NAVIGATION AND PATH PLANNING USING TURTLEBOT3

REPORT

Submitted By

CB.EN.U4AIE20019 GORANTLA V N S L VISHNU VARDHAN

CB.EN.U4AIE20039 MENTA SAI AASHISH

……………………

For the Completion of

21AIE213 ROBOTIC OPERATING SYSTEMS&ROBOT SIMULATION

CSE - AI

11th July 2022

ACKNOWLEDGEMENT

We would like to thank all those who have helped us in completing this project of "**NAVIGATION AND PATH PLANNING USING TURTLEBOT3**" under the subject "**21AIE213 ROBOTIC OPERATING SYSTEMS&ROBOT SIMULATION**".

We would like to show our sincere gratitude to our professor SAJITH VARIYAR without whom the project would not have initiated, who taught us the basics to start and visualise the project and enlightened us with the ideas regarding the project, and helped us by clarifying all the doubts whenever being asked.

We would like to thank ourselves. We helped each other and taught each other about various concepts regarding the project which helped in increasing our inner knowledge.

OBJECTIVE:

To design and implement the navigation and path planning with turtlebot3 waffle robot.

TOOLS:

- Gazebo simulator
- RVIZ

INTRODUCTION:

TURTLEBOT

TurtleBot is a low-cost, personal robot kit with open-source software. TurtleBot was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. With TurtleBot, you'll be able to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications.

TURTLEBOT 3

Turtlebot 3, announced and developed in collaboration with ROBOTIS and Open Source Robotics Foundation, is the smallest and cheapest of its generation.It has outstanding structural expansion capability due to ROBOTIS' renowned modular structure with the DYNAMIXEL. Turtlebot 3 will become available in 2 kits, the Turtlebot3 Burger and Turlebot3 Waffle.

The Burger will come with 2pcs of the DYNAMIXEL XL-430-W350-T servo, a Raspberry Pi board, laser distance sensor, microSD card, Lithium polymer battery, quickstart and parts.

The Waffle will come with 2pcs of the DYNAMIXEL XM430-W210-T servo, Intel Joule board, Intel RealSense and laser distance sensors, Lithium polymer battery, quickstart and parts.

In this project we will use turtlebot3 waffle model for navigation and path planning.

CONCEPTS REQURIED FOR MOVEMENT OF ROBOT

FRAMES

Frames in a robot define a coordinate system that the robot uses to know where it is and where to go. A frame is comprised of six main components: an X, Y, & Z axis and a rotation about each of these axes. There are generally three types of frames used on a robot: base (world) frame, user frame, and tool frame

Map frame has its origin at some arbitrarily chosen point in the world. This coordinate frame is fixed in the world.
Now we will see information about these map frame by typing the command “**rostopic echo amcl_pose”** in terminal.
“Amcl_pose “is responsible topic for finding poses of map and other information related to map frame

```
vishnu@vishnu-0308:~$ rostopic echo amcl_pose
header:
  seq: 33
  stamp:
    secs: 314
    nsecs: 387000000
  frame_id: "map"
pose:
  pose:
    position:
      x: 3.788857581415532
      y: 1.0188280395581324
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.06812447539075775
      w: 0.9976768293654684
  covariance: [0.00379468214151224, 0.0009290744650187754, 0.0, 0.0, 0.0, 0.0, 0.0009290744650183314,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0007911645910745471]
```

Odom frame has its origin at the point where the robot is initialized. This coordinate frame is fixed in the world.
Now we will see information about these odom frame by typing the command “**rostopic echo odom”** in terminal.
“odom “is responsible topic for finding poses of map and other information related to odom frame

```
vishnu@vishnu-0308:~$ rostopic echo odom
header:
  seq: 69611
  stamp:
    secs: 2320
    nsecs: 400000000
  frame_id: "odom"
child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: 3.884821407906678
      y: 1.12625461127769
      z: -0.0010082921000191362
    orientation:
      x: 4.358289259646241e-05
      y: 0.0015918402304871688
      z: -0.02824644856158618
      w: 0.9995997210328093
  covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10000
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.001]
twist:
  twist:
    linear:
      x: 8.194730208902955e-07
      y: 2.216674895778126e-06
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 1.3197760725694144e-05
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
```

There are many other frames which are related robots these 2 frames map and odom frames are the main frames.

