Worksheet 6 – The Maze Challenge

1. Introduction

The aim of this work is to develop an autonomous ROS-enabled robot, built from a LEGO Mindstorms kit, capable of navigate and exit a maze (see Fig. 1). The robot should also search and find a hidden object (e.g., magnetic dipole) in the maze. During the final demonstration, keep in mind that the maze structure could differ from the one depicted in the Fig. 1.

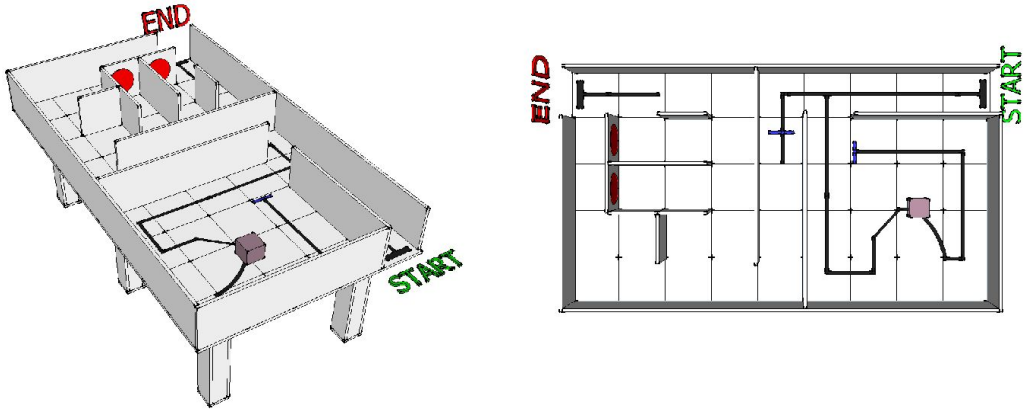


Fig. 1 - Example of a possible maze structure with approximate area of 2,7 x 1,5 m²

2. The Maze

The maze is divided into two zones. In the first one the robot must follow a line drawn in the floor to navigate all the way to the second zone. The line is made using black electrical tape with a width of 19 mm. The possible type of line intersections are depicted as follows:

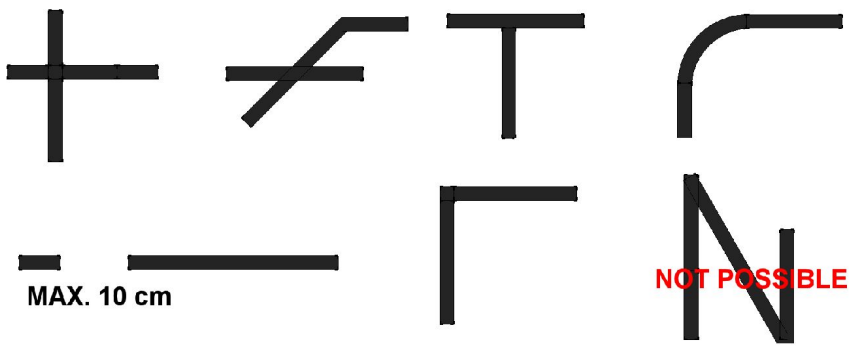


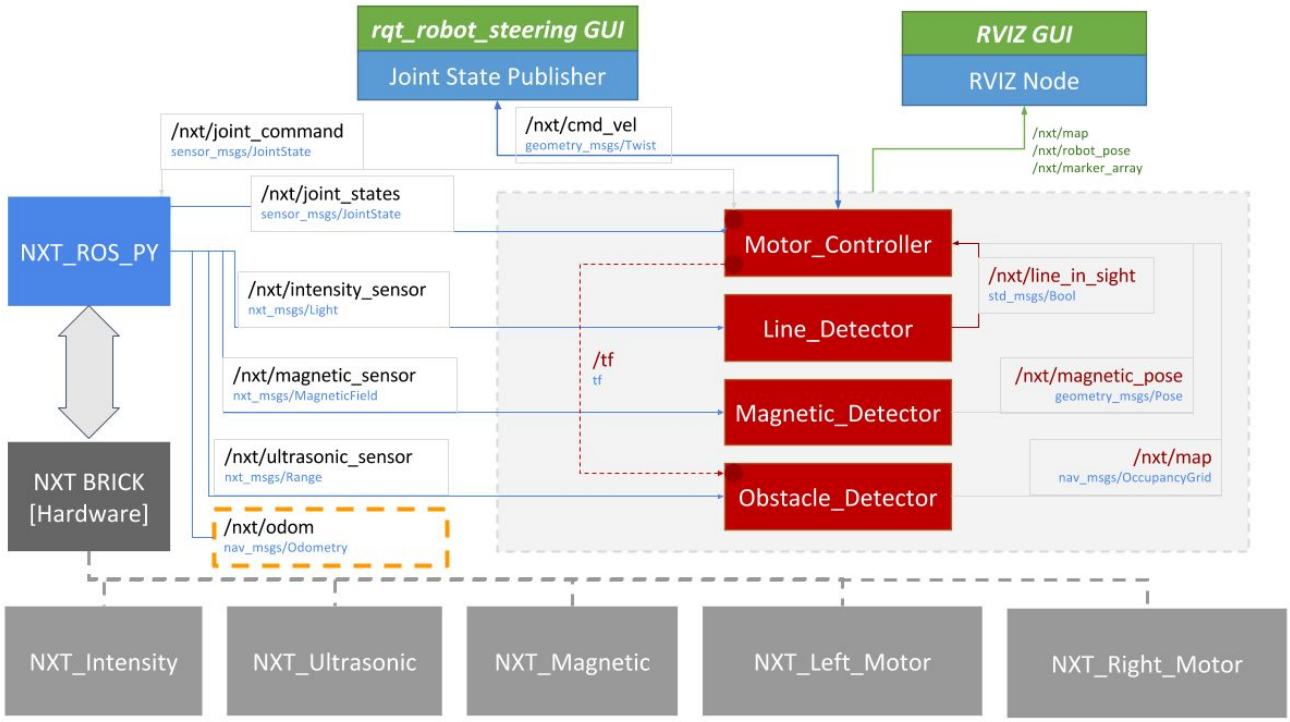
Fig. 2 - Some examples of line intersections and interruptions.

In the second zone there is no path line and the robot must use ultrasounds to assess its surroundings and navigate towards the exit. The entry and exit locations of the maze are identified respectively by the marks shown in Fig. 3 and virtually identified as "START" and "END" in the example of Fig. 1.

(a) (b)

Fig. 3 - Symbols denoting the START (a) and END (b) markers placed in the maze.

3.Implementation



First, you should download the new version of the nxt package from moodle. The new version contains a node that provides odometry information (see Apendix A) in topic ***nxt/odom*** (nav_msgs/Odometry). The final objective of this worksheet is to implement the necessary features that will enable the NXT robot to solve the problems depicted in section 4.

4. Evaluation

Objectives	Grade (%)	Description
❑ **Robust line following behavior**	**40**	The robot should be able to follow black lines under different light conditions and recover from path interruptions.
❑ Follow straight lines	5	---
❑ Follow curved lines	5	---
❑ Handle 90 degree curved lines	10	---
❑ Recover from path line interruptions	15	Upon detecting an interruption on the path, the robot must search and successfully return to the black line.
❑ Intelligent light adaptation	5	At the beginning of each run, the robot must calibrate its light sensor with actual intensity readings (black and white).
❑ **Safe maze navigation**	**40**	Using the ultrasonic sensor, the robot builds its own maze map for the traversal of the Zone 2 and search for the hidden object.
❑ Obstacle avoidance	15	Upon detecting an obstacle in its path, the robot should contour it and resume.
❑ Build and use maze map	25	An internal representation of the maze must be built and used by the robot.
❑ **Locate the hidden object - magnetic dipole**	**10**	The robot should be able to search and locate for the magnetic dipole, which location is changed every run.
❑ Detect the magnetic dipole	5	A magnetic field must be detected by the magnetic sensor installed in the robot.
❑ Search for magnetic dipole	5	Before exiting the maze, the robot should search for the magnetic dipole in the Zone 2 and output its global position.
❑ **Work report**	**10**	**Mandatory -** Report about the approach to the problem, the ROS ecosystem design and both encountered challenges and taken solutions.

The resulting NXT package must be submitted in the Moodle platform, as a zipped file, no later than June 9th, 23:55.

Appendix A

Differential robots

Kinematics

Differential robots can only travel in circular trajectories composed of linear and angular velocities (see Fig A1). This means that they are not able to move sideways (non-holonomic).

