Simulated Robot assignment to navigate SEYO World using ROS Melodic and Gazebo 9.0

By Rainer Doller

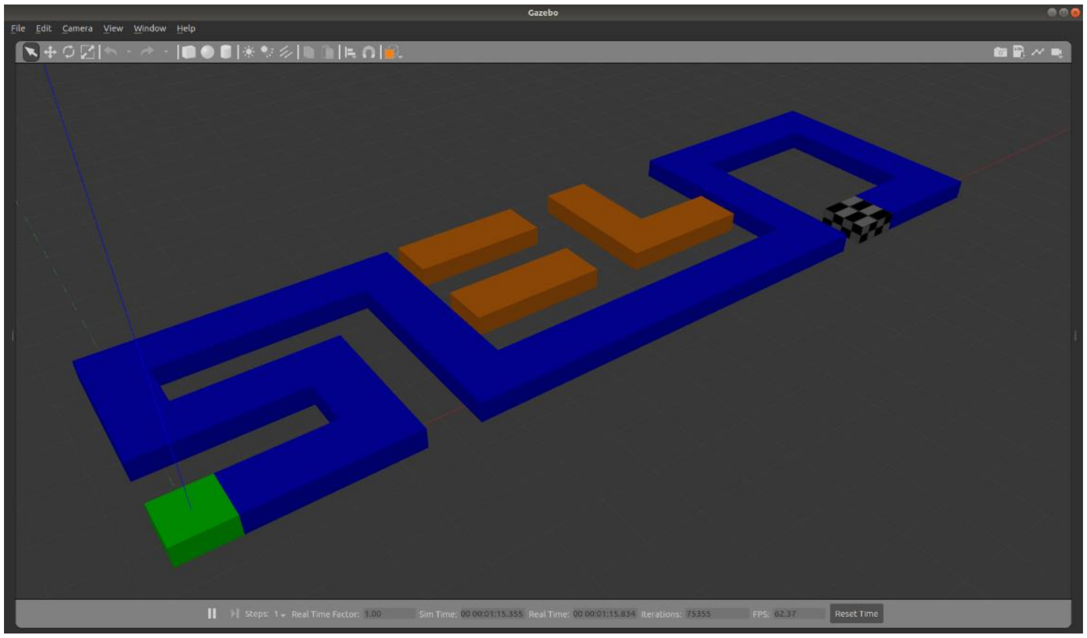

Gazebo
File Edit Camera View Window Help
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:01:15.355 Real Time: 00 00:01:15.834 Iterations: 75355 FPS: 62.37
Reset Time

Figure 1: The raised platform within Seyo World.

Objectives:

1) Create a robot model that has a starting position on the green block, as indicated in Figure 1, and that can adequately navigate the raised blue platform, in order to reach the final destination as indicated by the chequered block.
2) Allow the robot to be controlled with three high level movement instructions:
    - Move forward by a specified distance [fwd <distance>]
    - Turn right (rotate clockwise by 90 degrees) [right]
    - Turn left (rotate counter clockwise by 90 degrees) [left]
3) Include a launch file that starts the required Gazebo simulation and runs the necessary ROS nodes that control the simulated robot.

Assumptions:

1) The robot should be kept small, so that it can comfortably navigate the required path.
2) The robot design should be kept minimalistic in order to simplify the navigation controls.
3) It was assumed that the odometry readings, obtained within the simulation, are equivalent to the readings that could be obtained from by the necessary pose tracking sensors that the real robot would use.
4) In order to control the robot from point to point, along the required path, an accuracy tolerance of within 5cm was assumed to be adequate.

Design:

The initial robot model design is illustrated in Figure 2:

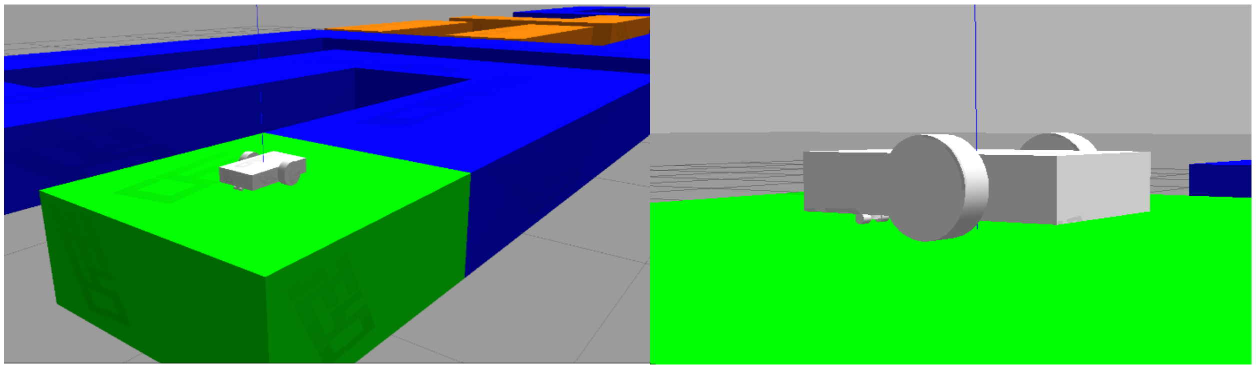


Figure 2: Views of the initial robot design with 2 front wheels and two small back wheels mounted on a pivot.

The initial design was chosen so that the necessary navigation requirements could efficiently be met using a two wheeled dive. The trailing back wheel design was chosen because there did not seem to be an equivalent “ball joint” within in the URDF file format (used within ROS), as opposed to the ball joint available in the SDF file format, which is used by Gazebo. The need for this ball joint is to simply model a castor wheel, which can support the robot body without inhibiting the drive motion.

This initial design did not prove to be adequate because the back wheels would not consistently behave as required and would at times jam when the robot model would turn. The offset of the front drive wheels, from the middle of the robot, caused the navigation to be more challenging than necessary. Therefore the next and final design was chosen, as illustrated in Figure 3.

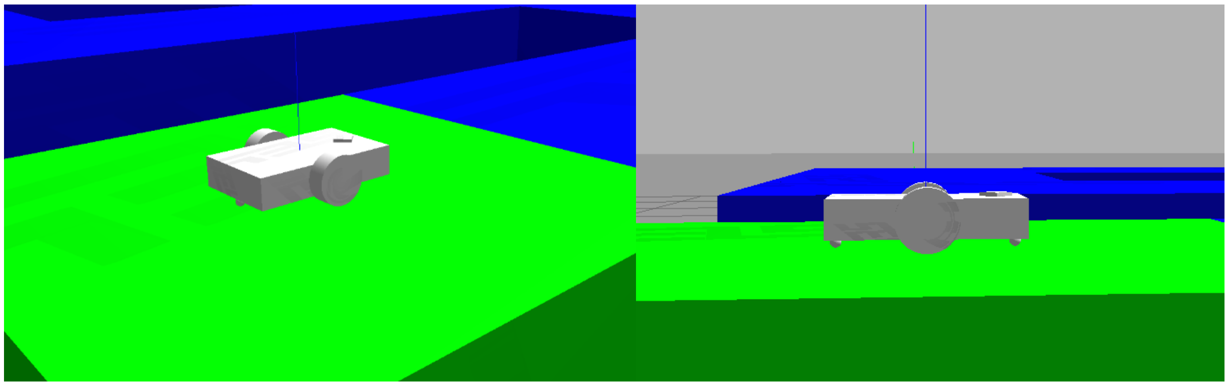


Figure 3: Views of the chosen robot design with centrally located drive wheels and two castor wheels for added support.

In the chosen robot design, a suitable model of a castor when was found and added to the front and back of the robot to provide additional stability. The centrally located drive wheels allowed the centre of the robot to be used as the control point in navigating the robot from point to point along the blue path. An additional diamond was positioned on the front of the robot to aid with the identifying the robot’s orientation during navigation.

This design easily met the project objectives and suitably navigated the blue path to reach the final position in a well-controlled manner.