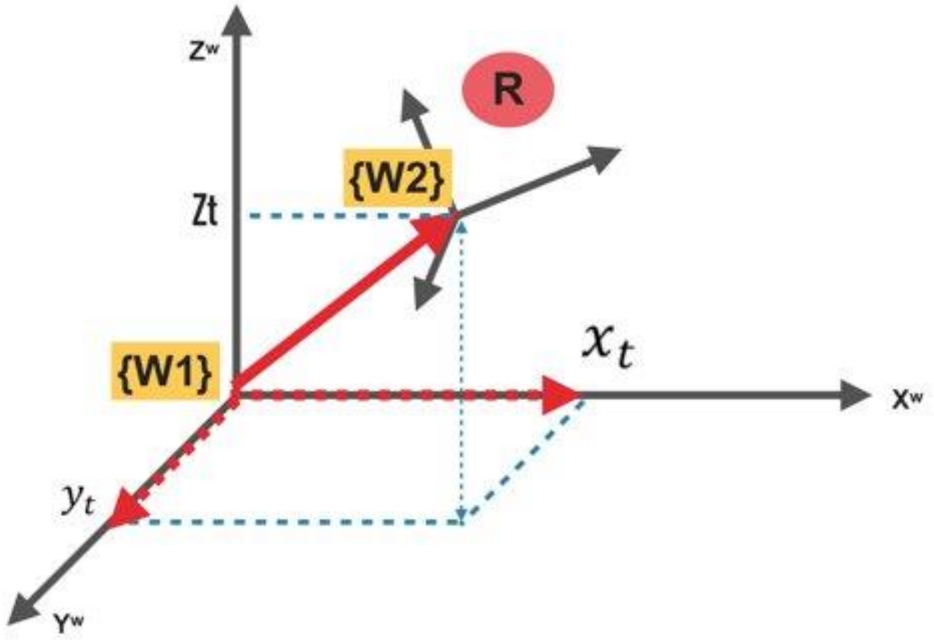
<u>**TRANSFORMATION**</u>

Transformation is simply the change of position and orientation of a frame attached to a body with respect to a frame attached to another body. Transformations in a planar space is known as 2D transformation and transformations in a spatial world is known as 3D transformation

Translation: Change in position

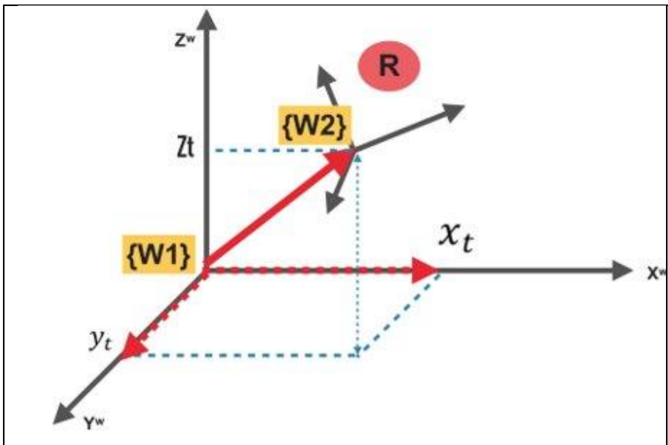
Rotation: Change in orientation

Transformation: Translation + Rotation



In the above diagram we can see that there are 2 coordinate frames w1 and w2 the w1 is the initial position and w2 is the position after transformation(changing position and orientation).

GENERAL ROTATION MATRIX IN 3D



$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_Y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0- \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_Z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

GENERAL TRANSFORMATION MATRIX IN 3D

$$\begin{bmatrix} {}^{W1}x \\ {}^{W1}y \\ {}^{W1}z \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 & x_t \\ r21 & r22 & r23 & y \\ r31 & r32 & r33 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} {}^{W2}x \\ {}^{W2}y \\ {}^{W2}z \\ 1 \end{bmatrix}$$

This is the general transformation matrix in 3d this matrix consist of both rotation matrix and translation vector.

$$\begin{bmatrix} {}^{W1}x \\ {}^{W1}y \\ {}^{W1}z \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{W1}R_{W2} & t \\ 0_{1x3} & 1 \end{bmatrix} * \begin{bmatrix} {}^{W2}x \\ {}^{W2}y \\ {}^{W2}z \\ 1 \end{bmatrix}$$

This is another way of representing the transformation matrix.

QUATERNION

Quaternion are another way of representing rotation

It is written as a scalar and a vector

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

The rotation matrix corresponding to a clockwise/left handed rotation by the unit quaternion axis

QUATERNION TO EULER CONVERSION

The purpose for conversion of quaternion to euler is used to represent in human readble form.

```python
import math

def euler_from_quaternion(x, y, z, w):

        t0 = +2.0 * (w * x + y * z)
        t1 = +1.0 - 2.0 * (x * x + y * y)
        roll_x = math.atan2(t0, t1)

        t2 = +2.0 * (w * y - z * x)
        t2 = +1.0 if t2 > +1.0 else t2
        t2 = -1.0 if t2 < -1.0 else t2
        pitch_y = math.asin(t2)

        t3 = +2.0 * (w * z + x * y)
        t4 = +1.0 - 2.0 * (y * y + z * z)
        yaw_z = math.atan2(t3, t4)

        return roll_x, pitch_y, yaw_z # in radians
```

EULER TO QUATERNION CONVERSION

We can convert the euler to quaternion form also

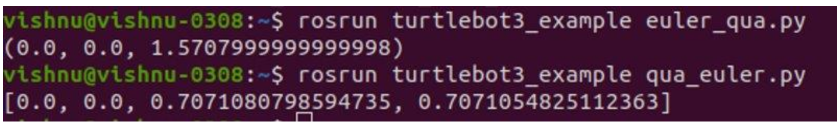
```python
import numpy as np # Scientific computing library for Python

def get_quaternion_from_euler(roll, pitch, yaw):

  qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin(pitch/2)
* np.sin(yaw/2)
  qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos(pitch/2)
* np.sin(yaw/2)
  qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin(pitch/2)
* np.cos(yaw/2)
  qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin(pitch/2)
* np.sin(yaw/2)

  return [qx, qy, qz, qw]
```

RESULT



TRANSFORMATION PACKAGE

TF stands for transformation library in ROS

It performs computation for transformations between frames.

