FWP – ROS
Dmitrii Dobriborsci

Maze Escape using Turtlebot3

Milestone 3: Final Presentation

Group 4:

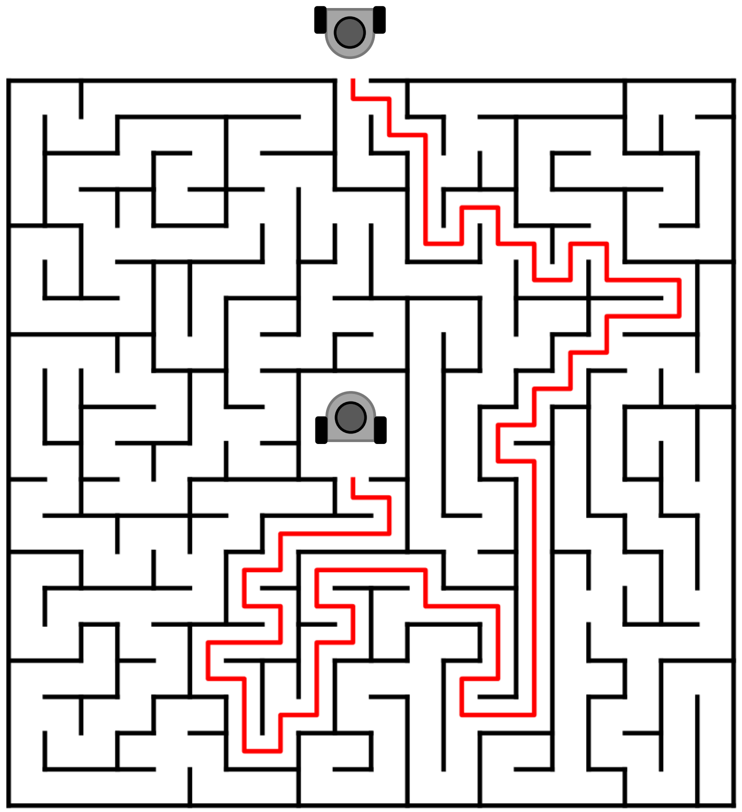
- Angeles Gil
- Contreras Ruben
- Gatti Felix
- Rojas Felipe

CONTENTS

INTRODUCTION

1. **DESCRIPTION**
   - The project aim is to develop and implement a search or navigation algorithm that lets the Turtlebot3 solve different kind of maze configurations effectively in a virtual environment.

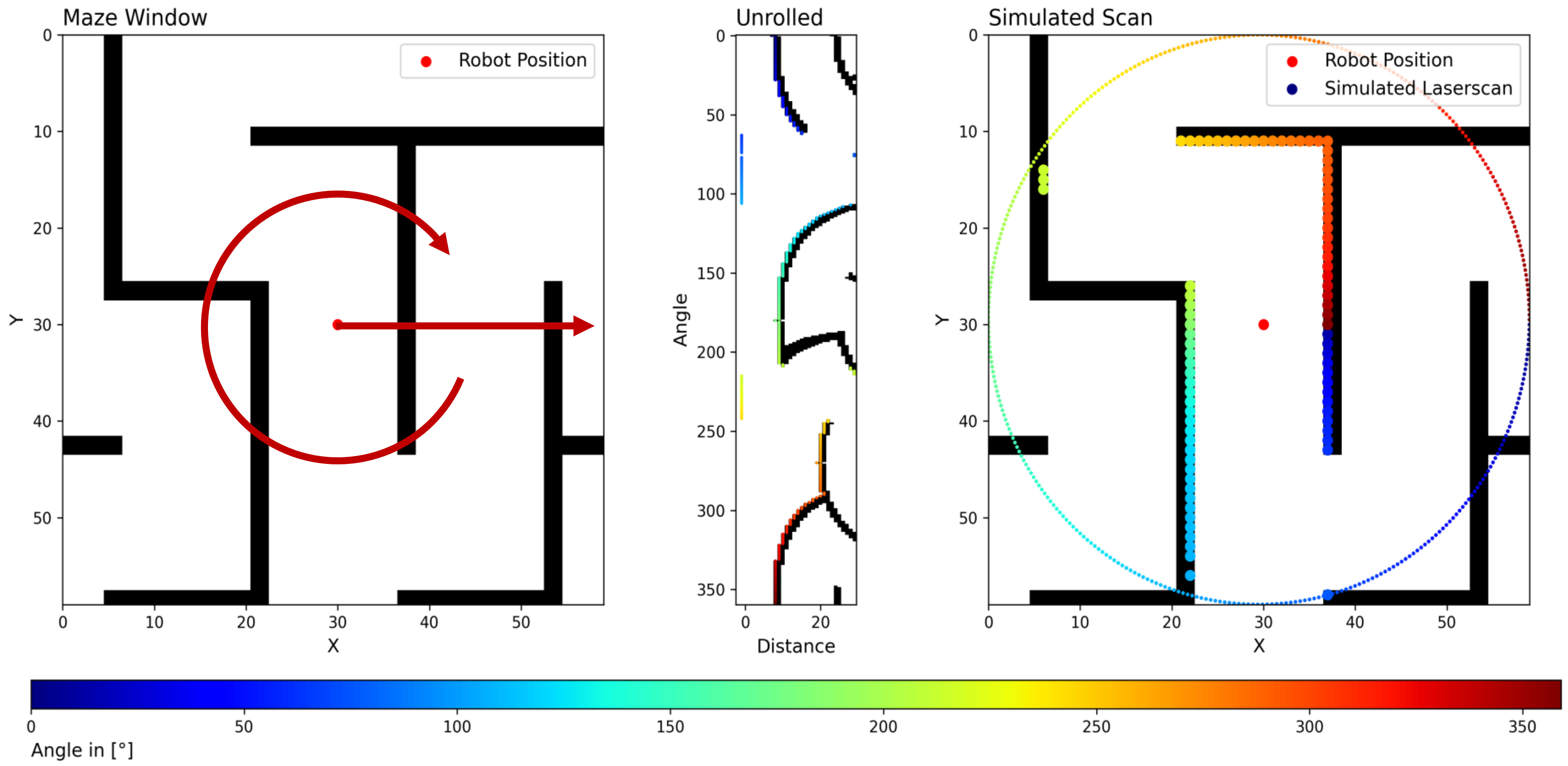
2. **DEFINITION INITIAL**
   - Scenario A)
     - “Random Exploration”
     - Robot has no information of the surrounding maze
     - Initial escape explores maze according to used algorithm
     - Second escape uses learned path to escape
     - Implementation 1: Mathematical Simulation using Python
     - Implementation 2: Gazebo Simulation



