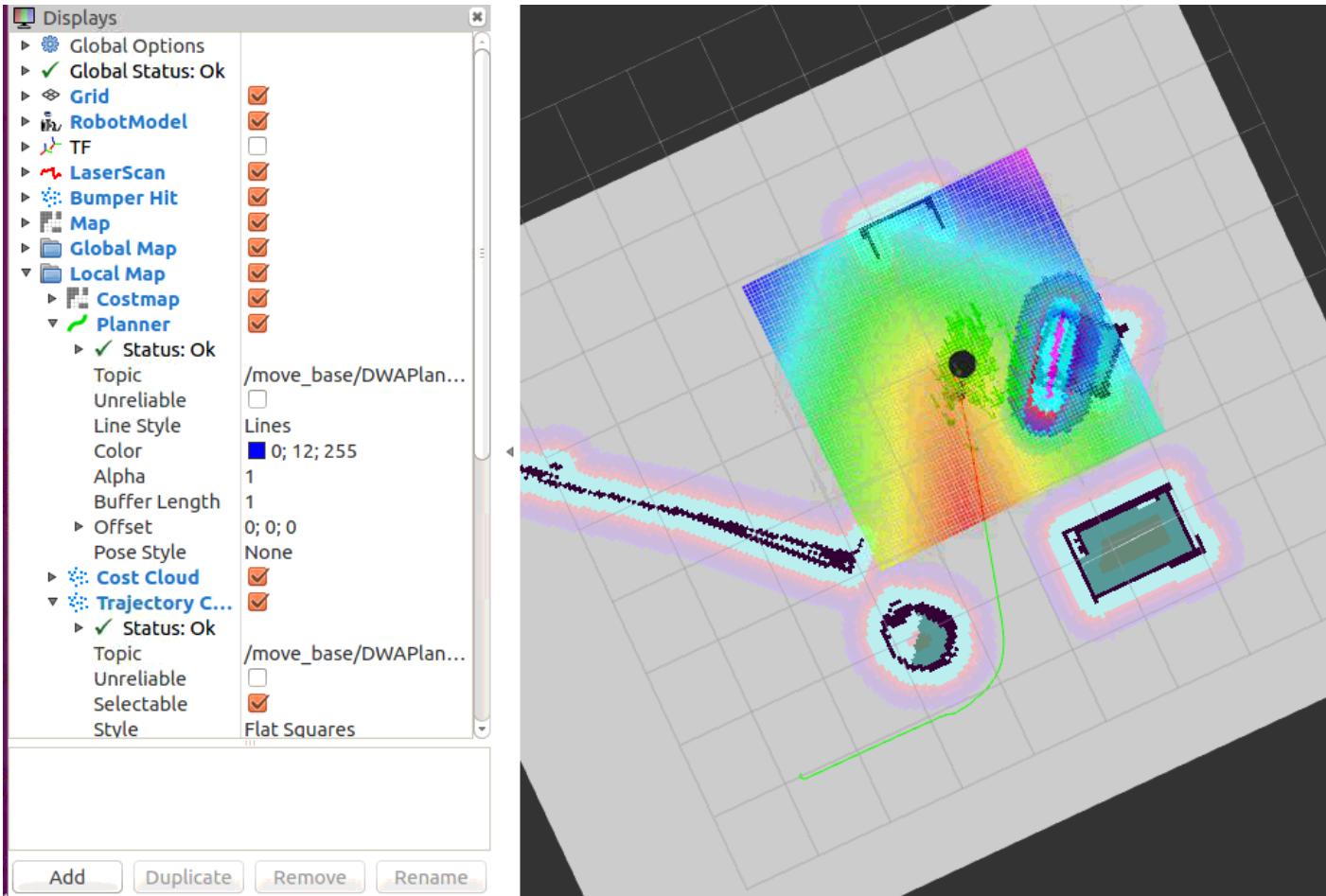
Run the robot visualization to show the map and position of robot:

