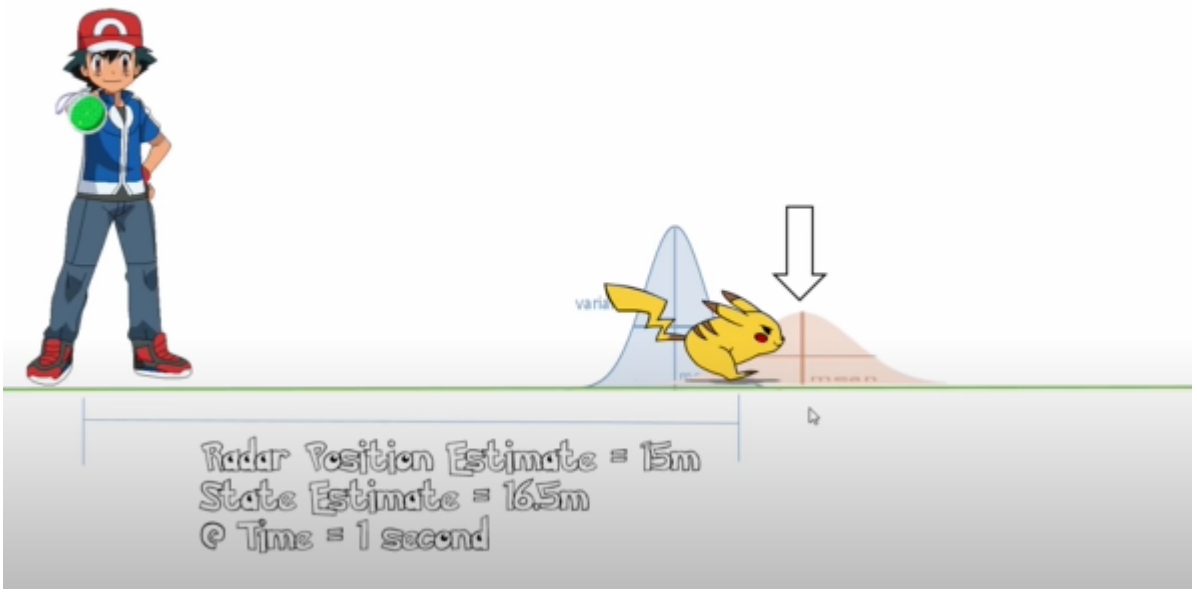
Autonomous Task

STEP 1 : BASIC INFORMATION

Localisation

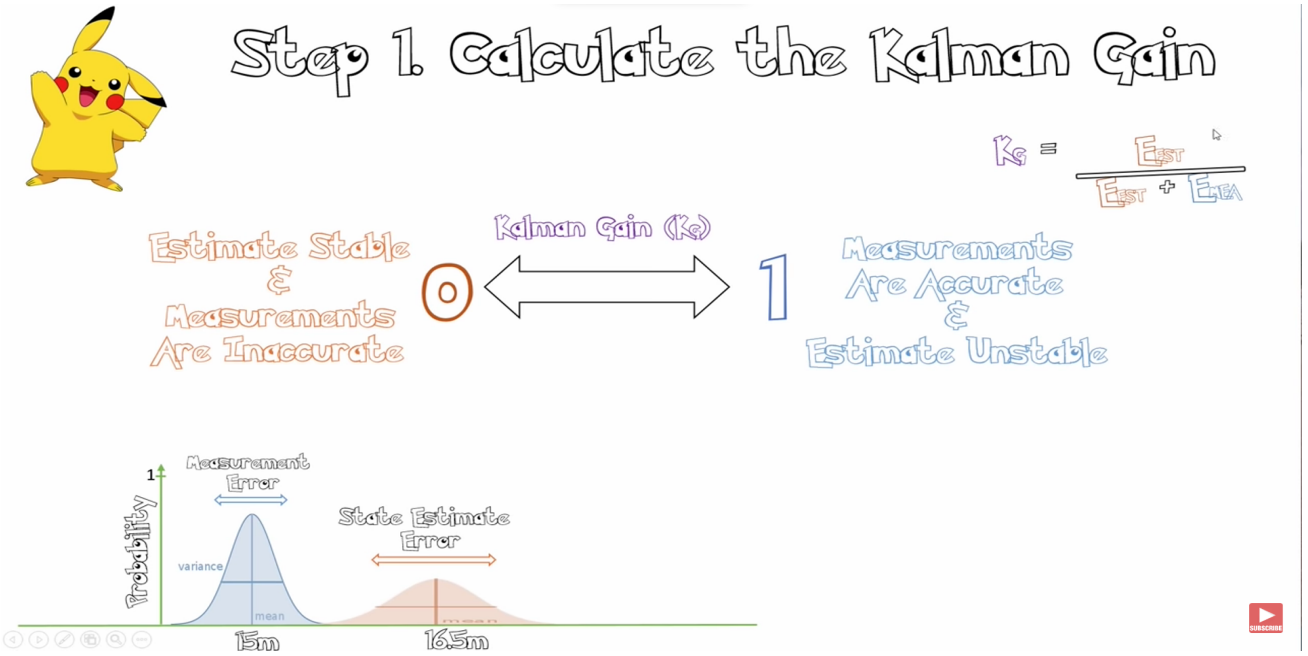
Kalman Filters

To calculate the optimal position of the object at a given instant of time. We have two estimates, the state estimate (calculated using velocity and accln. terms) and the radar estimate (calculated using position). Both have a mean and a variance centred at different values.