FINAL – SIMULATION – LASER SCAN

1. SIMULATION LASERSCAN

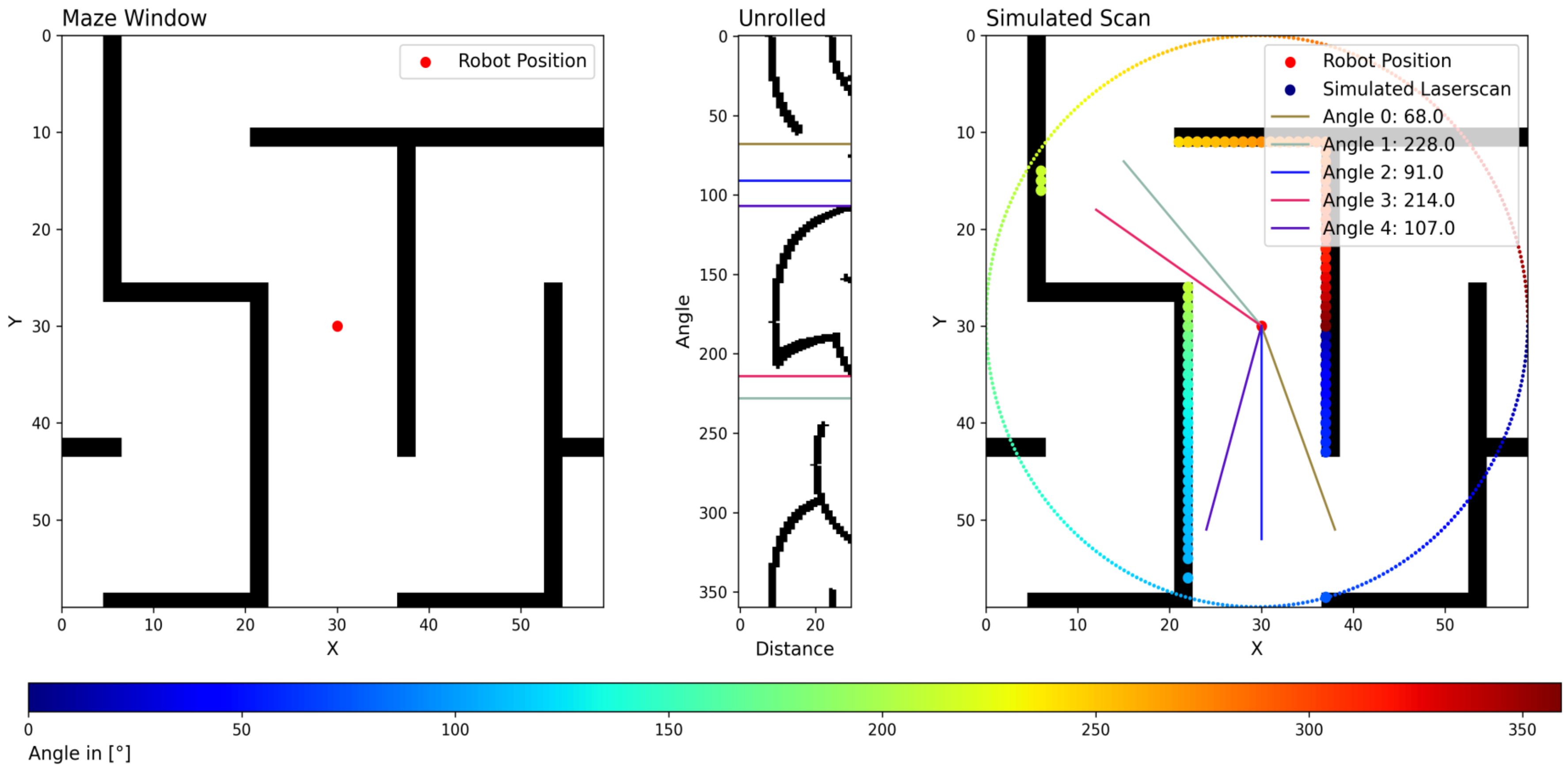
- **Scan Simulation**
- Get area with defined size (Laserscan range)
- Unroll area with resolution (360° resolution)
- Calculate first flank in x-direction
- **Map building**
- Build map on scan values in combination with robot position



FINAL – SIMULATION – LASER SCAN

1. LOCAL PATH PLANNING

- **Calc. possible angles**
- Global maxima (Numpy)
- Local maxima (Scipy)
- OoR (Out of Range)
  - Take middle value of oor angles



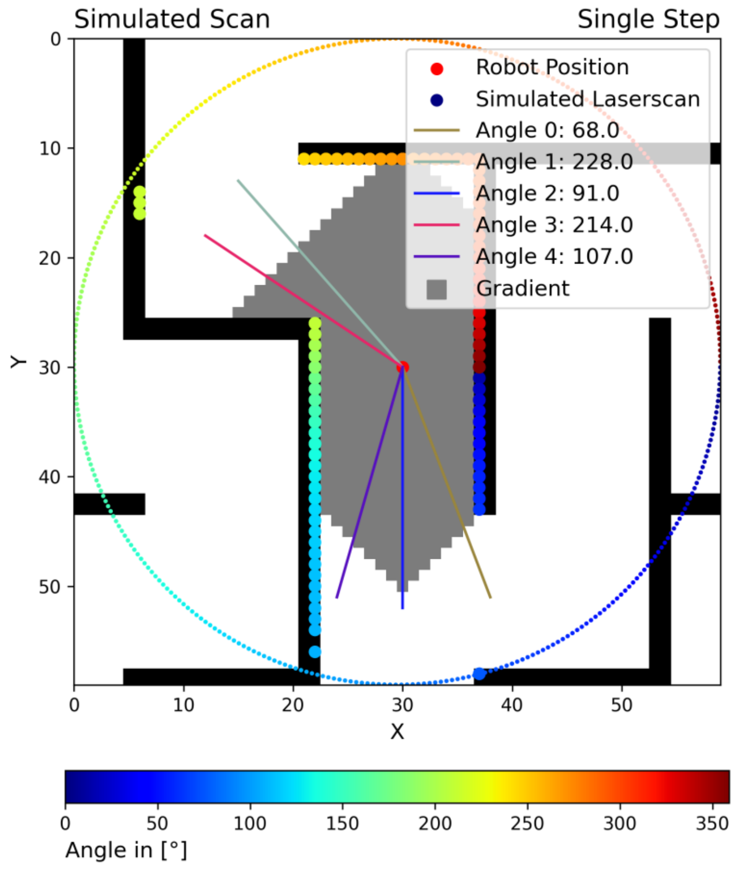
FINAL – SIMULATION – MAP/OCC GRID

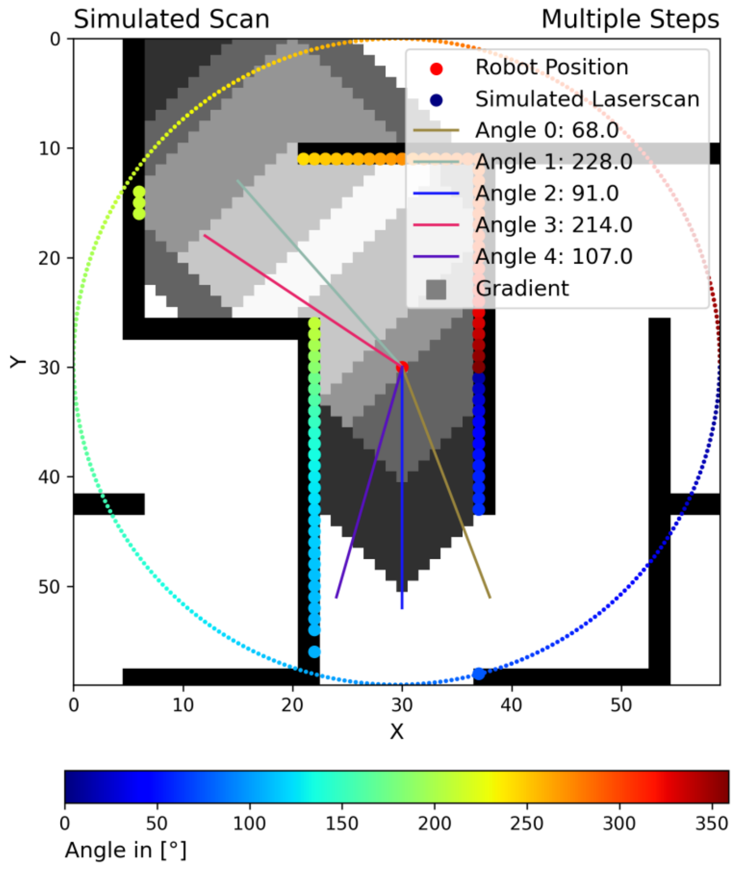
1. CALCULATION DECISION PROBABILITY

- **Gradient generation**
  - Imprint “footstep” to gradient map after each step
  - Behaves like Wavefront with fixed size of iterations
- **Probability calculation**
  - Check of endpoint of movement w. max step length/possible angle for occupancy
  - Calculate probability based on occupancy values for all possible angles

```
$x, y \leftarrow$ robot position
angle, distance $\leftarrow$ laserscan

for $\alpha, r \leftarrow$ angle, distance do
    $\Delta x, \Delta y = r \cdot cos(\alpha), r \cdot sin(\alpha)$
    $\text{occupancy}_{(\alpha,r)} = \frac{1}{\nabla \text{map}_{(x+\Delta x, y+\Delta y)}}$
end for
$p = \frac{\text{occupancy}}{\Sigma \text{occupancy}}$
```