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```



# Navigation and Autonomous Driving

Send Navigation Goal via RViz



# Navigation and Autonomous Driving

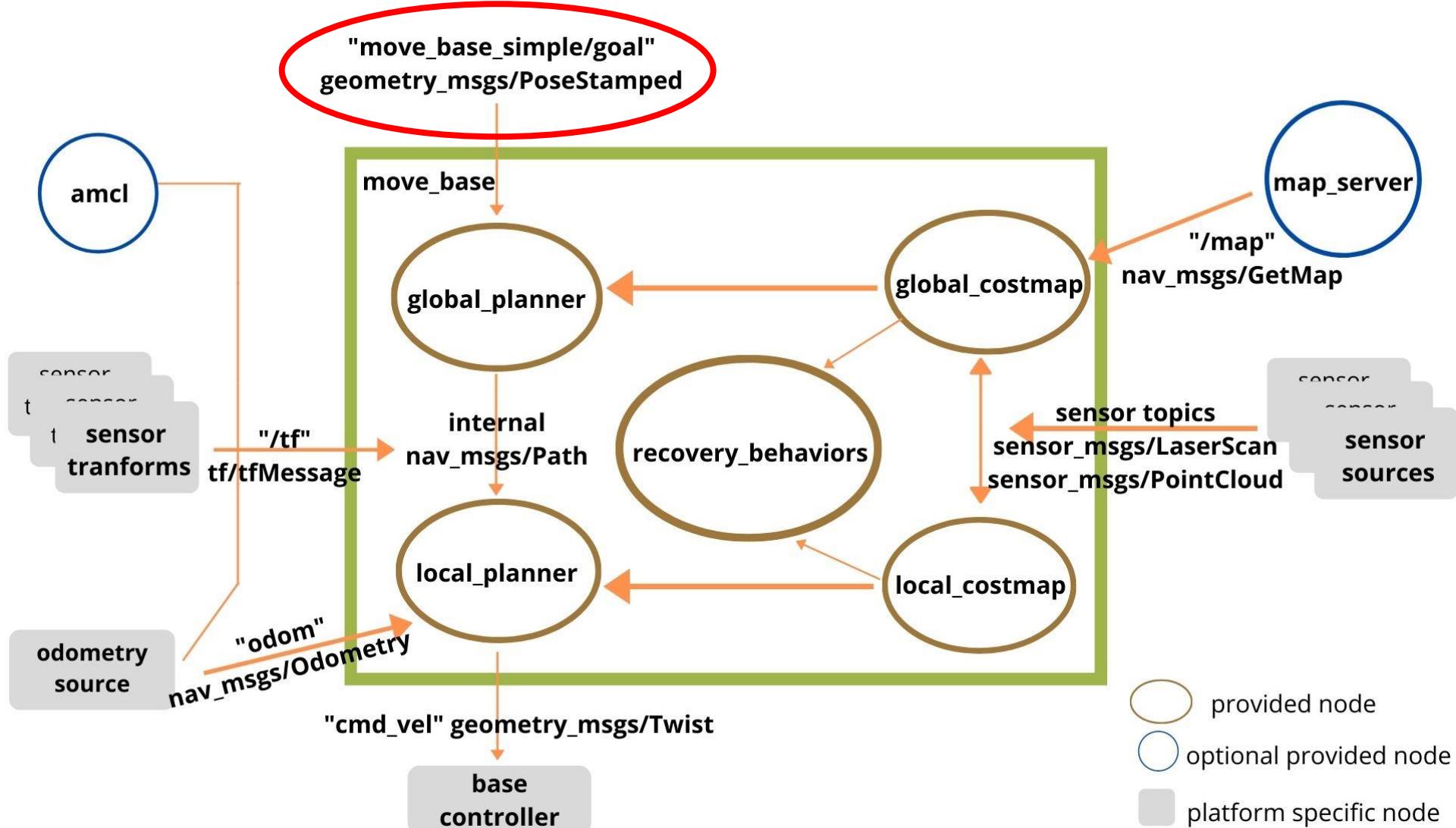
ROS Navigation Stack tuning guide

<http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide>

<http://kaiyuzh.me/documents/navguide.pdf>

# Navigation and Autonomous Driving

Send Navigation Goal using simple program



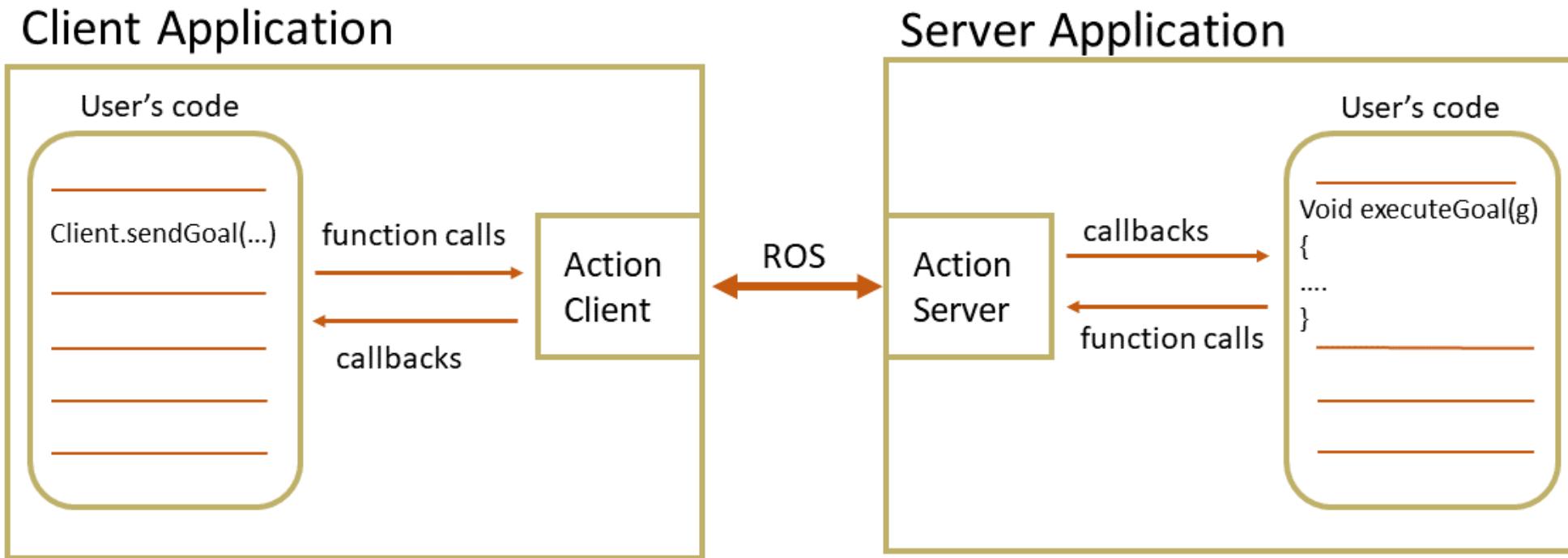
# **Example 1: Move forward with avoidance**





# Navigation and Autonomous Driving

Send Navigation Goal via simple program



# Navigation and Autonomous Driving

## Going Forward and Avoiding Obstacles with Code



```
#!/usr/bin/env python
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Quaternion, Pose, Point

if __name__ == '__main__':
    try:
        GoForwardAvoid()
    except rospy.ROSInterruptException:
        rospy.loginfo("Exception thrown")
```