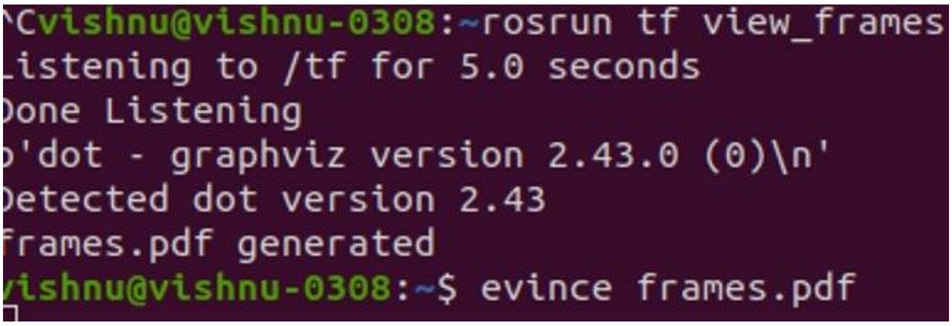
It allows to find the pose of any object in any frame using transformations

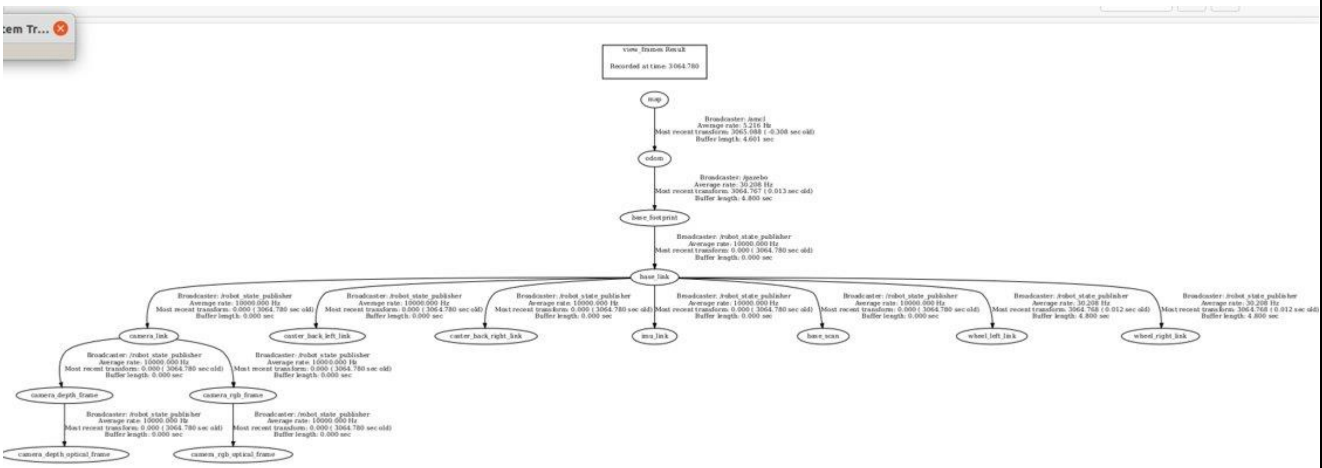
A robot is a collection of frames attached to its different joints

TF Package Nodes

The TF Package has several ROS nodes that provide utilities to manipulate frames and transformations in ROS

view_frames: visualizes the full tree of coordinate transforms.





The above picture shows all topics and nodes .

tf_monitor: monitors transforms between frames.

```
^Cvishnu@vishnu-0308:~rosrun tf tf_monitor



RESULTS: for all Frames

Frames:
Frame: base_footprint published by unknown_publisher Average Delay: 0.001 Max Delay: 0.001
Frame: base_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: base_scan published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: camera_depth_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: camera_depth_optical_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: camera_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: camera_rgb_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: camera_rgb_optical_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: caster_back_left_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: caster_back_right_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: imu_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: wheel_left_link published by unknown_publisher Average Delay: 0.00425 Max Delay: 0.005
Frame: wheel_right_link published by unknown_publisher Average Delay: 0.00425 Max Delay: 0.005

All Broadcasters:
Node: unknown_publisher 77.6699 Hz, Average Delay: 0.002625 Max Delay: 0.005
Node: unknown_publisher(static) 1e+08 Hz, Average Delay: 0 Max Delay: 0



RESULTS: for all Frames
```

tf_echo: prints specified transform to screen

```
vishnu@vishnu-0308:~$ rosrun tf tf_echo odom base_footprint
At time 3670.267
- Translation: [3.885, 1.126, -0.001]
- Rotation: in Quaternion [0.000, 0.002, -0.028, 1.000]
            in RPY (radian) [-0.000, 0.003, -0.055]
            in RPY (degree) [-0.000, 0.182, -3.175]
At time 3670.267
- Translation: [3.885, 1.126, -0.001]
- Rotation: in Quaternion [0.000, 0.002, -0.028, 1.000]
            in RPY (radian) [-0.000, 0.003, -0.055]
            in RPY (degree) [-0.000, 0.182, -3.175]
At time 3671.267
- Translation: [3.885, 1.126, -0.001]
- Rotation: in Quaternion [0.000, 0.002, -0.028, 1.000]
            in RPY (radian) [-0.000, 0.003, -0.055]
            in RPY (degree) [-0.000, 0.182, -3.175]
At time 3672.267
- Translation: [3.885, 1.126, -0.001]
- Rotation: in Quaternion [0.000, 0.002, -0.028, 1.000]
            in RPY (radian) [-0.000, 0.003, -0.055]
            in RPY (degree) [-0.000, 0.182, -3.175]
```