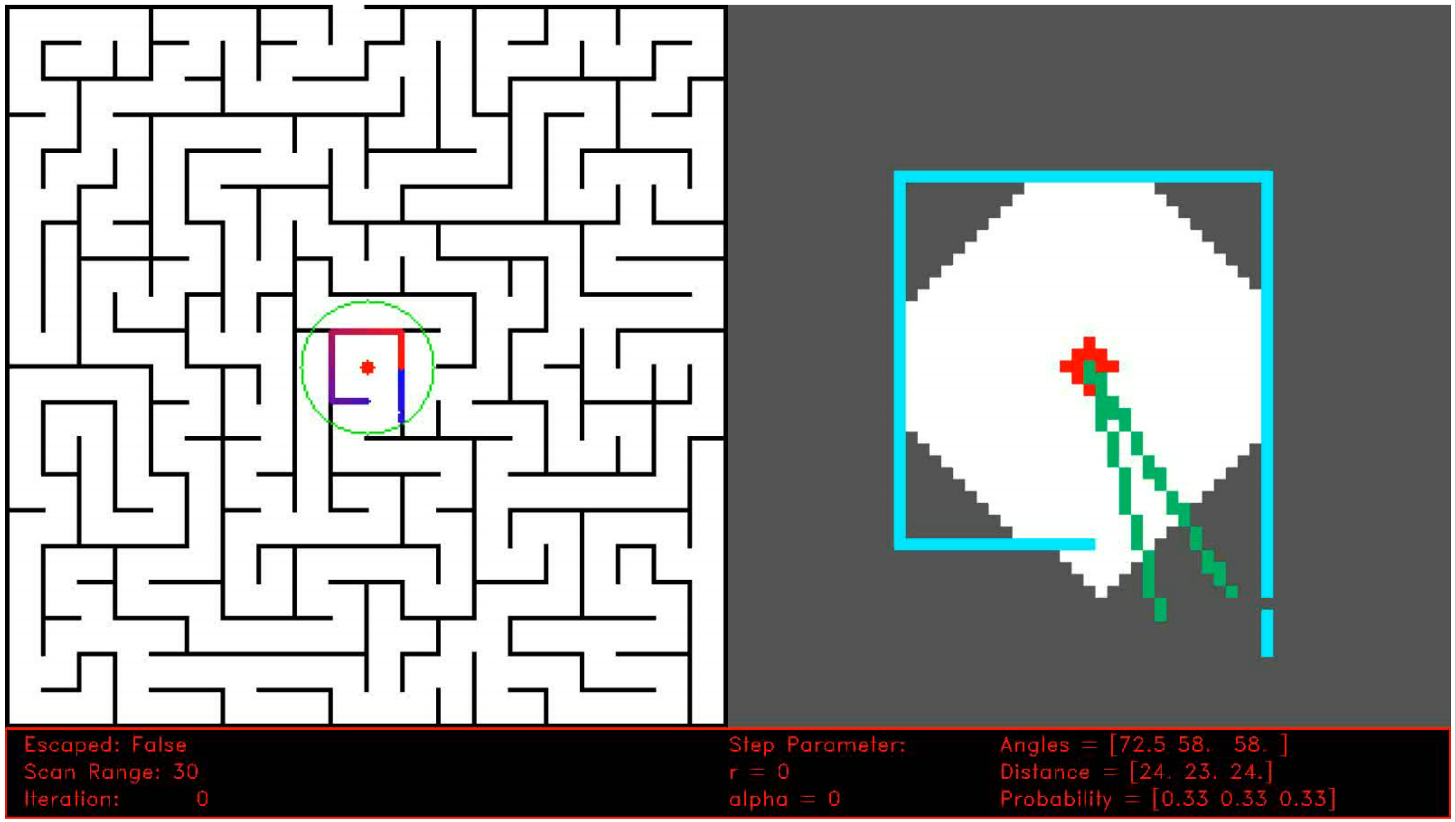




FINAL – SIMULATION

1. EXPLORATION PHASE

- Exploration based on occupancy grid
- Parameters:
  - Scan range (Laserscan max distance)
  - Footprint size, Gradient value (Occupancy grid generation)
  - Safety border value (Subtract from max step length)