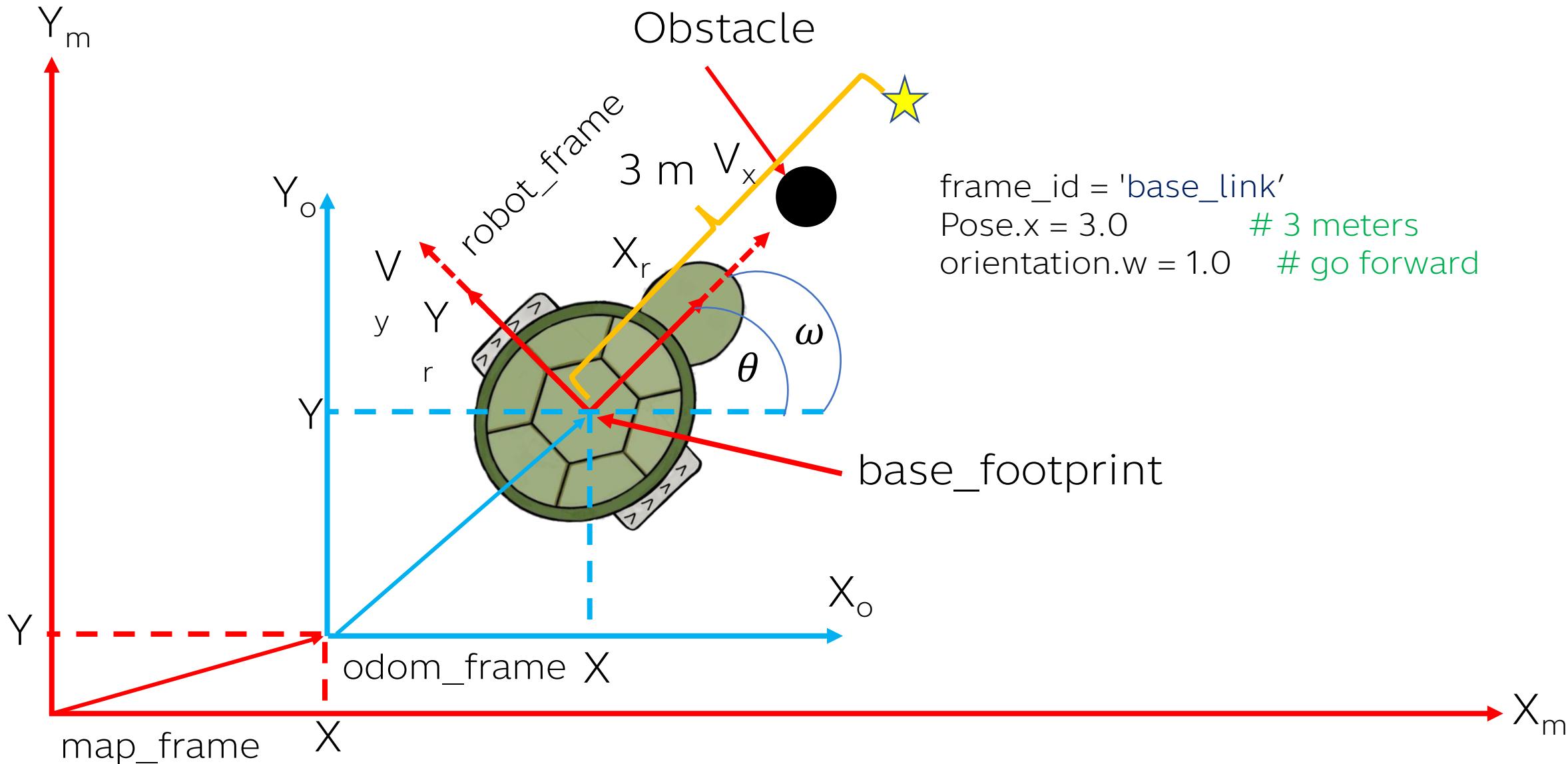
# Navigation and Autonomous Driving

```
class GoForwardAvoid():
    def __init__(self):
        rospy.init_node('nav_test', anonymous=False)
        rospy.on_shutdown(self.shutdown)
        #tell the action client that we want to spin a thread by default
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        rospy.loginfo("wait for the action server to come up")
        #allow up to 5 seconds for the action server to come up
        self.move_base.wait_for_server(rospy.Duration(5))
        #we'll send a goal to the robot to move 3 meters forward
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'base_footprint'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose.position.x = 3.0 #3 meters
        goal.target_pose.pose.orientation.w = 1.0 #go forward
```

} Set Goal



# Navigation and Autonomous Driving



# Navigation and Autonomous Driving

```
#start moving
self.move_base.send_goal(goal)
#allow TurtleBot up to 60 seconds to complete task
success = self.move_base.wait_for_result(rospy.Duration(60))
if not success:
    self.move_base.cancel_goal()
    rospy.loginfo("The base failed to move forward 3 meters for some reason")
else:
    # We made it!
    state = self.move_base.get_state()
    if state == GoalStatus.SUCCEEDED:
        rospy.loginfo("The base moved 3 meters forward")
```

```
def shutdown(self):
    stop_goal = MoveBaseGoal()
    self.move_base.send_goal(stop_goal)
    rospy.loginfo("Stop")
```

# Navigation and Autonomous Driving

Run following command On Turtlebot:

Run the AMCL:

```
$ rosrun turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml
```

Run the robot visualization to show the map and position of robot:

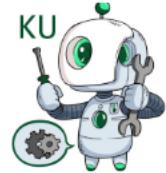
```
$ rosrun turtlebot_rviz_launchers view_navigation.launch
```

Set the robot position

Run the obstacle avoidance node:

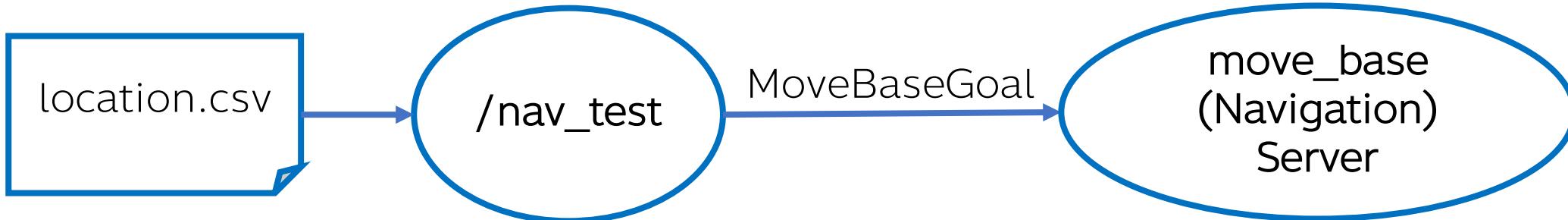
```
$ rosrun <package_name> <program_name>
```

# **Example 2: Go to Specific location**



# Navigation and Autonomous Driving

Going to a Specific Location on Map



```
#!/usr/bin/env python
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from tf.transformations import quaternion_from_euler
if __name__ == '__main__':
    try:
        robot_nav = RobotNavigation()
        robot_nav.go_to_location("location_name")
    except rospy.ROSInterruptException:
        rospy.loginfo("Exception thrown")
```

# Navigation and Autonomous Driving

```
class RobotNavigation():
    def __init__():
        rospy.init_node('nav_test', anonymous=False)
        rospy.on_shutdown(self.shutdown)
        #tell the action client that we want to spin a thread by default
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        rospy.loginfo("wait for the action server to come up")
        #allow up to 5 seconds for the action server to come up
        move_base.wait_for_server(rospy.Duration(5))

    def go_to_location(self, position_name):
        x,y,theta = read_position(position_name)
        x,y,theta = float(x), float(y), float(theta)
        #convert euler to quaternion
        q = quaternion_from_euler(0,0,theta)
```

} Load position

# Navigation and Autonomous Driving

```
goal = MoveBaseGoal()
goal.target_pose.header.frame_id = 'map'
goal.target_pose.header.stamp = rospy.Time.now()
goal.target_pose.pose = Pose(Point(x, y, 0.000), Quaternion(q[0], q[1], q
[2], q[3]))
self.move_base.send_goal(goal)
success = self.move_base.wait_for_result(rospy.Duration(60))
state = self.move_base.get_state()
result = False
if success and state == GoalStatus.SUCCEEDED:
    # We made it!
    result = True
else:
    self.move_base.cancel_goal()
    self.goal_sent = False
return result
```

# Navigation and Autonomous Driving

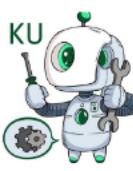
```
def read_csv(self):  
    thisdict = {}  
    with open("path_to_csv_file", "r") as csv_file:  
        csv_reader = csv.reader(csv_file, delimiter = ',')  
        for row in csv_reader:  
            thisdict[row[0]] = [row[1], row[2], row[3]]  
    return thisdict
```