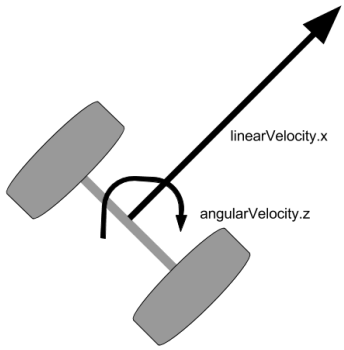


Fig. A1 - Differential robot movement.

Odometry

Using data from motion sensors it is possible to estimate position changes over time. In the case of the NXT-based differential wheeled robot the data from wheel encoders is integrated to estimate the position (nxt/basefootprint) relative to a starting location (odom) (see Fig A2). This method is sensitive to cumulative errors due to the integration of velocity measurements over time to give position estimates. For instance, if a wheel slides during motion its estimate will introduce errors on the robot's pose. The same is valid for rough terrains where wheel travel alone is not a good indicator for estimating position (See Fig A3).

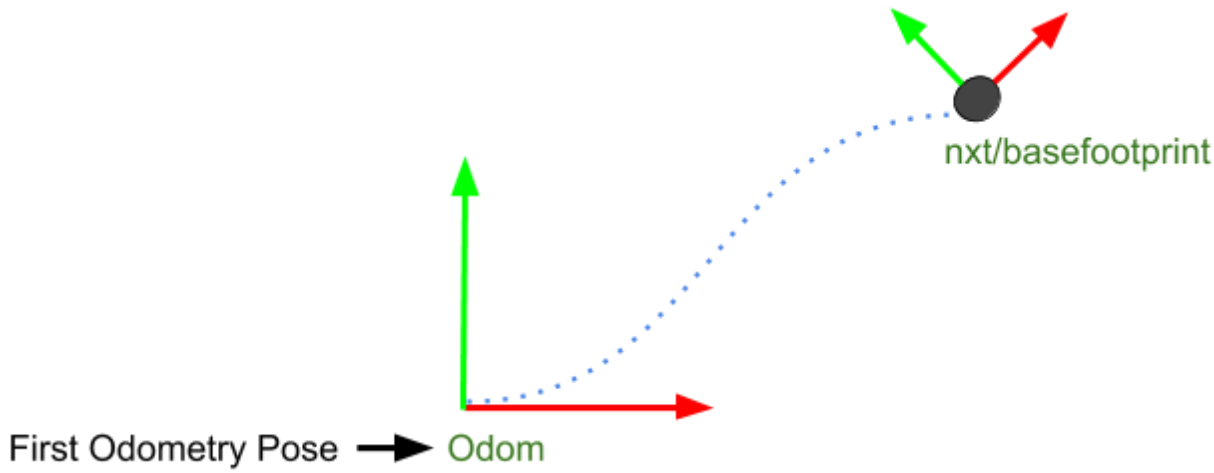


Fig. A3 - Odometry estimation of the robot's pose

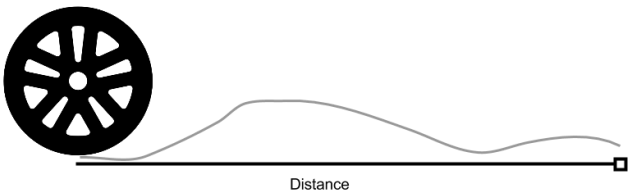


Fig. A3 - Travelling over rough terrain

Another factor that undermines wheel-based odometry is the wrong parameterization of wheel diameter/radius or wheel base. If the diameter is wrong the position will be overestimated or underestimated (see Fig A4). A calibration process should be repeated every time a change in wheel base or wheel diameter (e.g. different tyre pressure) is made (see Fig A5)

NOTE: You can change these parameters on the ***lego_robot.yaml*** file.