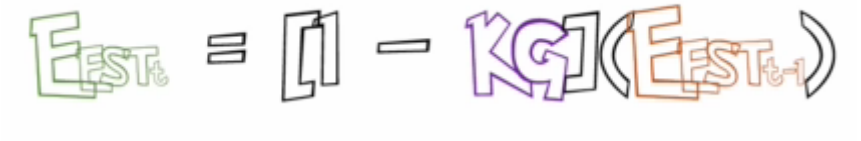
Goal: To combine the probability distribution functions (PDFs) to a single PDF and find the optimal position of the Pikachu.

Step 1: Calculate the Kalman Gain



Step 2: Calculate the current estimate

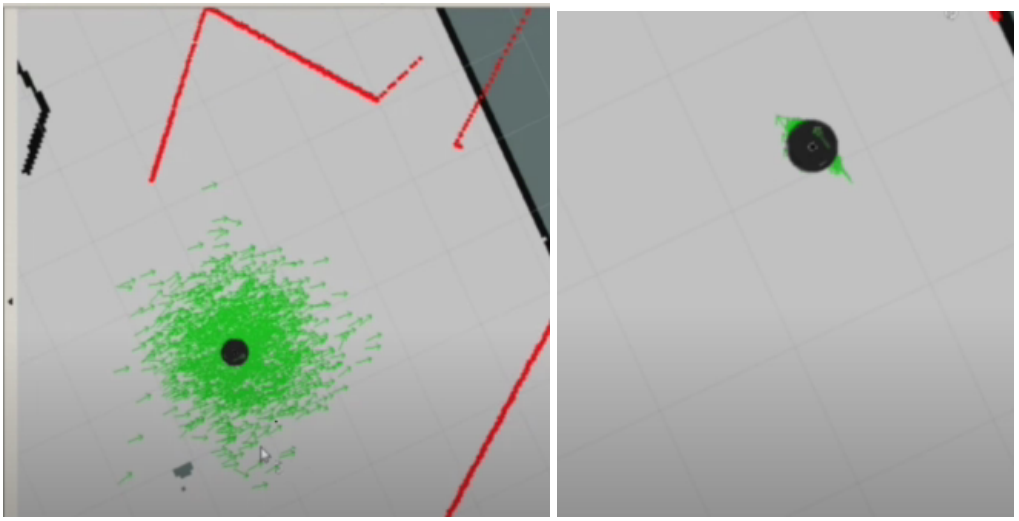
Step 3: Calculate the new state estimate error



Uses : Cruise control. Both available for 2D and 3D applications.