Convert csv to dictionary

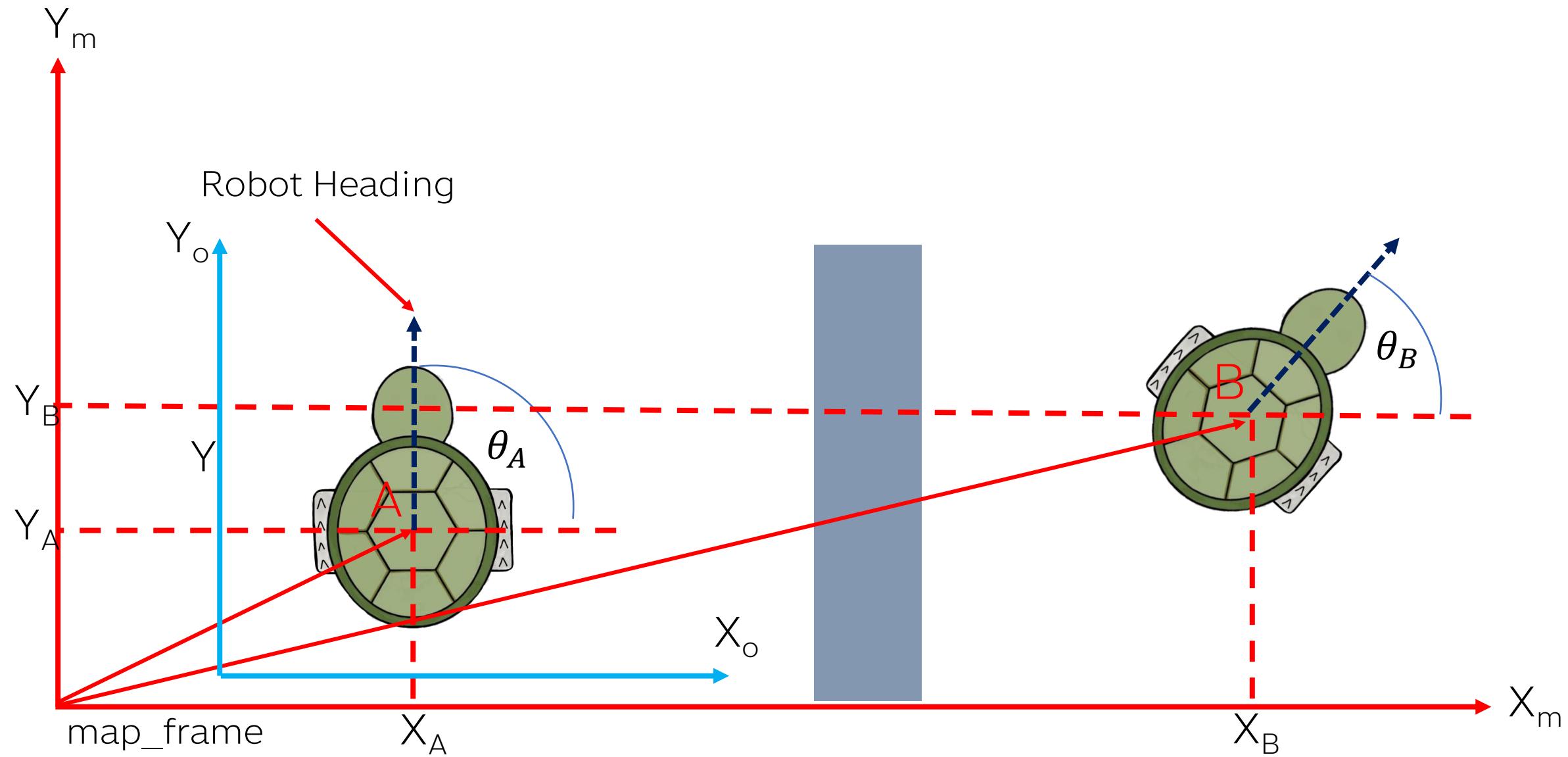
```
def read_position(position_name):  
    dict_position = read_csv()  
    rospy.loginfo(dict_position[position_name])  
    return dict_position[position_name]
```

Read Location from csv  
file

```
def shutdown(self):  
    stop_goal = MoveBaseGoal()  
    self.move_base.send_goal(stop_goal)  
    rospy.loginfo("Stop")
```



# Navigation and Autonomous Driving



# Navigation and Autonomous Driving

Map\_file.csv

```
A, XA, YA, θA
B, XB, YB, θB
```

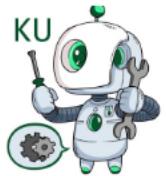
Get Turtlebot Position

```
$ rosrun tf tf_echo /map /base_footprint
```

At time 1263248513.809

- Translation: [2.398, 6.783, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.707, 0.707]  
in RPY [0.000, -0.000, -1.570]

