Exhaustive Search Scan Matching

1. Introduction

This test consists of implementing an algorithm intended to find the best pose of a rangefinder on an occupancy grid. The test aims to assess the following:

- Software design skills
- Command of the C++ language

2. Problem Statement

Assume a binary flat 2D world that consists of free and occupied space, and a 2D rangefinder in this space that can measure the distances from its center to occupied space around it within a specified field of view.

Assume further that the rangefinder operates in a part of the world represented by an occupancy grid (referred to as ***grid*** in the remainder of this document) where each cell represents whether this cell is free or occupied.

Given an approximate metric pose of the rangefinder in the world given as (***x*** meters, ***y*** meters, ***theta*** radians) plus tolerances on x, y and theta, the goal is to find the correct pose by matching the observation of the rangefinder to the occupancy grid.

3. Data Representation

3.1. Occupancy Grid

The occupancy grid representation includes the following information:

- Occupancy state of cells represented by uint8 values with 0 for free and 255 for occupied
- The pose of the ***grid*** in the world (i.e. the transform between grid coordinate frame and world coordinate frame) is represented by the ***x***, ***y***, ***theta*** coordinates of the center of the grid cell (0, 0).
- The ***height*** and ***width*** of the ***grid***
- The resolution of the ***grid*** is 0.05 meters

An oriented point (x_g, y_g, theta_g) in the grid coordinate frame has the following coordinates in the world frame:

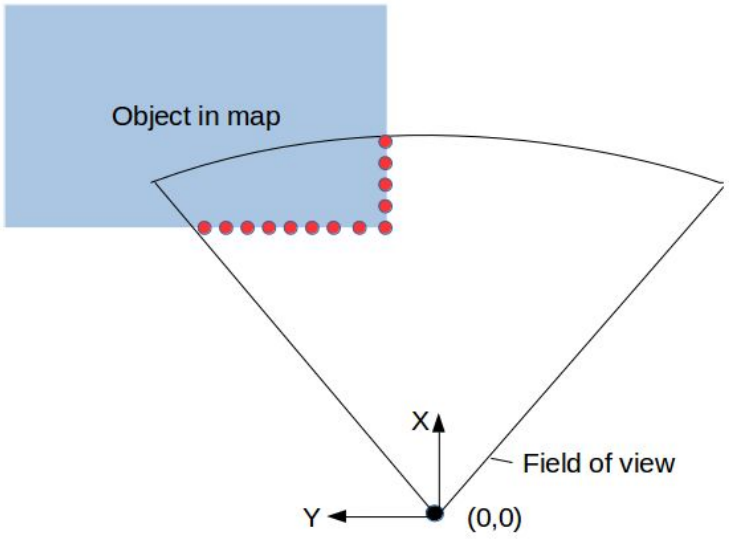
x_w = cos(theta) * x_g - sin(theta) * y_g + x
y_w = sin(theta) * x_g + cos(theat) * y_g + y
theta_w = theta_g + theta

Conversely, a point (x_w, y_w) in the world is equivalent to:
x_g = cos(theta) * (x_w - x) + sin(theta) * (y_w - y)
y_g = -sin(theta) * (x_w - x) + cos(theta) * (y_w - y)
theta_g = theta_w - theta

3.2. RangeFinder Data

The rangefinder measures the distance (range) from its center to the closest obstacle along different discrete directions.

- The output is represented by a list of range values measured along rays uniformly distributed across the sensor field of view (***fov*** radians) starting from ***-fov/2*** to ***+fov/2*** with respect to the main axis of the rangefinder.
- The increment between every two consecutive rays is the resolution (***res*** radians).
- ***-fov/2*** is guaranteed to be the first ray, the last ray could be right before or exactly at **+fov/2**.
- The rangefinder can measure up to **max_range** meters. Out of range values are represented by any arbitrary value higher than ***max_range***. For example if ***max_range*** is 20 m a value of 25.0 means that the Lidar observed only free space along the corresponding ray.



4. Algorithm To Be Implemented

The main task of this test is to solve the above pose estimation problem by performing an exhaustive search of all possible poses within the tolerance area and find the pose that best matches the ***grid***. The search space is discretized in x, y by a ***linear_search_resolution*** and in theta by ***angular_search_resolution***.

Input: Grid, rangefinder data, approximate pose in **World coordinate frame**, search resolutions
Output: Found pose in **World coordinate frame**.

The algorithm proceeds as follows:

- Transform the approximate pose from world into ***grid*** coordinate frame.
- Transform the rangefinder data into a Euclidean point cloud in ***grid*** coordinate frame as follows:
    1. Transform the rangefinder data into point cloud in rangefinder coordinate frame
        - A range (r) along an angle (theta) is transformed into a point x, y as follows:
            - x = r * cos(theta)
            - y = r * sin(theta)
    2. Transform the generated point cloud into ***grid*** coordinate frame.
- Generate a list of angles, x values, y values covering the search area to be evaluated during the search:
    - The lists are determined using the approximate pose (in grid frame), the tolerance parameters, the linear_search_resolution and the angular_search_resolution.

- For each angle theta_i in the list of angles:
    - Rotate the point cloud in grid coordinate frame by theta_i
    - For each x_i in the x values list:
        - Translate the rotated point cloud by x_i
        - For each y_i in the y values list:
            - Translate the point cloud (already translated by x_i) by y_i
            - Transform the metric points of the cloud into grid discrete coordinates using the grid resolution.
            - Determine the score of the hypothesis (x_i, y_i, theta_i) by summing the values on the ***grid*** that correspond to the points in the translated point cloud.
- The best pose is the (x_i, y_i, theta_i) with the highest score. If there are multiple poses with same score, it is preferable to select one that is closest to the initial approximate pose.
- Transform the best pose back into World Coordinate frame.

4.1. Assumptions

- The initial positions of all rangefinder queries are within the ***grid*** up to the tolerances provided.

> Note: The equations to apply a rotation theta to a point x, y are as follows:
> r_x = cos(theta) * x - sin(theta) * y
> r_y = sin(theta) * x + cos(theta) * y

5. Design and Implementation

1) Create all the data structures and classes required to implement the solution in a modular architecture.
2) Create a class Matcher with the appropriate interfaces.
3) The implementation should be done in C++ following best practices for production quality code.
4) Please follow Google's C++ Style Guide for style.
5) The code should compile with C++14 with no other external library dependencies.
6) Please discuss how you would ensure the correctness of your code.