We are finding relation between two frames at different time

roswtf: with the tfwtf plugin, helps you track down problems with tf.

static_transform_publisher is a command line tool for sending static transforms

NAVIGATION

There are two types of navigation used in ROS

1)Map-based Navigation

2)Reactive Navigation

In our project we performed map-based navigation

Map based navigation

Localization: it helps the robot to know where he is

Mapping: the robot needs to have a map of its environment to be able to recognize where he has been moving around so far

Motion planning or path planning: to plan a path, the target position must be well-defined to the robot, which require an appropriate addressing scheme that the robot can understand

Navigation packages

Three main packages of the navigation stack

move_base: makes the robot navigate in a map and move to move to a goal pose with respect to a given reference frame

mapping: creates maps using laser scan data

amcl: responsible for localization using an existing map.

SLAM

BUILDING A MAP: SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

It is the process of building a map using range sensors (e.g. laser sensors, 3D sensors, ultrasonic sensors) while the robot is moving around and exploring an unknown area

Sensor Fusion: This process uses filtering techniques like Kalman filters or praticle filters

SLAM APPROACHES

There are several SLAM approaches in ROS

gmapping which contains a ROS wrapper for OpenSlam's Gmapping

cartographer, which is a system developed by Google that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations.

hector_slam which is an other SLAM approach that can be used without odomtery

OCCUPANCY IN GRID MAP

A map is a grid (matrix) of cell

A cell can be empty or occupied

Depending on resolution, cell size can be 5 to 50 cm

Each cell hold a probability of occupancy 0% to 100%

Areas that are unknown are marked as -1

MARKING AND CLEARING

The map is built using a SLAM algorithm.

Cell has three possible states

Unknown

Empty

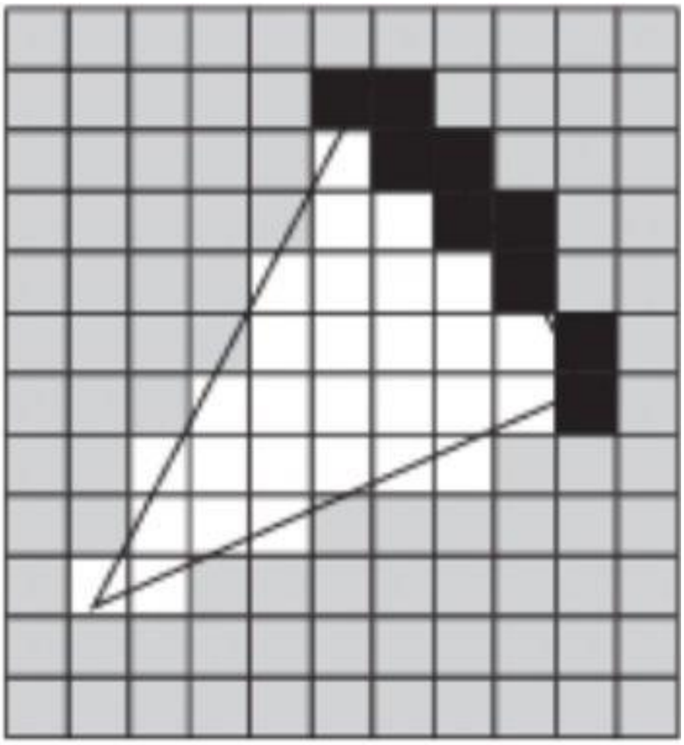
Occupied

It is based on the process of marking and clearing

Marking: a cell is marked as obstacle

Clearing: a cell is marked as empty

ray-tracing: used to find empty cell.



In the above diagram we can see that black box represents there is an obstacle. The white colour represents there is no obstacle and the grey colour represents the path is not covered.

USAGE OF ACTION SERVICES

For every movement of the robot the feedback will be published

Goal will be our target location.

Result will be either it is completed or not.

NAVIGATION STACK

The navigation stack has two motion planners:

Global Path Planner: plans a static obstacle-free path from the location of the robot to the goal location

Local Path Planner: execute the planned trajectory and avoids dynamic obstacle.

GLOBAL PATH PLANNER

The global path planner is responsible for finding a global obstacle-free path from initial location to the goal location using the environment map

Global path planner must adhere to the nav_core::BaseGlobalPlanner interface.