# ROS TF

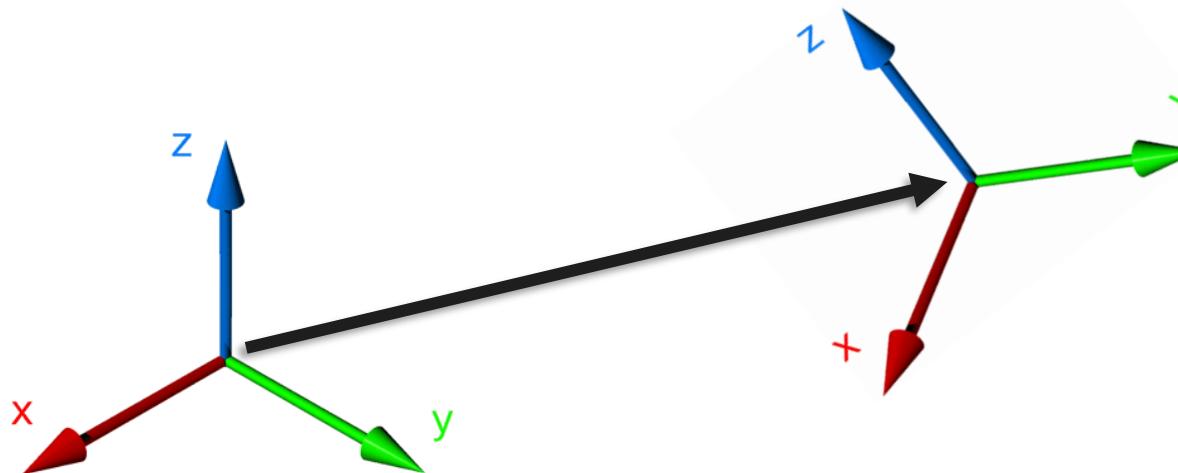


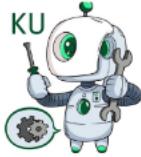


## What is tf

tf is a package that lets the user keep **tracking multiple coordinate frames** overtime.

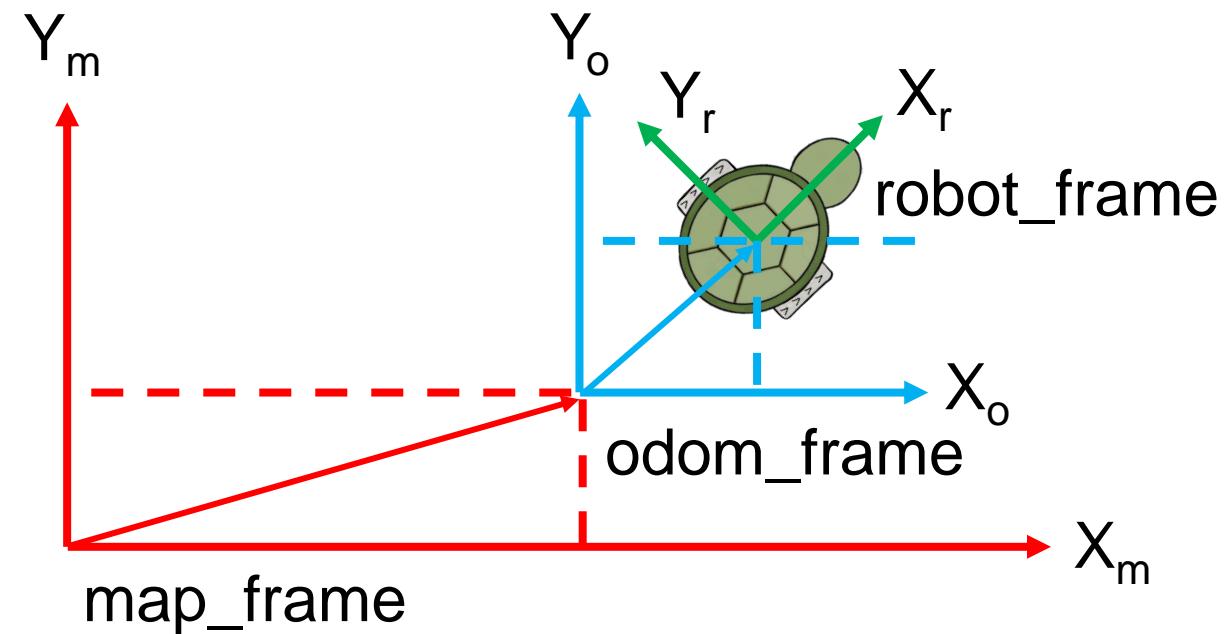
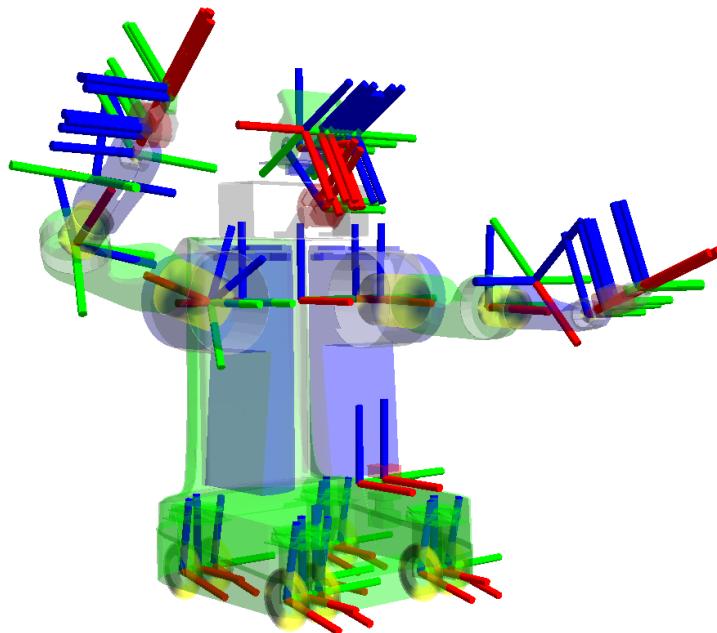
Transform points, or vectors between two coordinates