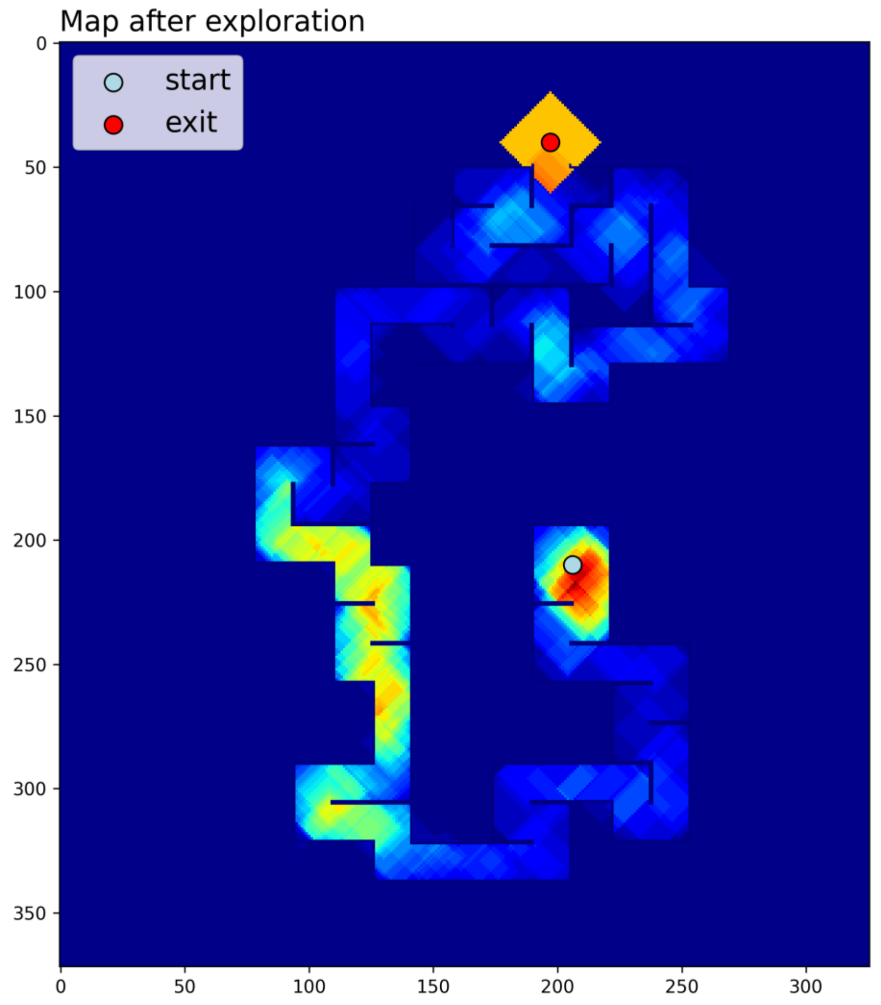


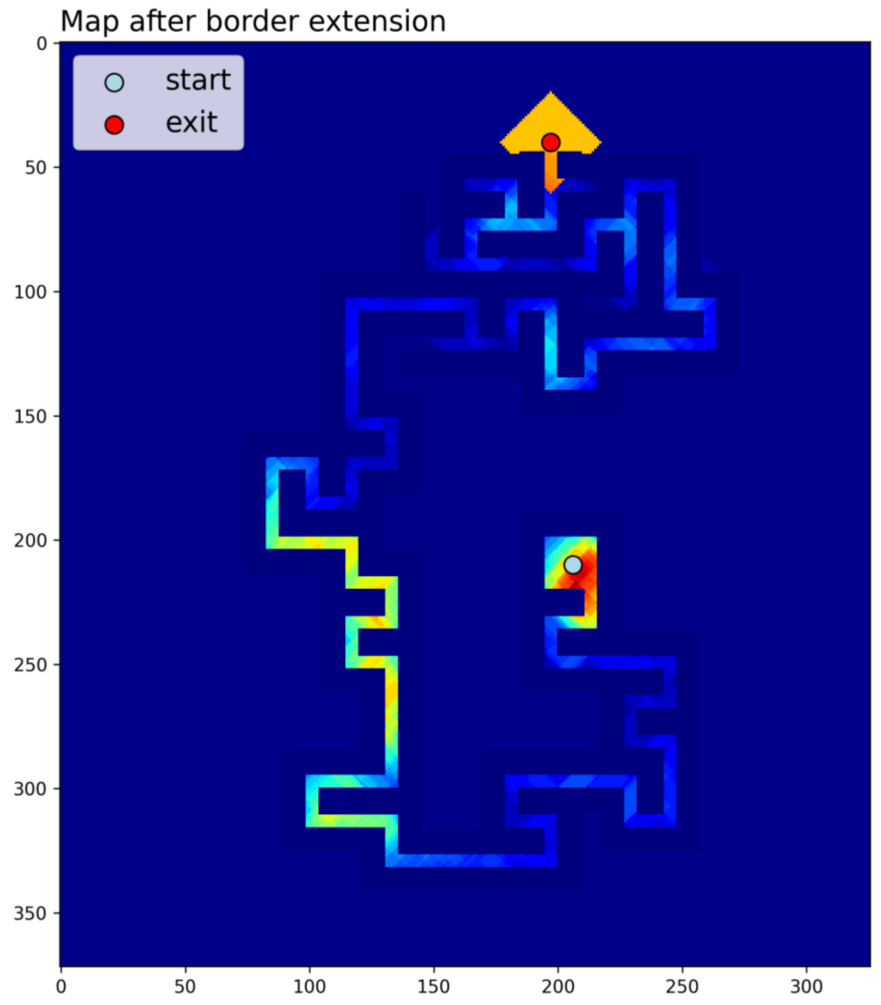
```
Escaped: False                                  Step Parameter:      Angles = [72.5 58.  58. ]
Scan Range: 30                                  r = 0                Distance = [24. 23. 24.]
Iteration:        0                             alpha = 0            Probability = [0.33 0.33 0.33]
```

FINAL– SIMULATION

1. BORDER EXPANSION

- Increase size of wall
- Ensures movement in middle of corridor
- Gradient values are reset after expansion