BUILT-IN GLOBAL PATH PLANNER IN ROS

There are three built-in global path planners in ROS:

carrot planner: simple global planner that takes a user specified goal point and attempts to move the robot as close to it as possible, even when that goal point is in an obstacle.

navfn: uses the Dijkstra's algorithm to find the global path between any two locations.

global_planner: is a replacement of navfn and is more flexible and has more options.

In our project we are using navfn package for global path planner

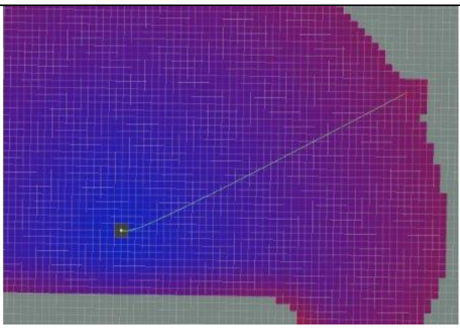
NAVFN

navfn uses Dijkstra's algorithm to find a global path

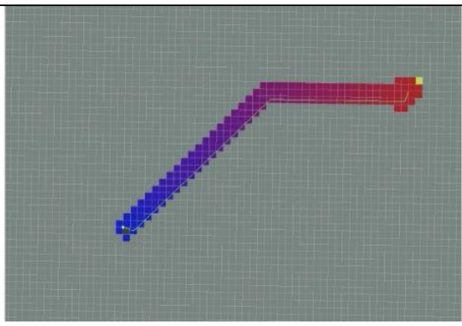
global_planner is a flexible replacement of navfn

Support of A*

Can use a grid path



DIJKSTRA



A*

LOCAL PATH PLANNER

The local path planner is responsible execution the static path determined by the global path planner while avoiding dynamic obstacle that might come into the path using the robot's sensors.

‣ Global path planner must adhere to the nav_core::BaseLocalPlanner interface.

The algorithm used for Local path planner is DWA Algorithm

DWA ALGORITHM

- Discretely sample in the robot's control space (dx,dy,dtheta)
- For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
- Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
- Pick the highest-scoring trajectory and send the associated velocity to the mobile base Rinse and repeat.

DWA PARAMETERS

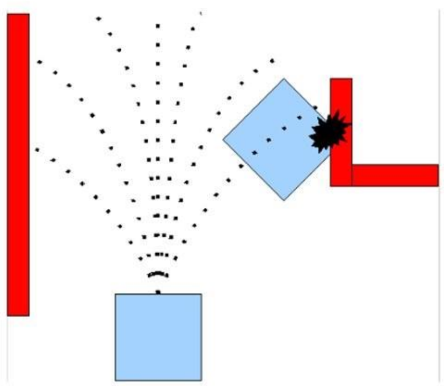
simulation time: time allowed for the robot to move with the sampled velocities

takes the velocity samples in robot's control space, and examine the circular trajectories represented by those velocity samples, and finally eliminate bad velocities

High simulation times (>=5) lead to heavier computation, but get longer paths

Low simulation times (<=2)

limited performance, especially when the robot needs to pass a narrow doorway, or gap between furnitures, because there is insufficient time to obtain the optimal trajectory that actually goes through the narrow passway



TRAJECTORY SCORING

cost = path distance bias (distance(m) to path from the endpoint of the trajectory) *

+ goal distance-bias (distance(m) to local goal from the endpoint of the trajectory) *

+ occdist-scale (maximum obstacle cost along the trajectory in obstacle cost (0-254))