## Why should I use tf

A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time



## ROS tf Command-line Tools

`tf_monitor <source_frame> <target_frame>`

```
$ rosrun tf tf_monitor /base_footprint /odom
```

`tf_echo <source_frame> <target_frame>`

```
$ rosrun tf tf_echo /map /base_footprint
```

## Writing a tf listener with Python

```
def get_position(self):
    self.listener = tf.TransformListener()
    rate = rospy.Rate(10.0)
    get_position = False
    while not rospy.is_shutdown() and not get_position:
        try:
            (trans,rot) = self.listener.lookupTransform("/map","/base_link", rospy.Time(0))
            if trans != None:
                get_position = True
                print(trans)

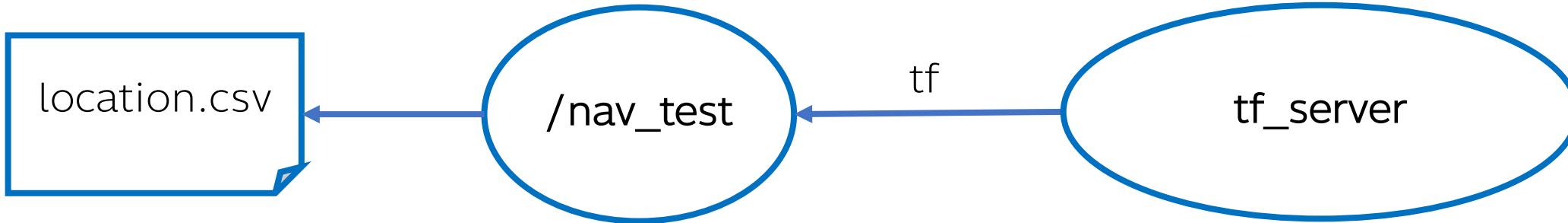
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
            continue
        rate.sleep()
    euler = tf.transformations.euler_from_quaternion(rot)
    return trans[0], trans[1], euler[2]
```

```
def save_position(self, position_name):
    try:
        x,y,theta = self.get_position()
        thisdict = self.read_csv()
        thisdict[position_name] = [x,y,theta]
        thislist = []
        for x in thisdict:
            thislist.append([x,thisdict.get(x)[0],thisdict.get(x)[1],thisdict.get(x)[2]])

        with open("file.csv", "w") as csv_file:
            csv_writer = csv.writer(csv_file,delimiter=',')
            for x in thislist:
                csv_writer.writerow(x)
    return True

except:
    return False
```

## Going to a Specific Location on Map



```
#!/usr/bin/env python
import rospy
import tf
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from tf.transformations import quaternion_from_euler
if __name__ == '__main__':
    try:
        robot_nav = RobotNavigation()
        robot_nav.go_to_location("location_name")
    except rospy.ROSInterruptException:
        rospy.loginfo("Exception thrown")
```



# THANK YOU