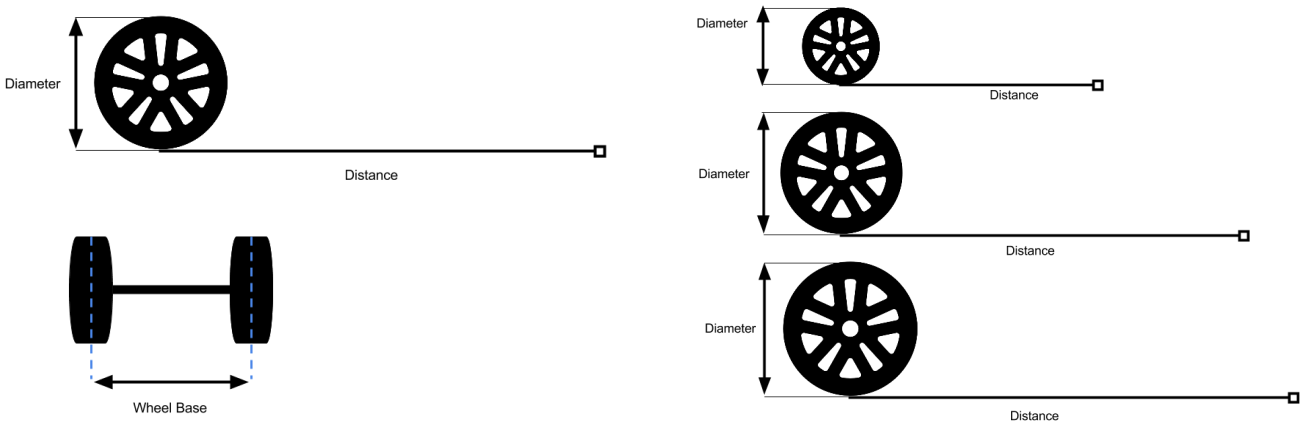


Fig. A4 - Wheel base and diameter and its influence on distance estimation.

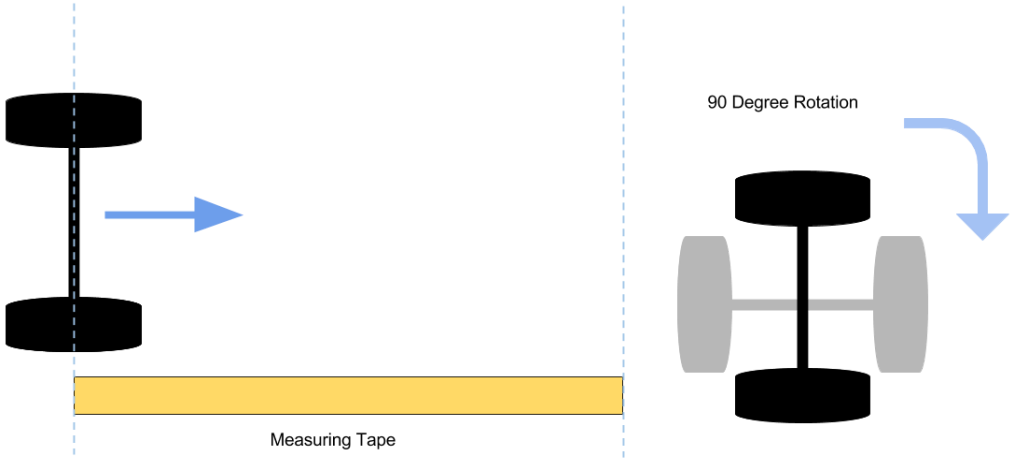


Fig. A5 - Calibration process where the robot's estimate is compared with external measurements.

Appendix B

Maze Map

The maze map is an internal representation of the robot's surrounding based on its ultrasonic range sensor. The end result should be published in the form of a nav_msgs/OccupancyGrid message, marking the obstacle map on a fixed coordinate frame (the one established by the odometry as the first pose).

So you have to subscribe to the nxt_msgs/Range message coming from the */nxt/ultrasonic_sensor* topic, listen and lookup the transform between the *frame_id* frame related to the ultrasonic message (in the message header) and that /odom frame of reference. You should recall on what you have done on a previous TF related lesson (if not, see Writing a tf listener).

With the rotation and translation read between sensor frame and /odom, you should be able to change the measured range in angular coordinates to an /odom world referenced cartesian coordinate pair, *(x,y),* to mark the measured obstacle into the *nav_msgs_OccupancyGrid* data vector (see the cell defined values for unknown, free and occupied space of values in OccupancyGrid Message).

Remember that if the measured range, *range* in *nxt_msgs/Range*, exceeds the maximum, range_max, no obstacle has been detected. If a range is in between the two you should mark, or affect, the respective world cell in the grid, with a trust, of some sort, for example a weighted average between current and past measurements on each cell.