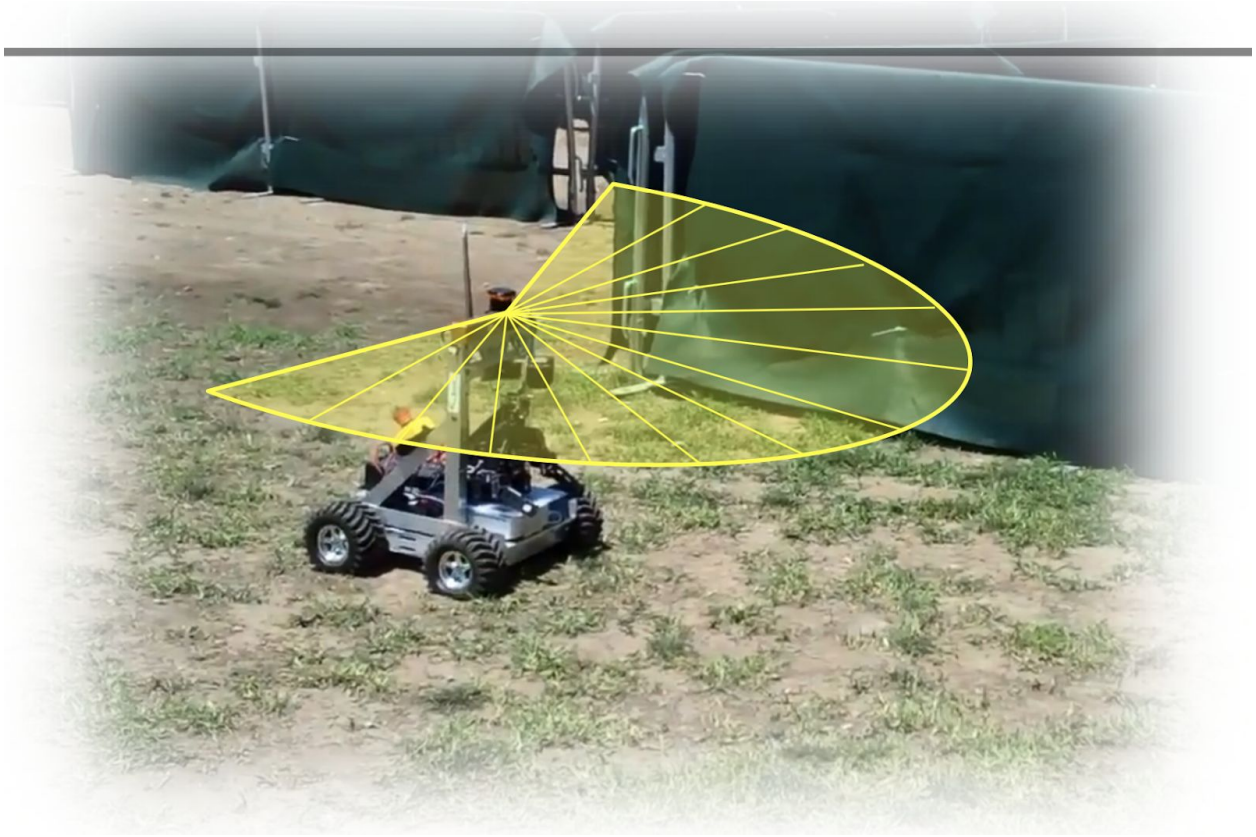
I/ Introduction

The goal of this assignment is to do simultaneous localization and mapping for a robot, given the readings from the robot's wheel encoder, LIDAR sensor, and IMU sensor.



Using dead reckoning and particle filter for localization, as well as occupancy grid map for mapping, I successfully recreate the robot's motion and the map from the given data.

II/ Problem Formulation

The problem is that given data from all sensors, how to localize the robot and map the environment at the same time, given both the sensors' errors and the enviroment's uncertainty (e.g. people walking by).

III/ Technical Approach

1. Localization using Wheel Odometry

I track the robot's position using dead reckoning, which is based on the movement of the robot's wheels. The calculation is as follow:

Tracking Angular Movement

- Encoder ticks (e) are observed at the inner and outer radii

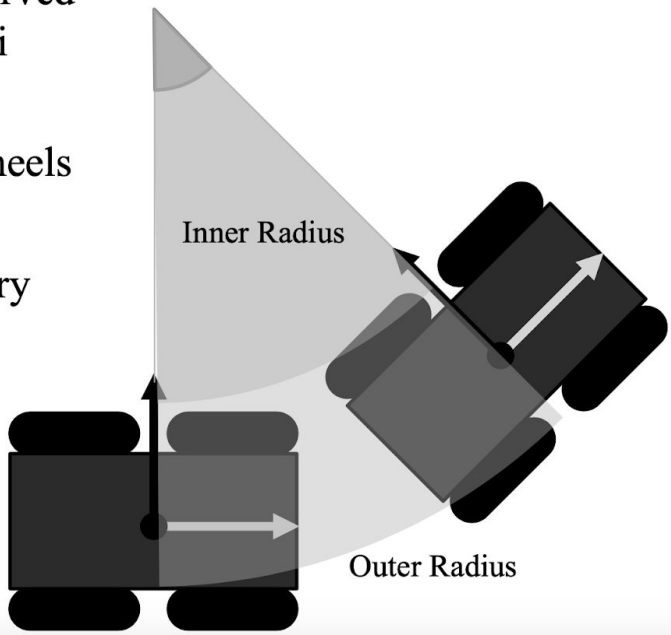
$$e_i = \theta r_i \quad e_o = \theta r_o$$