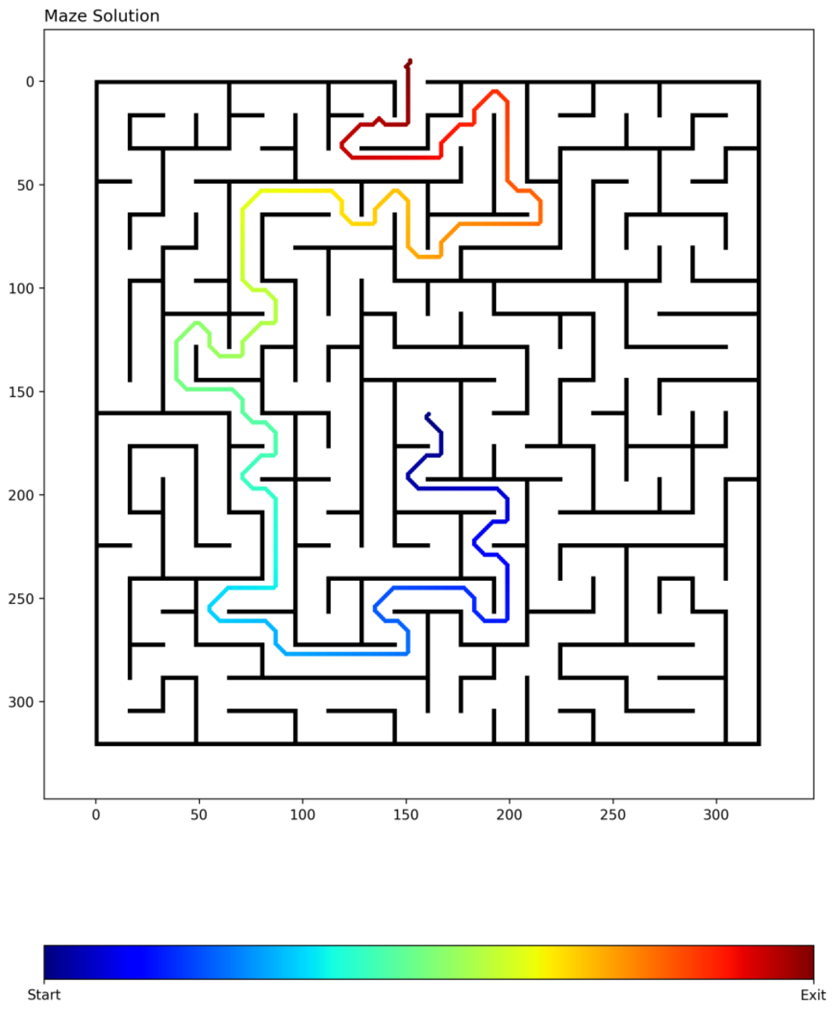


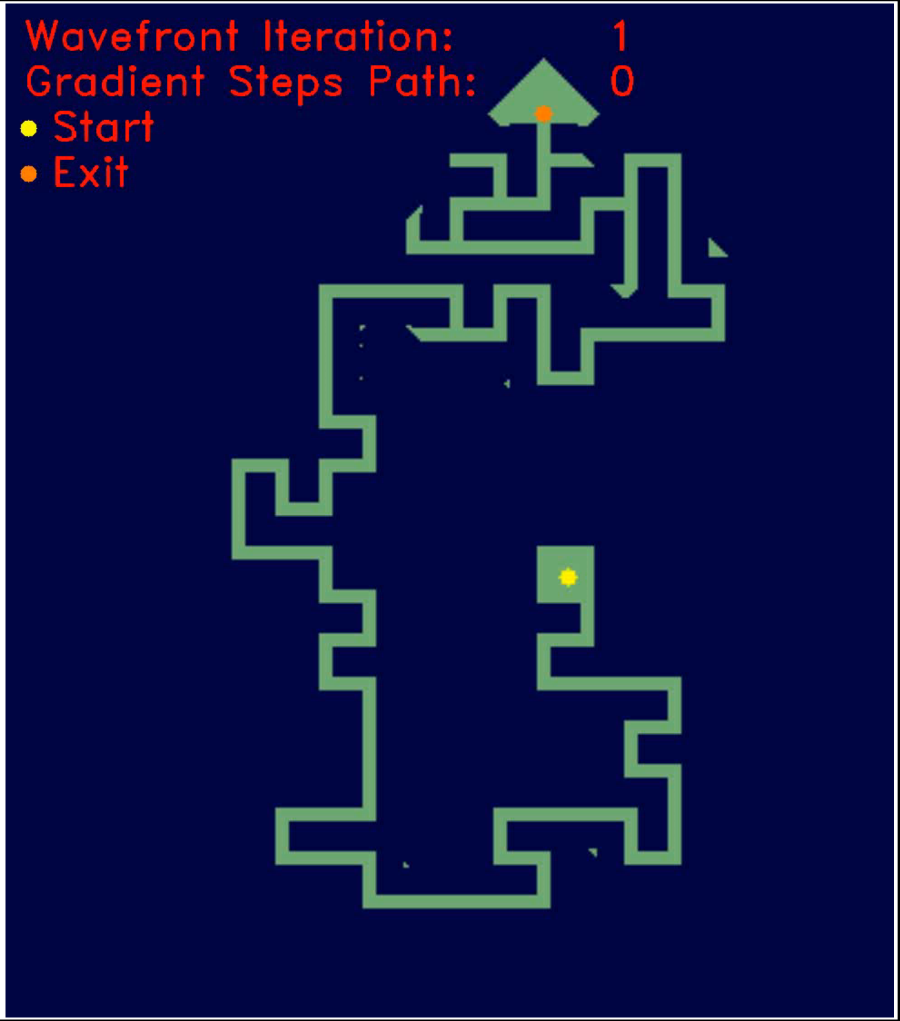


FINAL – SIMULATION

1. GLOBAL PATH PLANNING

- Wavefront Algorithm (BFS)
  - Goal: Exit of maze
  - Start: Spawn location
  - Environment: Map w. exp. borders
- Definition of neighborhood:
  - Moore (8 follow positions)
- Path finding
  - Gradient descent
- Known problems/bugs
  - Incomplete occupancy grid (missing connections after border expansion)-> Path blocked
  - Unnecessary corners in path (Python list returning first value for given gradient instead of visually best)





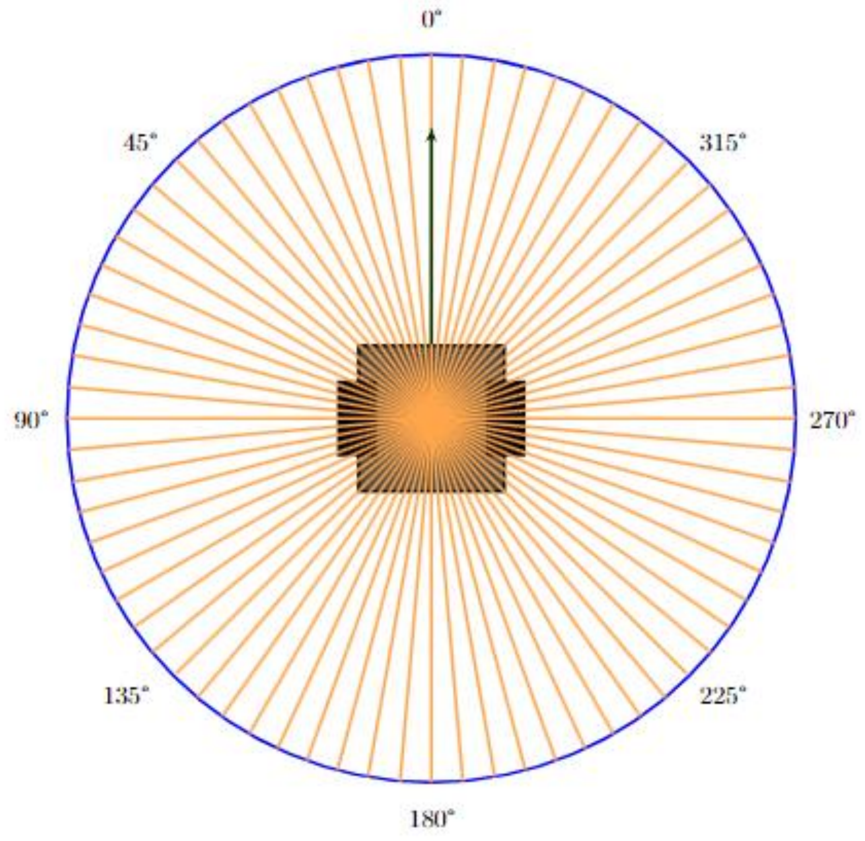
FINAL – IMPLEMENTATION

1. CALCULATION POSSIBLE MOVEMENT ANGLES

- For self navigation, LiDAR sensor data will be used to detect the walls around the robot.

- The LiDAR scan sensor (LDS-01) can accurately measure distances between 120 and 3500 mm with an angular range of 360° and an angular resolution of 1°.

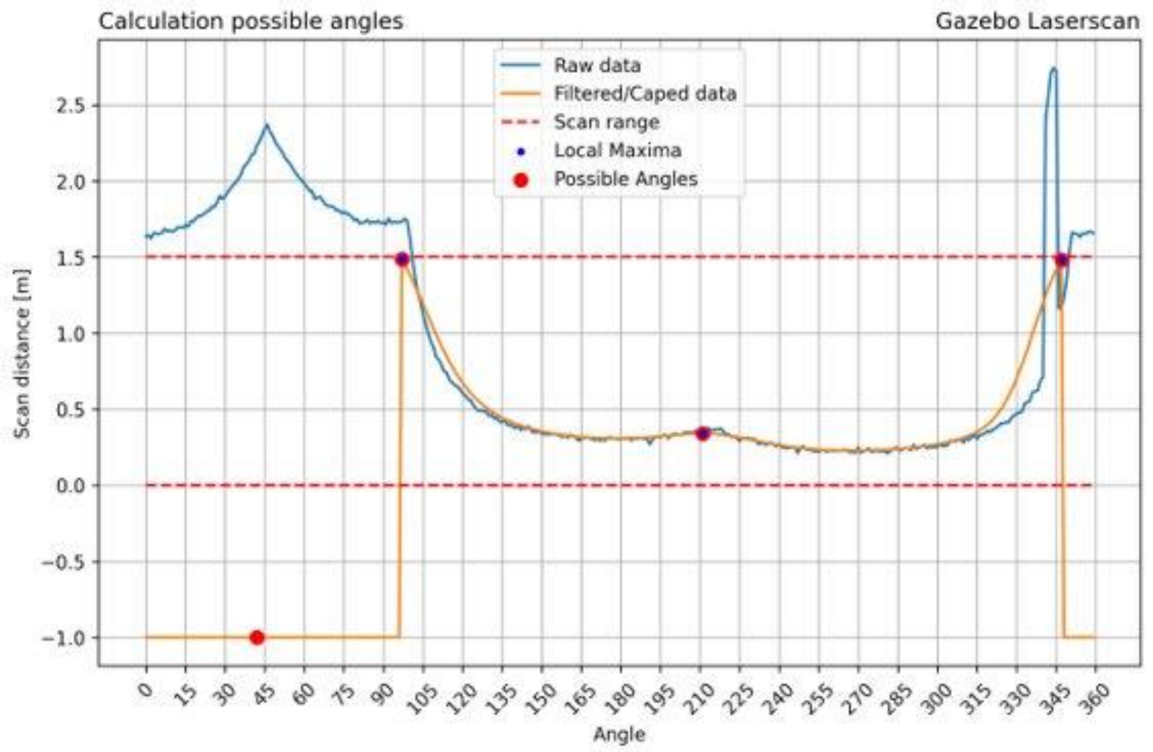
- In ROS, the laser_scan message from the sensor_msgs package will publish the scan values in the form of a tuple with 360 float values.