TURTLEBOT 3 IN EMPTY WORLD

UNIVERSAL ROBOT DESCRIPTION FORMAT (URDF) FOR TURTLEBOT WAFFLE MODEL

```
<?xml version="1.0" ?>
<robot name="turtlebot3_waffle" xmlns:xacro="http://ros.org/wiki/xacro">
<xacro:include filename="$(find turtlebot3_description)/urdf/common_properties.xacro"/>
<xacro:include filename="$(find
turtlebot3_description)/urdf/turtlebot3_waffle.gazebo.xacro"/>

<xacro:property name="r200_cam_rgb_px" value="0.005"/>
<xacro:property name="r200_cam_rgb_py" value="0.018"/>
<xacro:property name="r200_cam_rgb_pz" value="0.013"/>
<xacro:property name="r200_cam_depth_offset" value="0.01"/>

<link name="base_footprint"/>

<joint name="base_joint" type="fixed">
<parent link="base_footprint"/>
<child link="base_link" />
<origin xyz="0 0 0.010" rpy="0 0 0"/>
</joint>

<link name="base_link">
<visual>
<origin xyz="-0.064 0 0.0" rpy="0 0 0"/>
```

```
<geometry>
<mesh filename="package://turtlebot3_description/meshes/bases/waffle_base.stl"
scale="0.001 0.001 0.001"/>
</geometry>
<material name="light_black"/>
</visual>

<collision>
<origin xyz="-0.064 0 0.047" rpy="0 0 0"/>
<geometry>
<box size="0.266 0.266 0.094"/>
</geometry>
</collision>

<inertial>
<origin xyz="0 0 0" rpy="0 0 0"/>
<mass value="1.3729096e+00"/>
<inertia ixx="8.7002718e-03" ixy="-4.7576583e-05" ixz="1.1160499e-04"
iyy="8.6195418e-03" iyz="-3.5422299e-06"
izz="1.4612727e-02" />
</inertial>
</link>

<joint name="wheel_left_joint" type="continuous">
<parent link="base_link"/>
<child link="wheel_left_link"/>
<origin xyz="0.0 0.144 0.023" rpy="-1.57 0 0"/>
<axis xyz="0 0 1"/>
</joint>

<link name="wheel_left_link">
<visual>
<origin xyz="0 0 0" rpy="1.57 0 0"/>
<geometry>
<mesh filename="package://turtlebot3_description/meshes/wheels/left_tire.stl"
scale="0.001 0.001 0.001"/>
</geometry>
<material name="dark"/>
</visual>

<collision>
<origin xyz="0 0 0" rpy="0 0 0"/>
<geometry>
<cylinder length="0.018" radius="0.033"/>
</geometry>
</collision>
```

```
<inertial>
<origin xyz="0 0 0" />
<mass value="2.8498940e-02" />
<inertia ixx="1.1175580e-05" ixy="-4.2369783e-11" ixz="-5.9381719e-09"
iyy="1.1192413e-05" iyz="-1.4400107e-11"
izz="2.0712558e-05" />
</inertial>
</link>

<joint name="wheel_right_joint" type="continuous">
<parent link="base_link"/>
<child link="wheel_right_link"/>
<origin xyz="0.0 -0.144 0.023" rpy="-1.57 0 0"/>
<axis xyz="0 0 1"/>
</joint>

<link name="wheel_right_link">
<visual>
<origin xyz="0 0 0" rpy="1.57 0 0"/>
<geometry>
<mesh filename="package://turtlebot3_description/meshes/wheels/right_tire.stl"
scale="0.001 0.001 0.001"/>
</geometry>
<material name="dark"/>
</visual>

<collision>
<origin xyz="0 0 0" rpy="0 0 0"/>
<geometry>
<cylinder length="0.018" radius="0.033"/>
</geometry>
</collision>

<inertial>
<origin xyz="0 0 0" />
<mass value="2.8498940e-02" />
<inertia ixx="1.1175580e-05" ixy="-4.2369783e-11" ixz="-5.9381719e-09"
iyy="1.1192413e-05" iyz="-1.4400107e-11"
izz="2.0712558e-05" />
</inertial>
</link>

<joint name="caster_back_right_joint" type="fixed">
<parent link="base_link"/>
<child link="caster_back_right_link"/>
<origin xyz="-0.177 -0.064 -0.004" rpy="-1.57 0 0"/>
</joint>
```

```xml
<link name="caster_back_right_link">
<collision>
<origin xyz="0 0.001 0" rpy="0 0 0"/>
<geometry>
<box size="0.030 0.009 0.020"/>
</geometry>
</collision>

<inertial>
<origin xyz="0 0 0" />
<mass value="0.005" />
<inertia ixx="0.001" ixy="0.0" ixz="0.0"
iyy="0.001" iyz="0.0"
izz="0.001" />
</inertial>
</link>

<joint name="caster_back_left_joint" type="fixed">
<parent link="base_link"/>
<child link="caster_back_left_link"/>
<origin xyz="-0.177 0.064 -0.004" rpy="-1.57 0 0"/>
</joint>

<link name="caster_back_left_link">
<collision>
<origin xyz="0 0.001 0" rpy="0 0 0"/>
<geometry>
<box size="0.030 0.009 0.020"/>
</geometry>
</collision>

<inertial>
<origin xyz="0 0 0" />
<mass value="0.005" />
<inertia ixx="0.001" ixy="0.0" ixz="0.0"
iyy="0.001" iyz="0.0"
izz="0.001" />
</inertial>
</link>

<joint name="imu_joint" type="fixed">
<parent link="base_link"/>
<child link="imu_link"/>
<origin xyz="0.0 0 0.068" rpy="0 0 0"/>
</joint>
```

```
<link name="imu_link"/>

<joint name="scan_joint" type="fixed">
<parent link="base_link"/>
<child link="base_scan"/>
<origin xyz="-0.064 0 0.122" rpy="0 0 0"/>
</joint>

<link name="base_scan">
<visual>
<origin xyz="0 0 0" rpy="0 0 0"/>
<geometry>
<mesh filename="package://turtlebot3_description/meshes/sensors/lds.stl" scale="0.001
0.001 0.001"/>
</geometry>
<material name="dark"/>
</visual>

<collision>
<origin xyz="0.015 0 -0.0065" rpy="0 0 0"/>
<geometry>
<cylinder length="0.0315" radius="0.055"/>
</geometry>
</collision>

<inertial>
<mass value="0.114" />
<origin xyz="0 0 0" />
<inertia ixx="0.001" ixy="0.0" ixz="0.0"
iyy="0.001" iyz="0.0"
izz="0.001" />
</inertial>
</link>

<joint name="camera_joint" type="fixed">
<origin xyz="0.064 -0.065 0.094" rpy="0 0 0"/>
<parent link="base_link"/>
<child link="camera_link"/>
</joint>

<link name="camera_link">
<visual>
<origin xyz="0 0 0" rpy="1.57 0 1.57"/>
<geometry>
<mesh filename="package://turtlebot3_description/meshes/sensors/r200.dae" />
</geometry>
</visual>
```

```
<collision>
<origin xyz="0.003 0.065 0.007" rpy="0 0 0"/>
<geometry>
<box size="0.012 0.132 0.020"/>
</geometry>
</collision>

<!-- This inertial field needs doesn't contain reliable data!! -->
<!-- <inertial>
<mass value="0.564" />
<origin xyz="0 0 0" />
<inertia ixx="0.003881243" ixy="0.0" ixz="0.0"
iyy="0.000498940" iyz="0.0"
izz="0.003879257" />
</inertial>-->
</link>

<joint name="camera_rgb_joint" type="fixed">
<origin xyz="${r200_cam_rgb_px} ${r200_cam_rgb_py} ${r200_cam_rgb_pz}" rpy="0 0 0"/>
<parent link="camera_link"/>
<child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame"/>

<joint name="camera_rgb_optical_joint" type="fixed">
<origin xyz="0 0 0" rpy="-1.57 0 -1.57"/>
<parent link="camera_rgb_frame"/>
<child link="camera_rgb_optical_frame"/>
</joint>
<link name="camera_rgb_optical_frame"/>

<joint name="camera_depth_joint" type="fixed">
<origin xyz="${r200_cam_rgb_px} ${r200_cam_rgb_py + r200_cam_depth_offset}
${r200_cam_rgb_pz}" rpy="0 0 0"/>
<parent link="camera_link"/>
<child link="camera_depth_frame"/>
</joint>
<link name="camera_depth_frame"/>

<joint name="camera_depth_optical_joint" type="fixed">
<origin xyz="0 0 0" rpy="-1.57 0 -1.57"/>
<parent link="camera_depth_frame"/>
<child link="camera_depth_optical_frame"/>
</joint>
<link name="camera_depth_optical_frame"/>
 </robot>
```