- Known width between wheels

$$w = r_o - r_i$$

- Calculate angular odometry

$$\theta = \frac{e_o - e_i}{w}$$

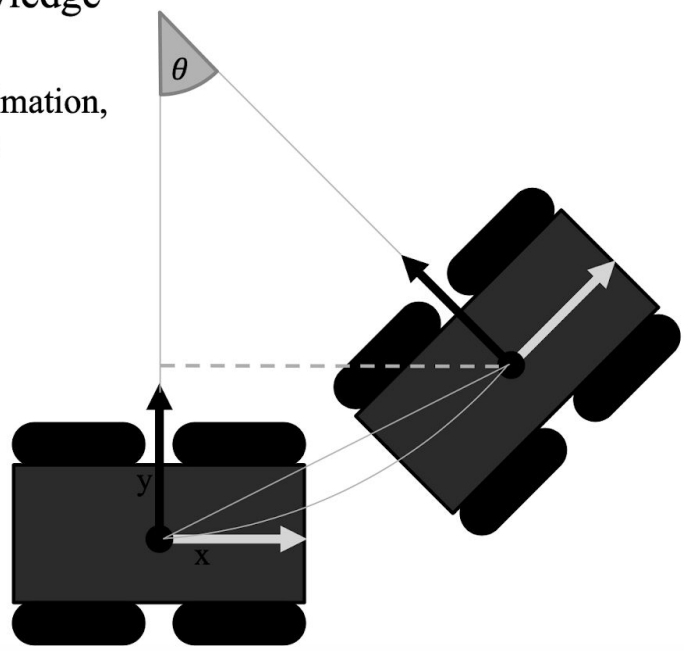


Tracking Translational Motion

- Translation requires knowledge of the angular movement
  - Use circular sector approximation, valid for small movements

$$y = \frac{e_o + e_i}{2} \cos\theta$$

$$x = \frac{e_o + e_i}{2} \sin\theta$$



2. Mapping using Occupancy Grid Map

I build the map using Occupancy Grid Map method. Basically, I divide the environment into a fine-grain grid map. If a lidar ray hits an obstacle in a cell, I will increase the score of occupancy in that cell (+0.9). If the lidar ray travels through a cell without hitting anything, I will decrease the score of occupancy in that cell (-0.7).

At the end of the robot's travel, I will consider a cell to be occupied if its score is higher than certain threshold (> 100), and a cell to be empty if its score is lower than certain threshold (< -100). This method attempts to account for uncertainty in the environment.

Occupancy Grid Mapping

- Example

Constant Measurement Model

$\log odd\_occ := 0.9$

$\log odd\_free := 0.7$

Update

- Case I : cells with z=1

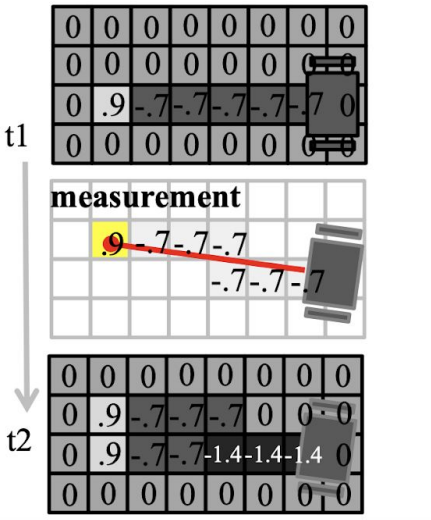
$\log odd \leftarrow 0 + \log odd\_occ$

- Case II : cells with z=0

$\log odd \leftarrow 0 - \log odd\_free$

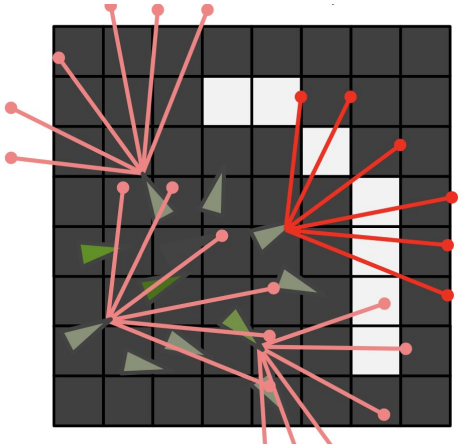
Update Rule:

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



3. SLAM using Particle Filter

To account for errors in both odometry calculation and lidar readings at the same time, I use Particle Filter. The main idea is that at any time, I randomly add odometry noises to create several different robot poses - or different particles. Then I find out which particle is more likely to be the correct robot pose, based on how well its generated map agrees with the previous map.



These are the calculation steps & parameters used:

Number of particles - 20 to 30 particles seem to work well for the Dev & Test data. The higher the number of particles, the more accurate the map seems to become.

Odometry noises - I add 3 random noises to the robot's x-axis, y-axis, and theta angle to create the above particles. Each noise is randomly drawn from a standard normal distribution, with standard deviation ranging from 0.0002 to 0.0003 depending on the number of particles used.

Map correlation & particles' weights update - At each time step, I use LIDAR data to calculate the new map that each particle would generate. Then I calculate the correlation between each particle's new map with the previous map. Then update each particle's weight by multiplying it with the map's correlation. By doing this, the particles that agree with the previous map will get higher weights, while the particles that disagree with the previous map will have smaller weight.