FINAL – IMPLEMENTATION

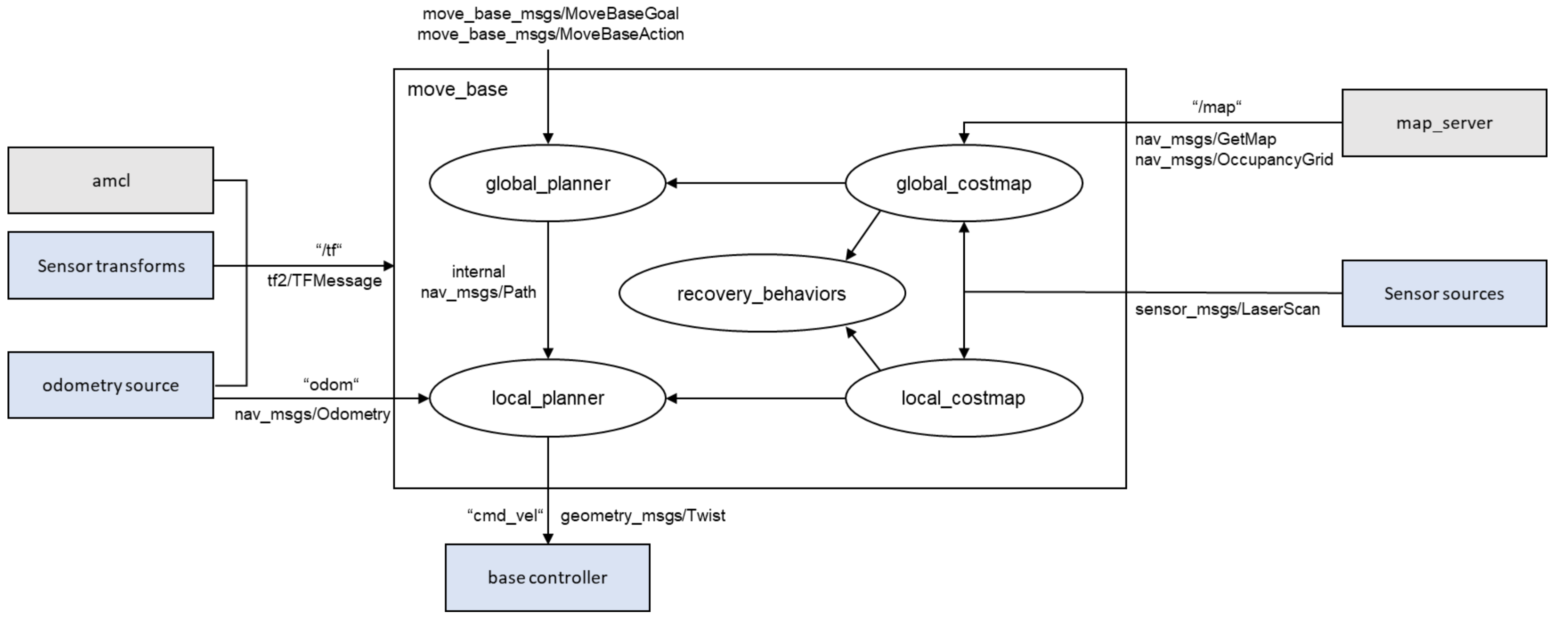
1. CALCULATION POSSIBLE MOVEMENT ANGLES

- For obstacle avoidance, we are looking through each of the 360 values.
- Like in the mathematical model implementation, the algorithm search for objects detected within the sensor range, and will grasp those that are the most far away from the robot as possible pathways.
- Also, for angles that the sensor don't detect anything within its range will be check and marked as possible pathways.
- The difference with the theorical object is that now we get some noise in the sensor readings. That's why we are applying a 1-D Gaussian filter with

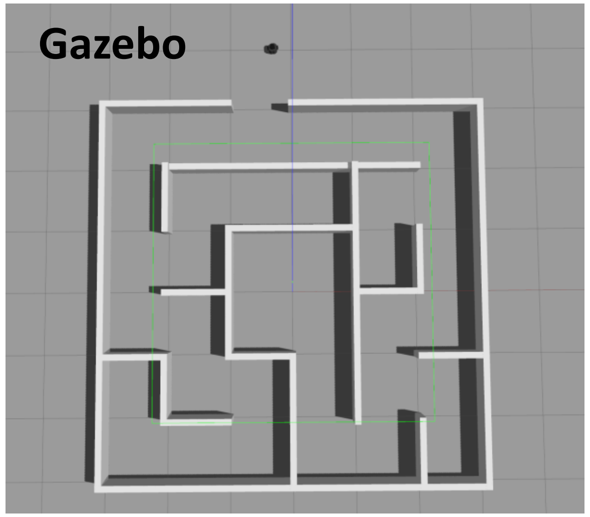


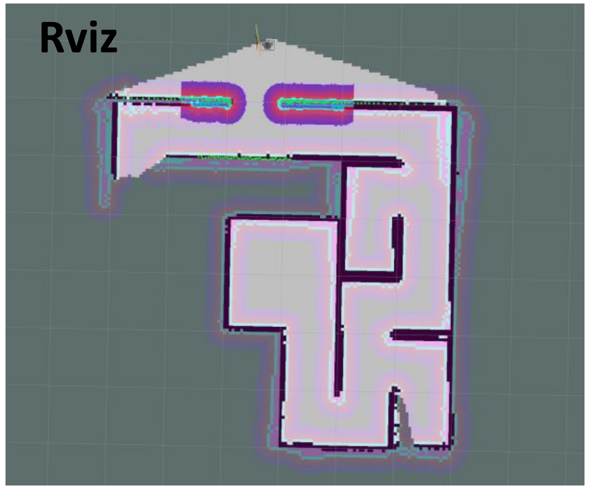
FINAL – IMPLEMENTATION

1. 2D Navigation Stack – Move_base package



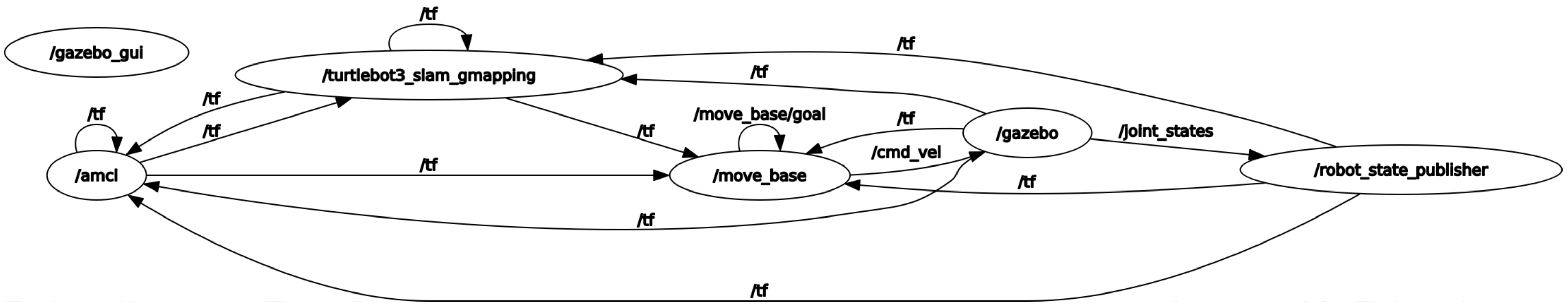
Navigation Stack Setup [1]