IMPLEMENTATION

MOVING BOT TO A DESIGNATED GOAL

PYTHON CODE

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist, Point, Quaternion
import tf
from math import radians, copysign, sqrt, pow, pi, atan2
from tf.transformations import euler_from_quaternion
import numpy as np

msg = """
control your Turtlebot3!
-----------------------
Insert xyz - coordinate.
x : position x (m)
y : position y (m)
z : orientation z (degree: -180 ~ 180)
If you want to close, insert 's'
-----------------------
"""


class GotoPoint():
    def __init__(self):
        rospy.init_node('turtlebot3_pointop_key', anonymous=False)
        rospy.on_shutdown(self.shutdown)
        self.cmd_vel = rospy.Publisher('cmd_vel', Twist, queue_size=5)
        position = Point()
        move_cmd = Twist()
        r = rospy.Rate(10)
        self.tf_listener = tf.TransformListener()
        self.odom_frame = 'odom'

        try:
            self.tf_listener.waitForTransform(self.odom_frame, 'base_footprint', rospy.Time(),
rospy.Duration(1.0))
            self.base_frame = 'base_footprint'
        except (tf.Exception, tf.ConnectivityException, tf.LookupException):
            try:
                self.tf_listener.waitForTransform(self.odom_frame, 'base_link', rospy.Time(),
rospy.Duration(1.0))
                self.base_frame = 'base_link'
            except (tf.Exception, tf.ConnectivityException, tf.LookupException):
                rospy.loginfo("Cannot find transform between odom and base_link or
base_footprint")
                rospy.signal_shutdown("tf Exception")
```

```
(position, rotation) = self.get_odom()
last_rotation = 0
linear_speed = 1
angular_speed = 1
(goal_x, goal_y, goal_z) = self.getkey()
if goal_z > 180 or goal_z < -180:
    print("you input wrong z range.")
    self.shutdown()
goal_z = np.deg2rad(goal_z)
goal_distance = sqrt(pow(goal_x - position.x, 2) + pow(goal_y - position.y, 2))
distance = goal_distance

while distance > 0.05:
    (position, rotation) = self.get_odom()
    x_start = position.x
    y_start = position.y
    path_angle = atan2(goal_y - y_start, goal_x- x_start)

    if path_angle < -pi/4 or path_angle > pi/4:
        if goal_y < 0 and y_start < goal_y:
            path_angle = -2*pi + path_angle
        elif goal_y >= 0 and y_start > goal_y:
            path_angle = 2*pi + path_angle
    if last_rotation > pi-0.1 and rotation <= 0:
        rotation = 2*pi + rotation
    elif last_rotation < -pi+0.1 and rotation > 0:
        rotation = -2*pi + rotation
    move_cmd.angular.z = angular_speed * path_angle-rotation

    distance = sqrt(pow((goal_x - x_start), 2) + pow((goal_y - y_start), 2))
    move_cmd.linear.x = min(linear_speed * distance, 0.1)

    if move_cmd.angular.z > 0:
        move_cmd.angular.z = min(move_cmd.angular.z, 1.5)
    else:
        move_cmd.angular.z = max(move_cmd.angular.z, -1.5)

    last_rotation = rotation
    self.cmd_vel.publish(move_cmd)
    r.sleep()
(position, rotation) = self.get_odom()

while abs(rotation - goal_z) > 0.05:
    (position, rotation) = self.get_odom()
    if goal_z >= 0:
        if rotation <= goal_z and rotation >= goal_z - pi:
```