Update odometry & map - After each time step, the particle with highest weight will be considered the correct robot pose. Its odometry and map will be updated to the robot.

Particle resampling - After some time, the number of good particles will decrease, and I need to resample a new set of particles to increase the good particles. Resampling will happen when this condition is met:

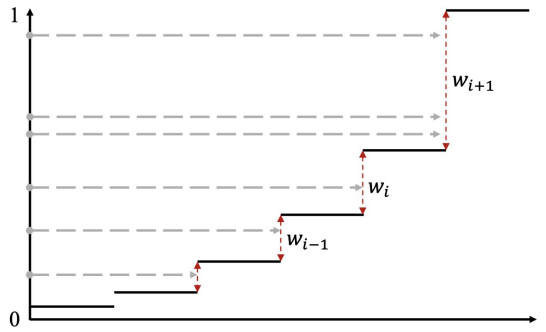
$$n_{effective} = \frac{(\sum_i w_i)^2}{\sum_i w_i^2}$$

$n_{effective}$ < 0.5 * Number of particles

Here, w_i is the weight of particle i.

Particles are then resampled using the method below. New particles will be given equal weights.

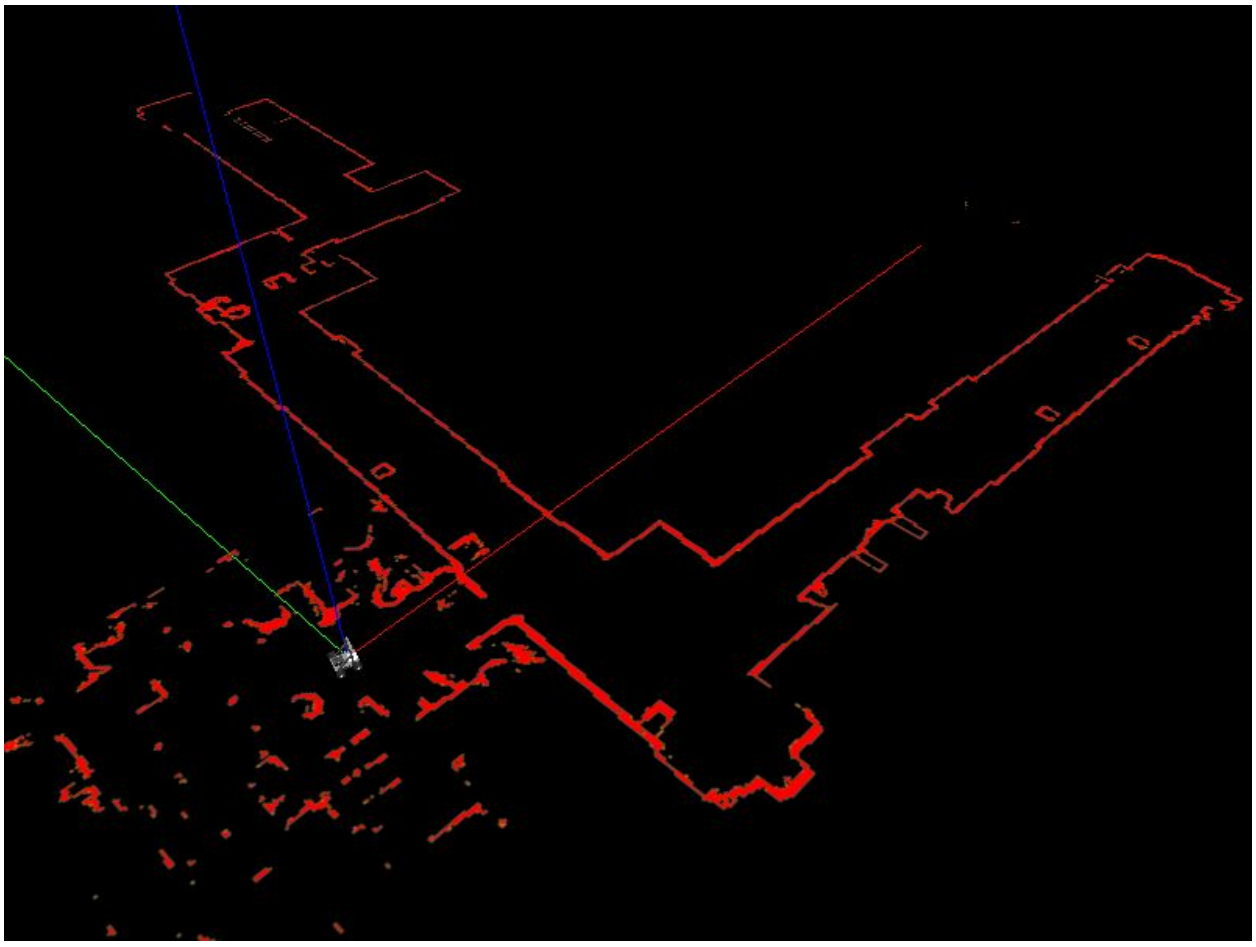
Sample number uniformly between 0 and 1 of the cumulative range, and find which w_i includes that number