FINAL – IMPLEMENTATION

1. **ROS Graph – General Nodes**

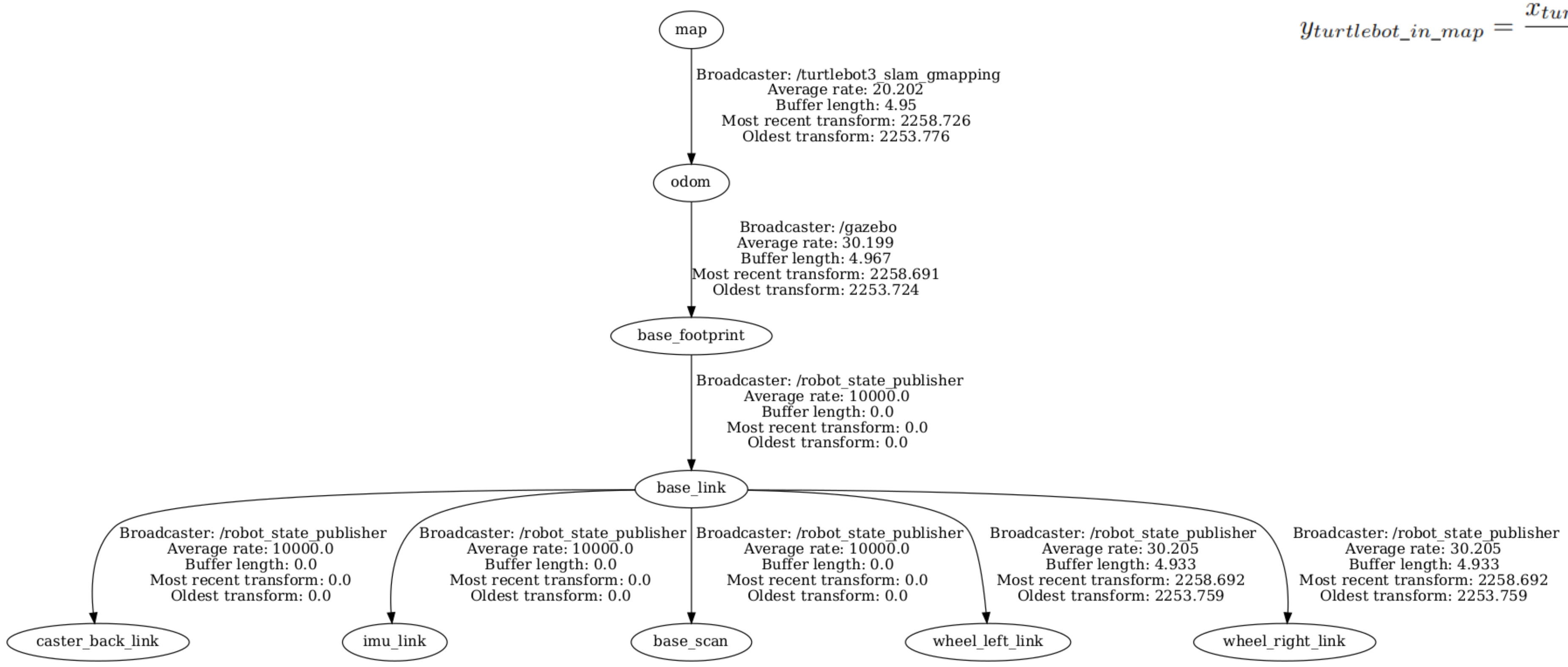


FINAL – IMPLEMENTATION

1. TRANSFORMATION TREE

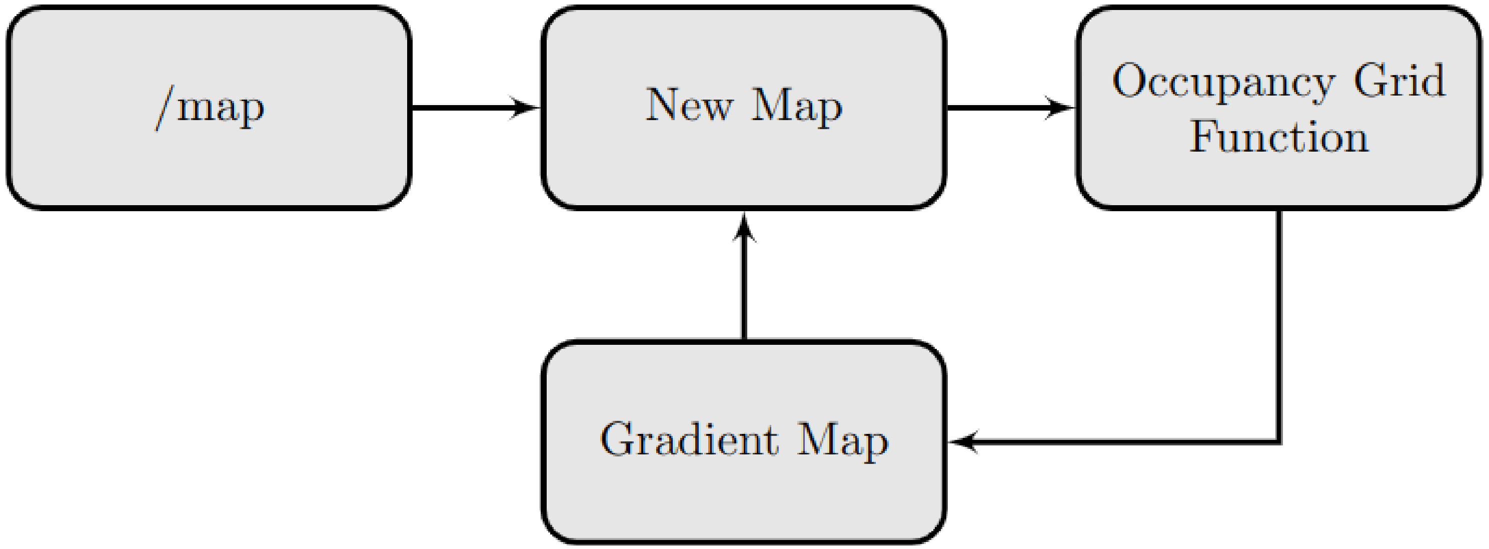
$$x_{turtlebot\_in\_map} = \frac{y_{turtlebot\_pose} - y_{map\_pose}}{resolution_{map}} \tag{1a}$$

$$y_{turtlebot\_in\_map} = \frac{x_{turtlebot\_pose} - x_{map\_pose}}{resolution_{map}} \tag{1b}$$


map
Broadcaster: /turtlebot3_slam_gmapping
Average rate: 20.202
Buffer length: 4.95
Most recent transform: 2258.726
Oldest transform: 2253.776
odom
Broadcaster: /gazebo
Average rate: 30.199
Buffer length: 4.967
Most recent transform: 2258.691
Oldest transform: 2253.724
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 30.205
Buffer length: 4.933
Most recent transform: 2258.692
Oldest transform: 2253.759
Broadcaster: /robot_state_publisher
Average rate: 30.205
Buffer length: 4.933
Most recent transform: 2258.692
Oldest transform: 2253.759
caster_back_link
imu_link
base_scan
wheel_left_link
wheel_right_link