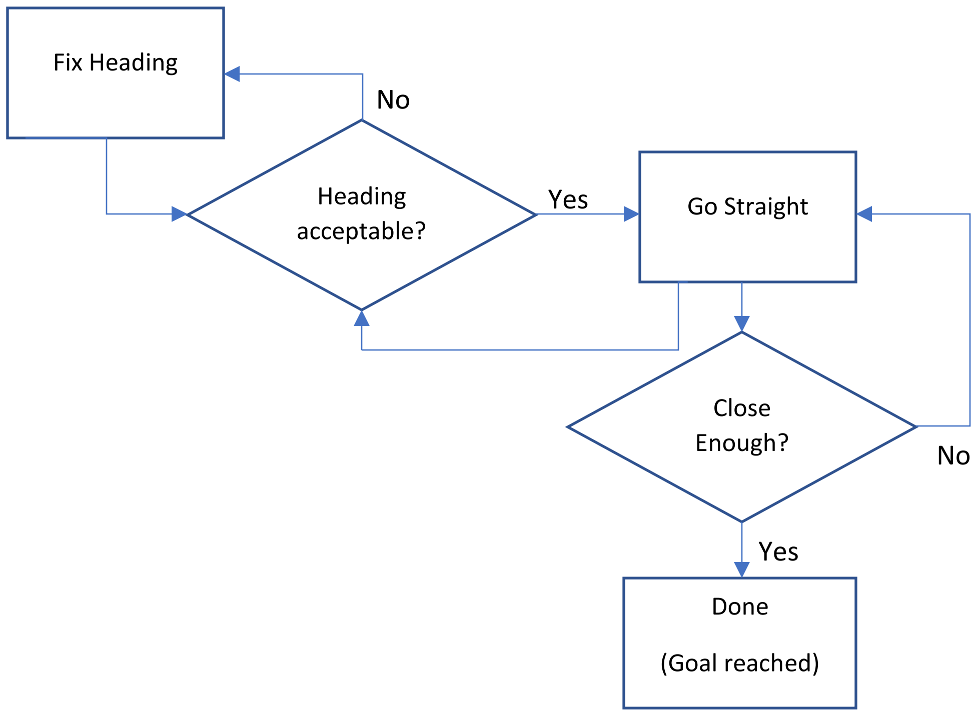
Control philosophy:

The robot will be controlled using 4 states

1) State 0: Turning left or right by 90 degrees
2) State 1: Fixing the heading direction while traversing from point to point on the path (Fix Heading).
3) State 2: Driving straight forward (Go straight).
4) State 3: Command goal reached (Done).

The turning command will be applied by turning the robot clockwise for a “right” or counter clockwise for a “left” command. The new target heading will be based on turning the robot 90 degrees in the relevant direction, based on the current target heading. So if the robot is facing in the direction 0 (from the odometry readings), then a “left” command will turn the robot to face pi/2 degrees (radians) or -pi/2 degrees (radians) for a “right” command.

To drive the robot from the current point to the next required point on the desired path; the target point is set based on the current robot location and the current target direction that the robot is heading. Therefore a “fwd” command with an indicated distance (in metres) will set the next target point to be in the required (positive or negative) x or y direction. This strategy is adequate because the blue path runs parallel to either the x or y axis of the Gazebo world layout. The point to point control strategy is displayed out in the flow chart below:



Note that the speed of the robot will be controlled based on the distance to the target, with the speed reducing as the robot gets closer to the target position.

List of open-source libraries and plugins (scripts?):

1) pysdf to convert sdf to urdf files - https://github.com/andreasBihlmaier/pysdf.git
   This package was used initially to design the first robot model.
2) gazebo plugins in the robot model urdf file:
   - The transmission elements for the right and left wheels
   - The "differential_drive_controller" to control the linear and angular motions of the robot model.
3) check_urdf (script to check that the urdf parses)

To run the project: (using ROS Melodic installed with Gazebo 9.0)

1) Ensure that all the nodes have the required executable permission using the "chmod +x" command.
2) Run the launch file "roslaunch seyo_assignment seyo_assignment.launch" to start all the nodes and to position the robot model at the starting position within seyo.world.
3) In a new terminal, the following commands can be issued in order to:
   - Drive the robot model forward by a specified distance in metres: "rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 1.0" (this will drive the robot forward by 1 metre)
   - Turn the robot model 90 degrees left: "rostopic pub -1 /userCommands seyo_assignment/user_commands left 0" (the number is needed to satisfy the argument inputs but the value is ignored in the code)
   - Turn the robot model 90 degrees right: "rostopic pub -1 /userCommands seyo_assignment/user_commands right 0" (the number is needed to satisfy the argument inputs but the value is ignored in the code)

Note that the first terminal, that was used to run the launch file, will indicate when the next required control command can be issued in the second terminal.

>> To control the robot model from start to finish; the required commands will need to be issued in the following order: (each command is issued for 3 seconds)

1. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 3.0
2. rostopic pub -1 /userCommands seyo_assignment/user_commands left 0
3. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 2.0
4. rostopic pub -1 /userCommands seyo_assignment/user_commands left 0
5. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 3.0
6. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
7. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 2.0
8. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
9. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 5.0
10. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
11. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 4.0
12. rostopic pub -1 /userCommands seyo_assignment/user_commands left 0

13. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 8.0
14. rostopic pub -1 /userCommands seyo_assignment/user_commands left 0
15. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 4.0
16. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
17. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 4.0
18. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
19. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 4.0
20. rostopic pub -1 /userCommands seyo_assignment/user_commands right 0
21. rostopic pub -1 /userCommands seyo_assignment/user_commands fwd 2.5