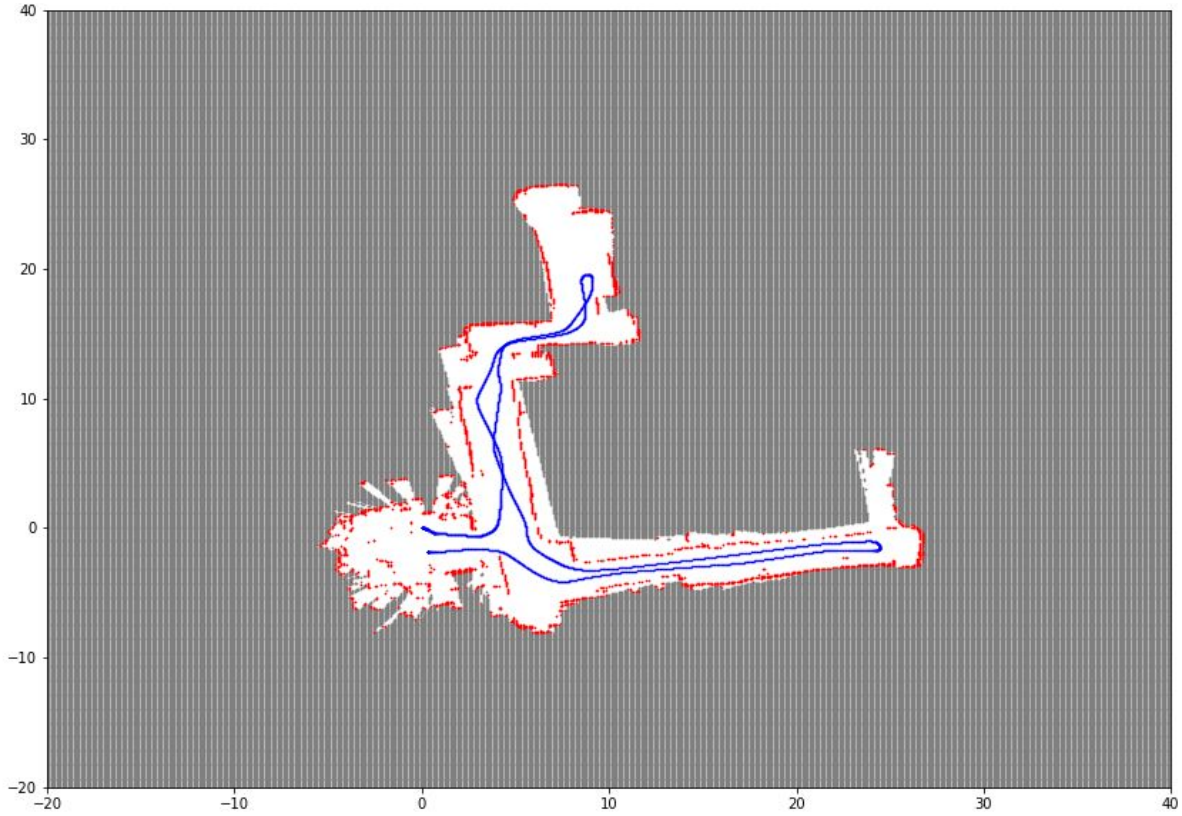
IV/ Results

In the recreated maps in this section, red depicts the wall, blue depicts the robot's trajectory, white depicts the empty space, and gray depicts the unknown region.