```
                move_cmd.linear.x = 0.00
                move_cmd.angular.z = 0.5
            else:
                move_cmd.linear.x = 0.00
                move_cmd.angular.z = -0.5
        else:
            if rotation <= goal_z + pi and rotation > goal_z:
                move_cmd.linear.x = 0.00
                move_cmd.angular.z = -0.5
            else:
                move_cmd.linear.x = 0.00
                move_cmd.angular.z = 0.5
        self.cmd_vel.publish(move_cmd)
        r.sleep()


    rospy.loginfo("Stopping the robot...")
    self.cmd_vel.publish(Twist())

  def getkey(self):
    x, y, z = input("| x | y | z |\n").split()
    if x == 's':
        self.shutdown()
    x, y, z = [float(x), float(y), float(z)]
    return x, y, z

  def get_odom(self):
    try:
        (trans, rot) = self.tf_listener.lookupTransform(self.odom_frame, self.base_frame,
rospy.Time(0))
        rotation = euler_from_quaternion(rot)

    except (tf.Exception, tf.ConnectivityException, tf.LookupException):
        rospy.loginfo("TF Exception")
        return

    return (Point(*trans), rotation[2])

   def shutdown(self):
    self.cmd_vel.publish(Twist())
    rospy.sleep(1)
 if __name__ == '__main__':
  try:
    while not rospy.is_shutdown():
        print(msg)
        GotoPoint()
   except:  rospy.loginfo("shutdown program.")
```

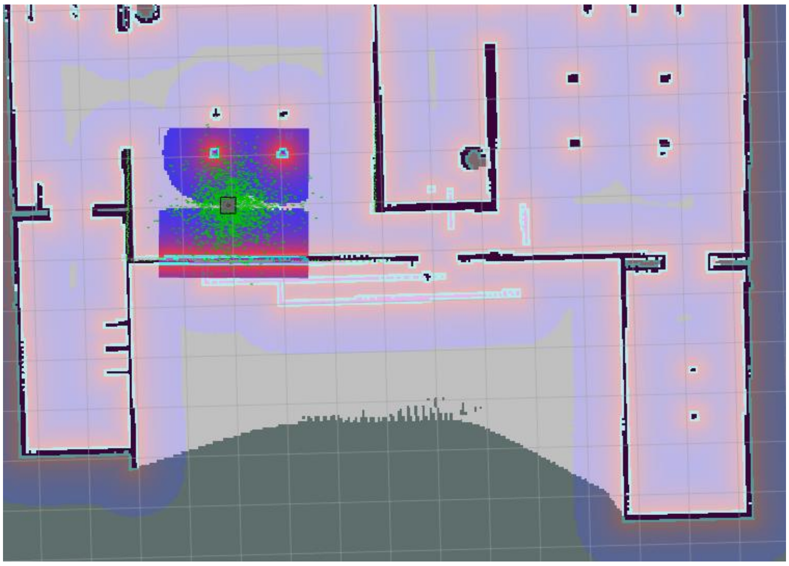
RESULTS

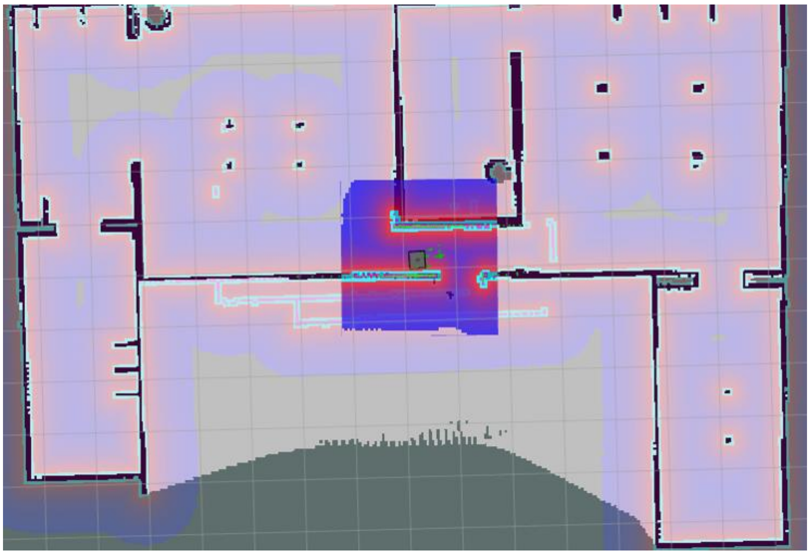
```
vishnu@vishnu-0308:~$ rosrun turtlebot3_example turtlebot3_pointop_key

control your Turtlebot3!
-----------------------
Insert xyz - coordinate.
x : position x (m)
y : position y (m)
z : orientation z (degree: -180 ~ 180)
If you want to close, insert 's'
-----------------------

| x | y | z |
0.4 0.2 0
```

INTIAL LOCATION





FINAL LOCATION

REFERENCES

http://wiki.ros.org/navigation
https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
https://robotacademy.net.au/lesson/describing-rotation-and-translation-in-3d/