FINAL – IMPLEMENTATION

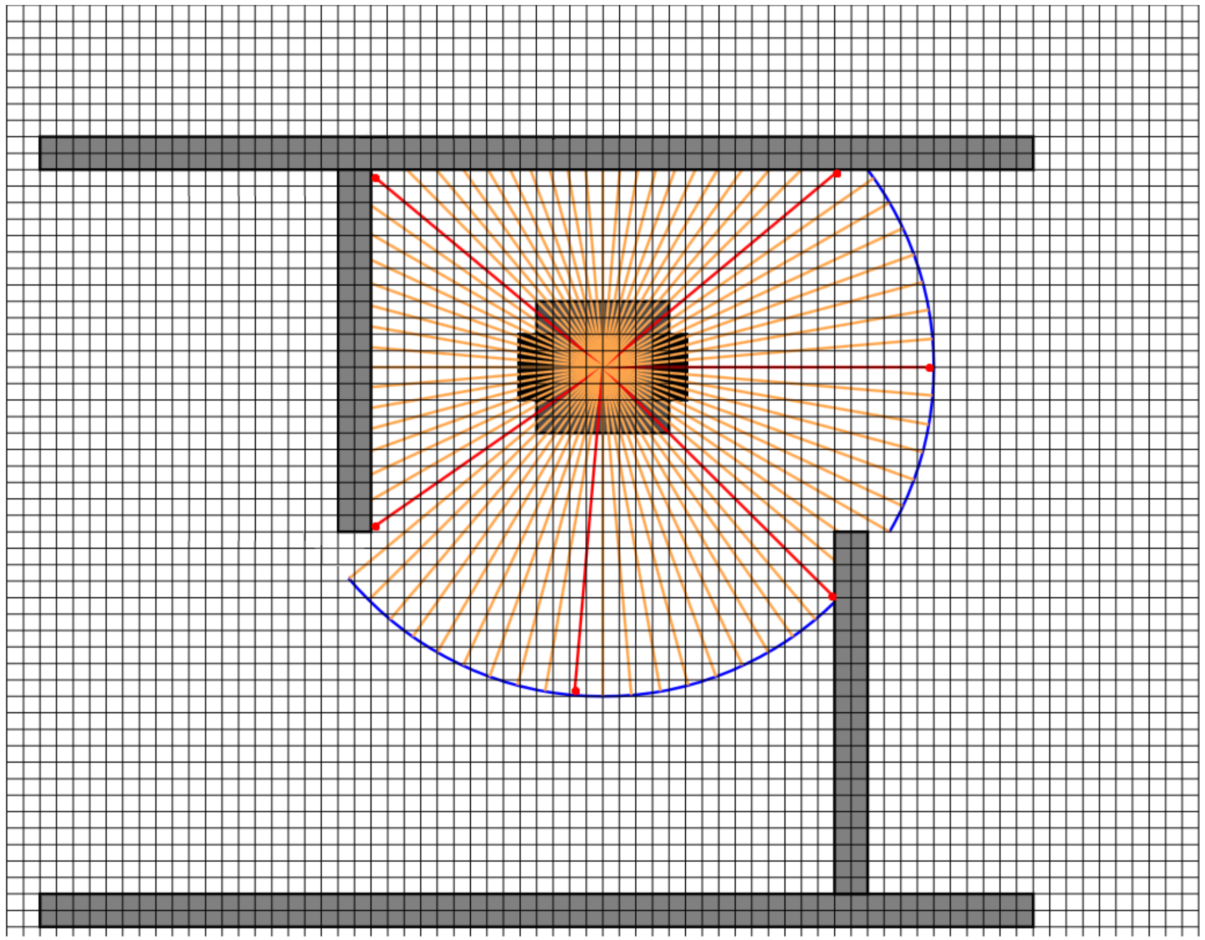
1. MAP UPDATE



FINAL– IMPLEMENTATION

1. **INTEGRATION OF OCCURENCY MAP AND LIDAR SCAN**

- After integration the gradient map to the Gazebo world, we can use the individual gradient values to determine which angle is better.
- During the FIRST ITERATIONS, there won't be much difference between each gradient, so the best angle selection will be almost completely random.
- After some iterations, the gradient values will be SIGNIFICANT at the moment of selection the best angle for the navigation



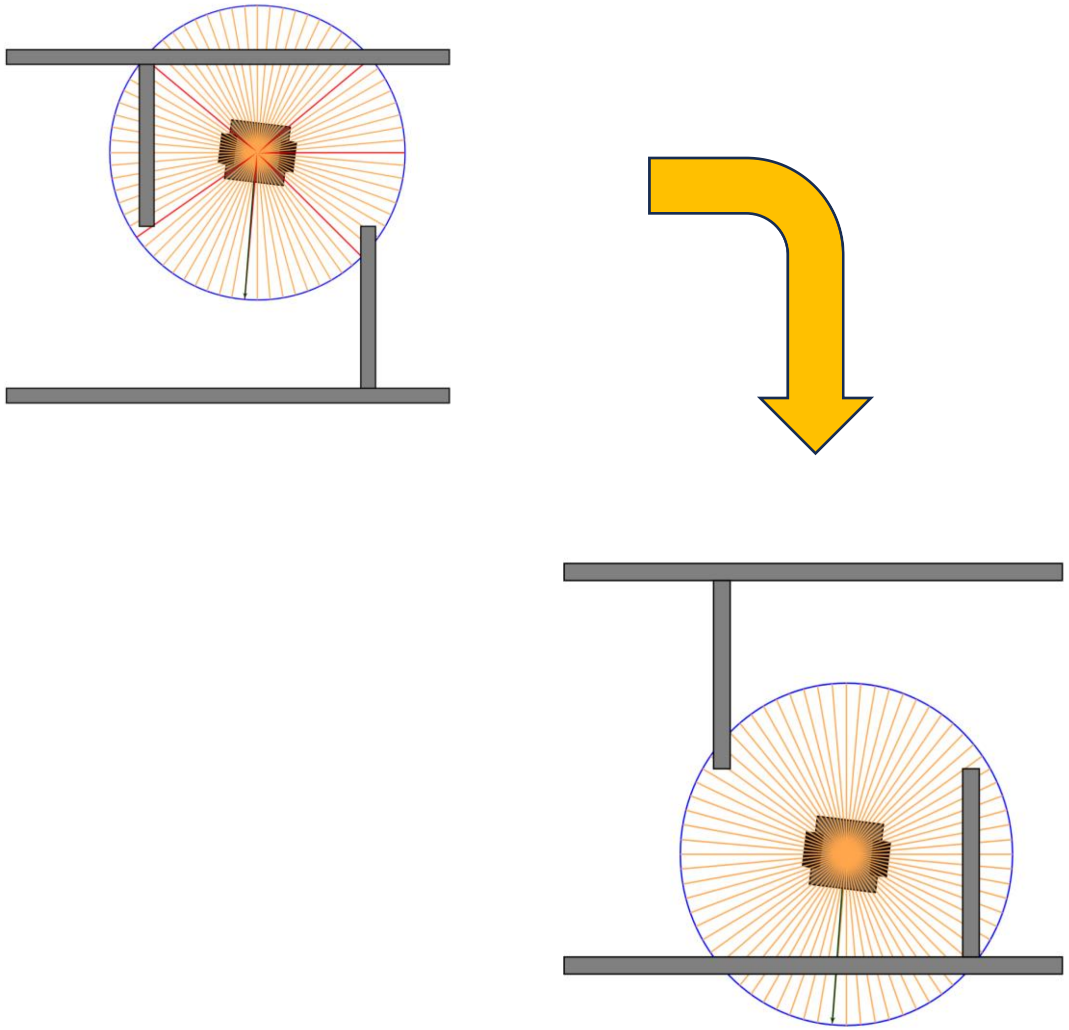
FINAL – IMPLEMENTATION

1. INTEGRATION WITH NAVIGATION PACKAGE

- To get the robot to move in the Gazebo environment, we used the Move Base package. After calculating the best angle with the optimal distance, we will transform the angle and the distance into a X and Y point in the Global reference frame.

- With the method send_goal from the Move Base package, we can give close local points for the robot to move.

- Now the robot has a way to explore the maze by its own, mapping and updating the gradient map every time it move to a new local point.



DEMONSTRATION

1.

CONCLUSION

1. **COMPARISON SIMULATIONS**
   A. Mathematical simulation
      - Ideal perception of the environment
      - Generated map is perfect representation of maze image(no rotation / artifacts)
      - Ideal action, by jumping directly to target location

   B. Gazebo simulation
      - More uncertainty and noise
      - Sensor data needs to be filtered
      - High tolerances for local movement
        - Corner location cannot be reached due to robot size

2. **OUTLOOK**
   A. Mathematical simulation
      - Weighted probabilities based on angle type
      - Non-linear footprint
      - Replacement border expansion

   B. Gazebo simulation
      - Porting of wavefront / solution determination
      - Map Improvement

   C. General
      - Database with all experienced mazes (Maps+Solution)
        - Maze recognition during exploration
        - Trying of previous solution

THANK YOU

REFERENCE

1. E. Marder-Eppstein, “Navigation.” http://wiki.ros.org/navigation?distro=noetic, 2020. Accessed: 15.06.2023.