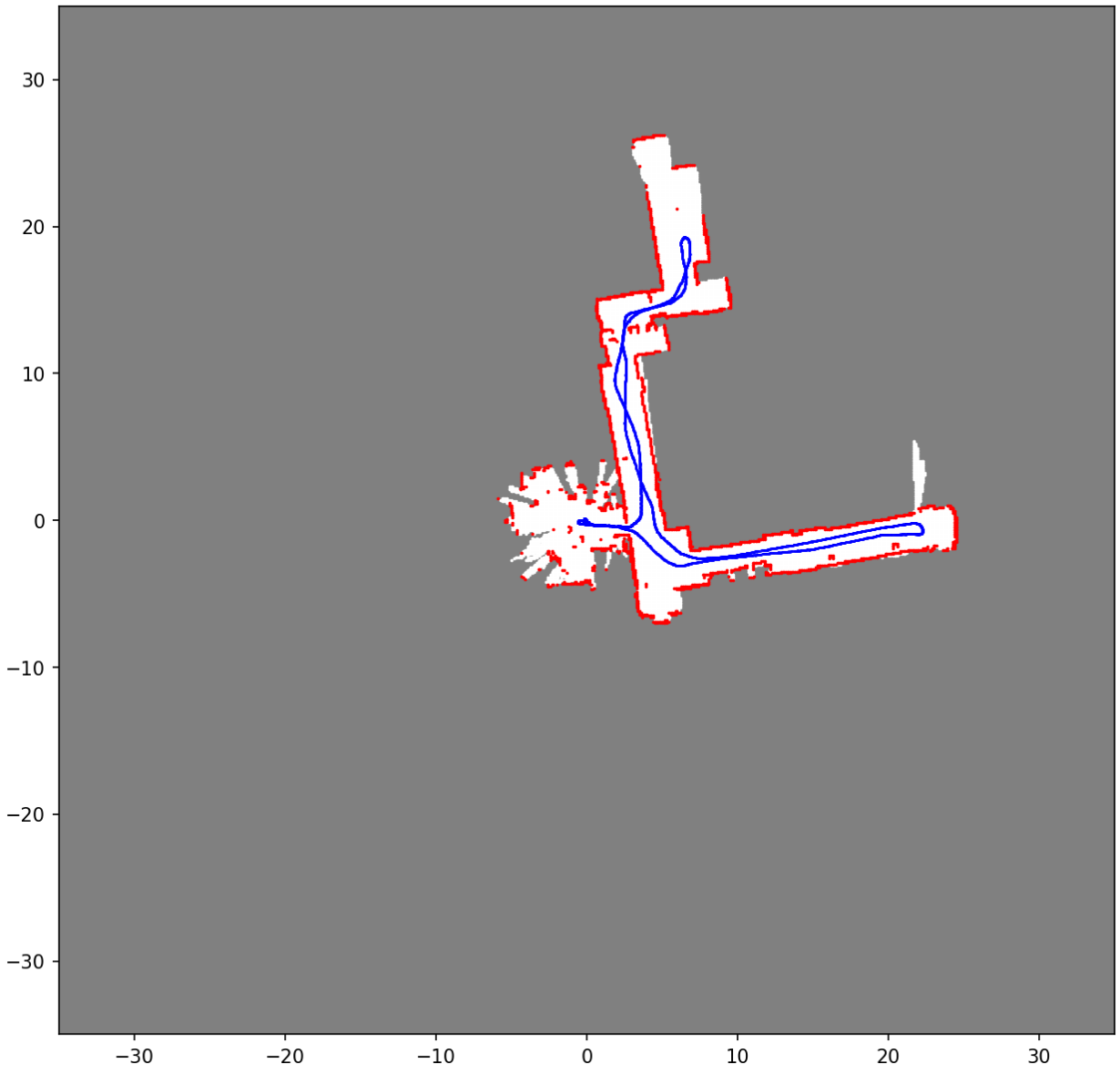
Map 20 - Real map



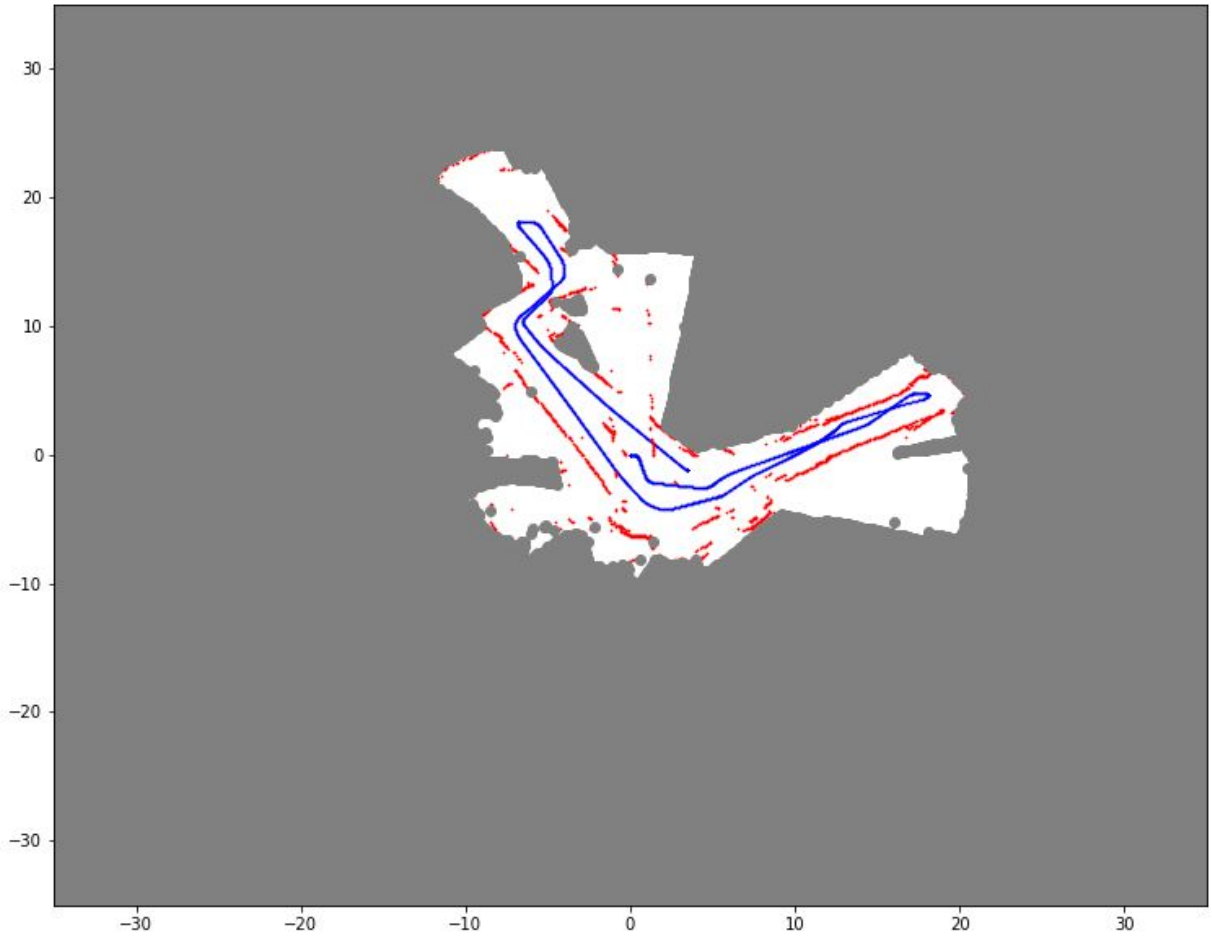
Map 20 - Recreated without Particle Filter



