Third Milestone

The problem

The goal of the milestone 3 was to implement an algorithm for detecting the field width, finding the absolute position and heading of the robot on the field and create a driving algorithm that uses the position and heading information to keep the robot inside of the field.

Our approach

Field width detections

The field width is calculated in the field_width_node which takes as an input the final map from the map_node and calculates based on that the field width. Finally, the result is published to field_width_node/width topic.

The field width detection has following steps:

- *Receive map of the environment.*
- *Remove all other than green points in the map .*
- *Check if there are enough green points, otherwise stop.*
- *Find first line using PCL RANSAC and remove it from the original cloud.*
- *Find second line if there are still enough points left.*
- *Calculate the distance and the angle between the found lines, check if the lines are parallel and the distance makes sense.*
- *Publish width as a vector3 message.*
- *Calculate average of individual width measurement and stop after 30 samples (after that it will publish always the same width since the width of the field is constant).*

Location detection

For location detection two different approaches were tried, one based on labeling and on based on PCL iterative closest point algorithm.

Labeling approach

It is implemented in the localizer.cpp, and functions based on calculating the closest object in the ideal map, and associating it’s label to it. Objects in the ideal map are labeled clockwise, beginning with the map origin, which is assumed to be the left-most pole on the top.

Iterative closest point approach

This approach is rather simple and it works quite reliably. The implementation of this approach is found in the location_node.

The approach the following steps:

- *Receive map from map_node and field width from field_width_node.*
- *Remove pucks from the map*

- Calculate number of points representing poles, blue goals and yellow goals.

- Calculate direction vector of the field based on two centroids of pucks or goals. Centroids of goals are used if both goals are detected, otherwise one goal and pole centroid is used.

- Calculate corresponding vector for the ideal field map.

- Calculate position difference and angle difference between vectors.

- Translate the input map based on the vector distance and angle difference, this brings the points of the measured map roughly to the correct position (e.g. near the ideal map).

- Do final transformation using PCL iterative closest point algorithm, and get the transformation.

- Combine the transformations by multiplying the transformation matrices and publish the transformation from map to robot1/odom frame.

Driving algorithm

The driving algorithm was implement so that the robot has five different states: drive_to, drive_random, rotate, move, stop). The drive_to state tries to move the robot to a specific point on the field, it does not use any path planning so it does not find the best routes to a position. The drive _random mode works similar to the drive_to mode, except that it does not drive to a specific location.

Drive_to and drive_random modes use collision avoidance which is based on the lidar data and on the closest wall distance (the closest wall distance is the distance to the edge of the field in front of the robot). Based on the closest obstacle the robot rotates in the opposing direction and decreases speed as the obstacle gets closer. If the closest wall is closer than a threshold value the direction of rotation is determined always by the wall, to reduce unnecessary rotation movements of the robot.

Rotate and move modes are for moving the robot without collision avoidance and are needed in the initialization process when the location of the robot is unknown. Move state will be useful for driving backwards to leave a buck on a goal. In the stop state the robot does not move.

Work division

The tasks were divided in the following way for this milestone:

Bálint: Field width detection

Reza: Location detections

Jaakob: Driving algorithm