Map 20 - Recreated with Particle Filter

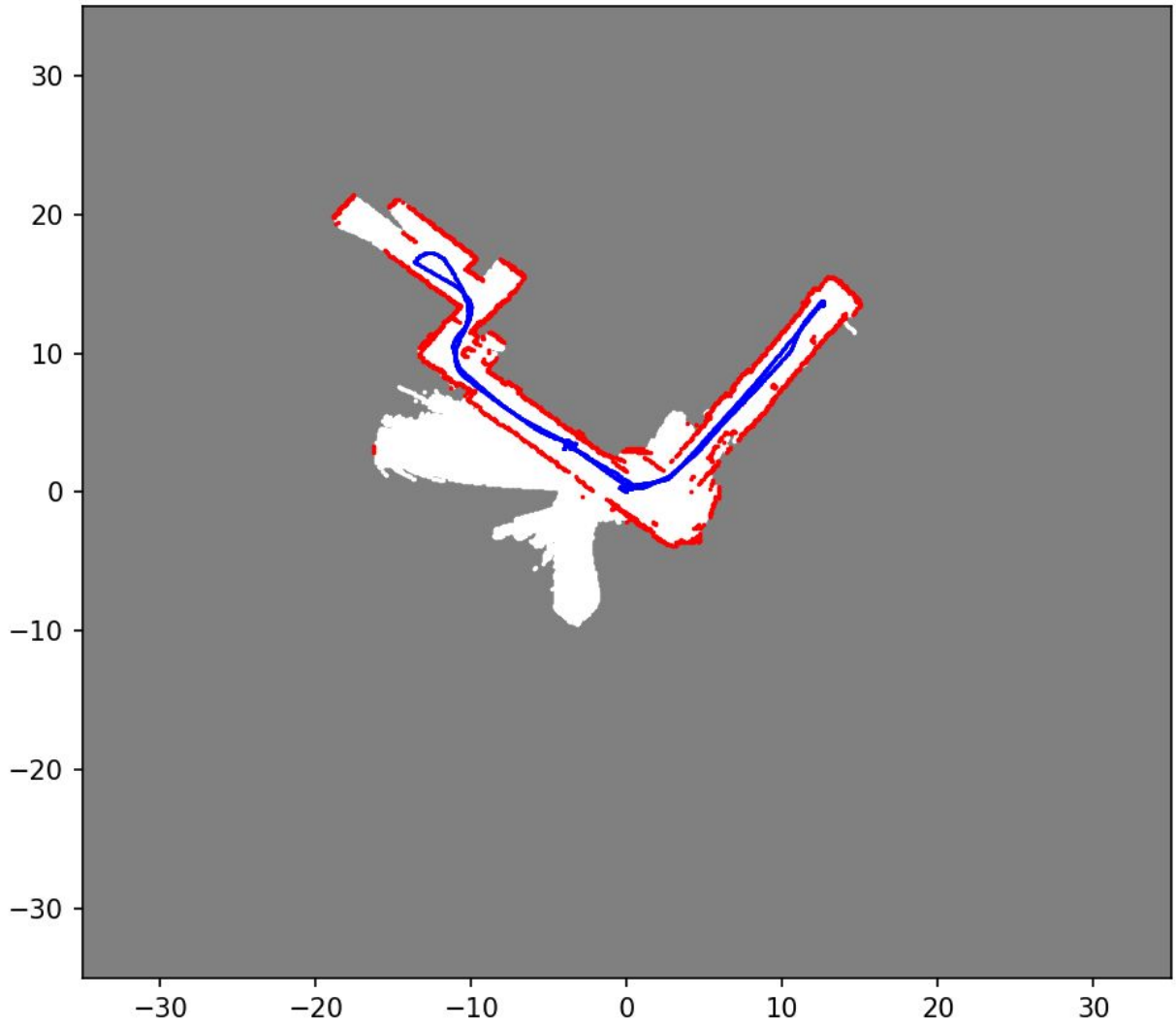


Map 21 - Recreated without Particle Filter



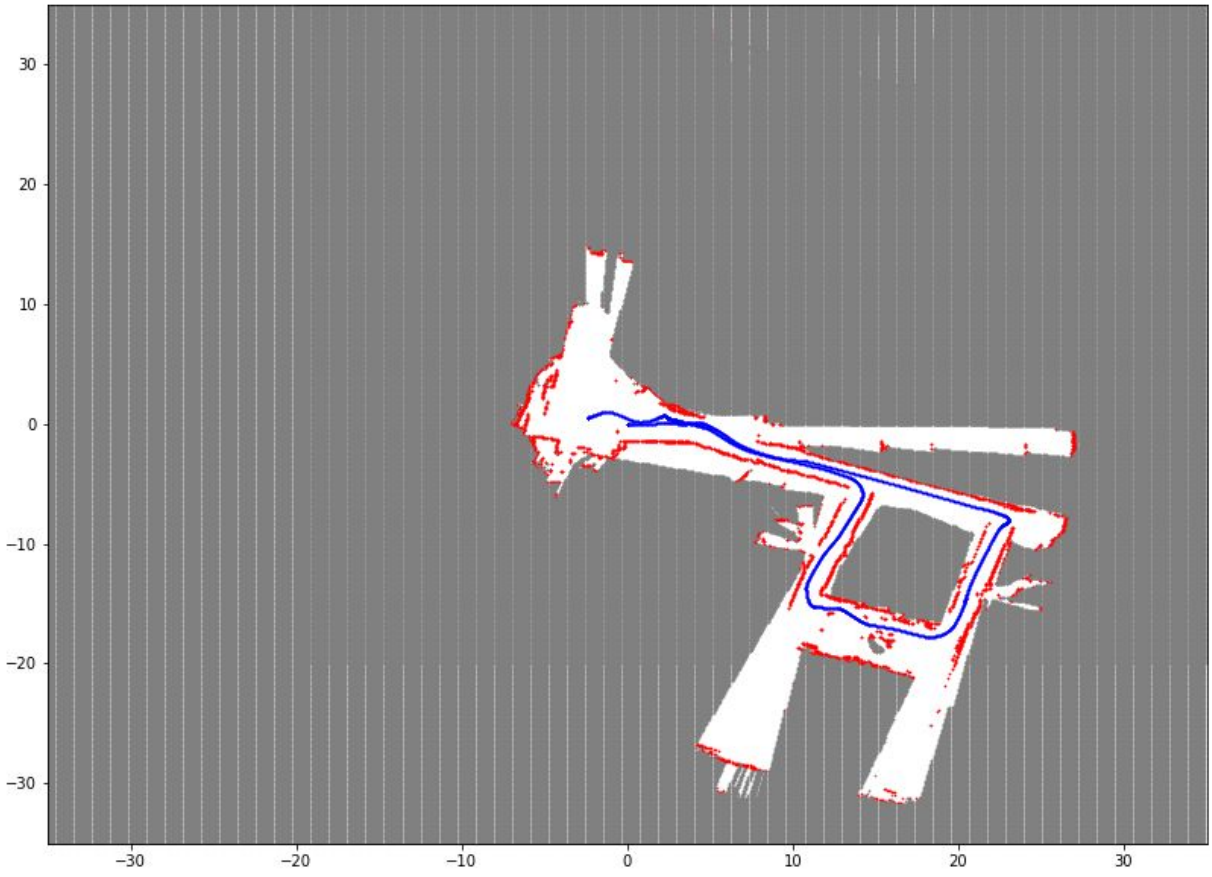
Map 21 - Recreated with Particle Filter

Map 22 - Recreated with Particle Filter

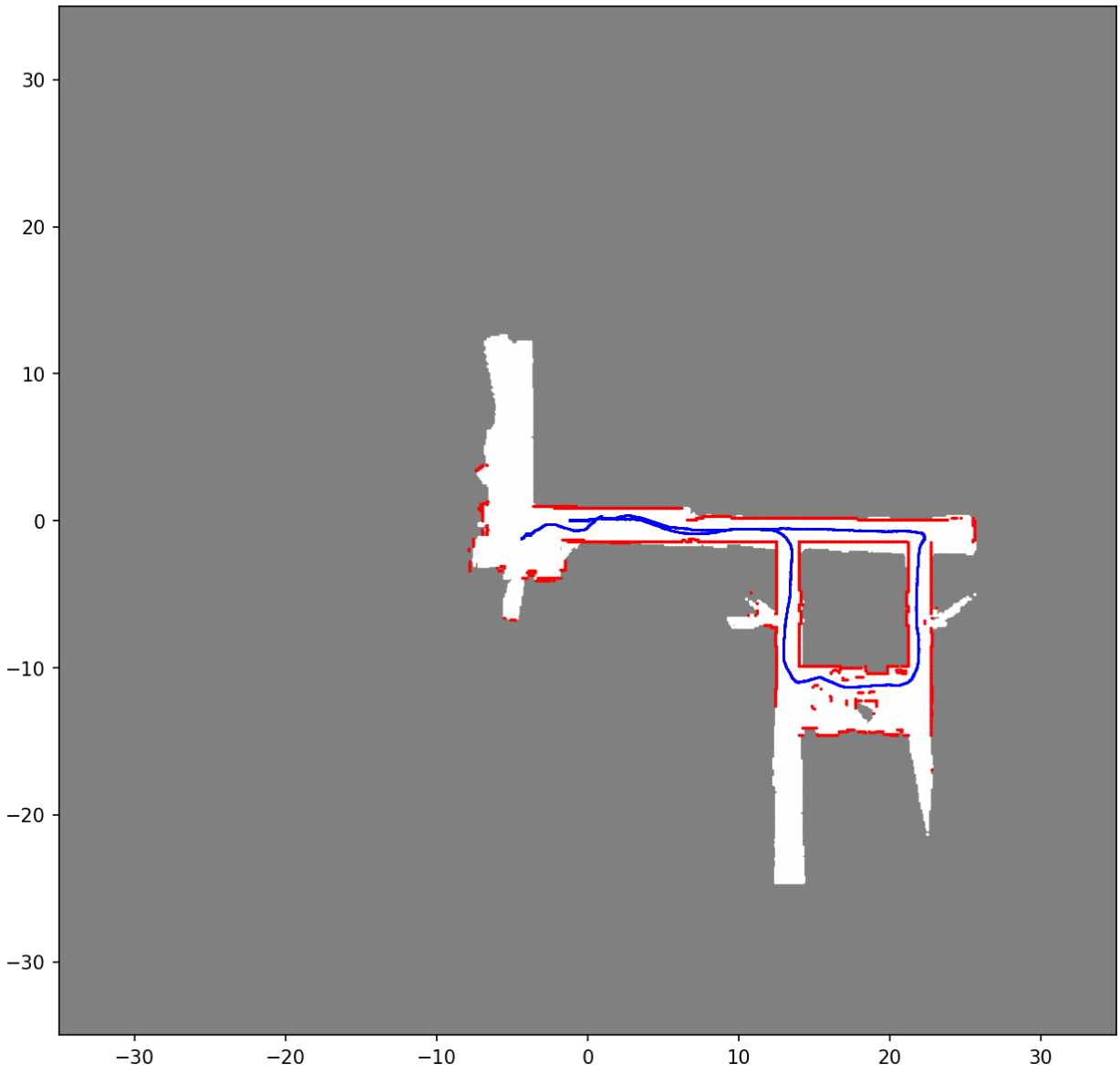


Map 23 - Real map

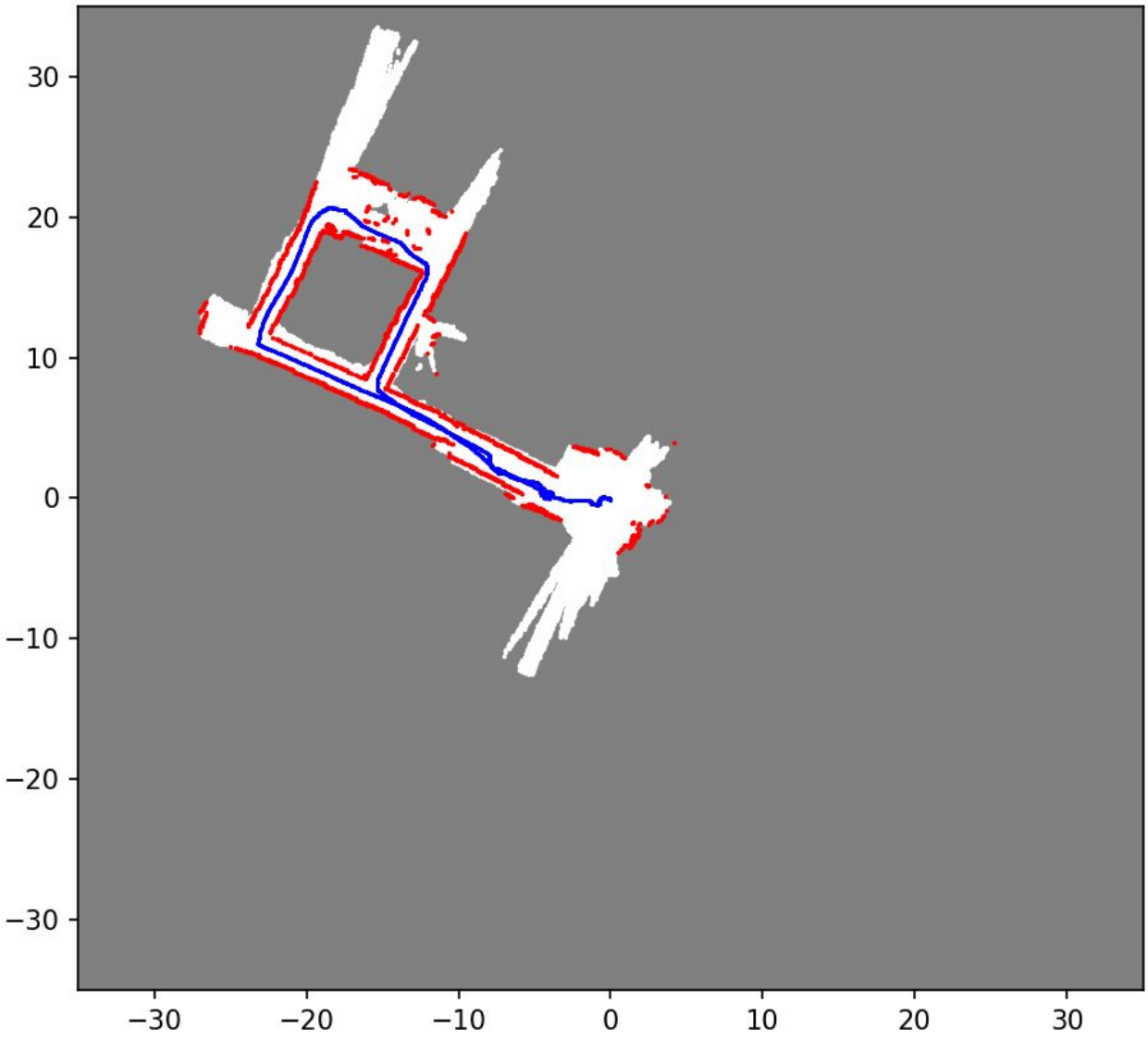
Map 23 - Recreated without Particle Filter



Map 23 - Recreated with Particle Filter



Map 24 - Recreated with Particle Filter



V/ Discussion

Wheel odometry and occupancy grid map do a decent job of tracking the robot's motion and environment. However this method is sensitive to imperfect data, for example small errors when turning the wheel can cause wrong localization, or uneven terrain can cause wrong LIDAR mapping. This explains the noisy shapes in the maps without Particle Filter.

Particle Filter takes this uncertainty into account and improves the result further, resulting in much more accurate maps with cleaner details. In my experiments, 20-30 particles seem to do the job, and the more particles used the more accurate the map details get.

Future improvements include incorporating the IMU data to account for noises from the robot's imbalance due to uneven ground.