What is ROS Navigation?

What do we need? The first thing is a map, created using mapping. Second, we need to localise the robot on the map. In an unknown city, you want to know where you are, this is called Localisation.



The localisation of the robot before and after initial introspection by movement. (green arrows indicate the localisation region) The more the robot moves in the environment, the more accurate the localization gets.

Thirdly, we can send go locations to the rover for the rover to automatically navigate. (path planning)

Fourthly, we need our rover to dodge objects that were previously not in the map. (obstacle avoidance)

What is the Navigation Stack?

It is the set of ROS nodes and algorithms which are used to move the rover autonomously.

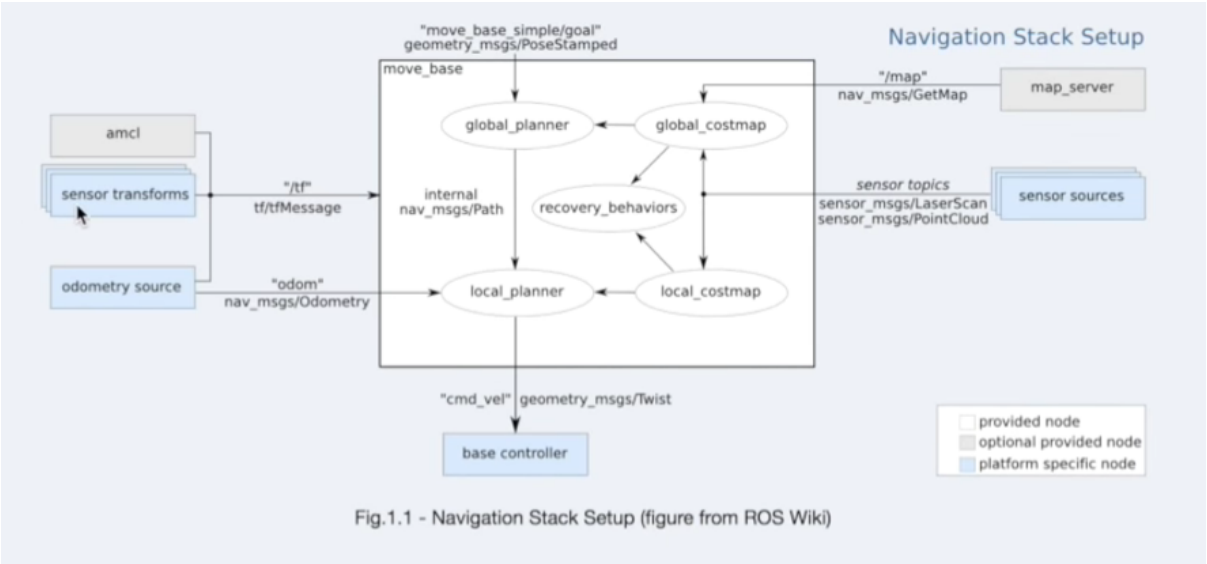


Fig.1.1 - Navigation Stack Setup (figure from ROS Wiki)

Odometry source - data from the wheel encoders
Sensor source - generally, laser data for localisation

Sensor transforms /tf - the data from the sensors should be referenced to a common frame of reference, generally the base link in order to interpret the data and respond accordingly. The robot should publish the relationship between the main robot coordinate frame and the different sensor frames using ROS tf.
Base_controller - convert the output of the Navigation Stack (the Twist command) to move the robot.

Hardware Requirements

1. Planar Laser to map the environment
2. Ability to take in and comprehend Twist type commands.

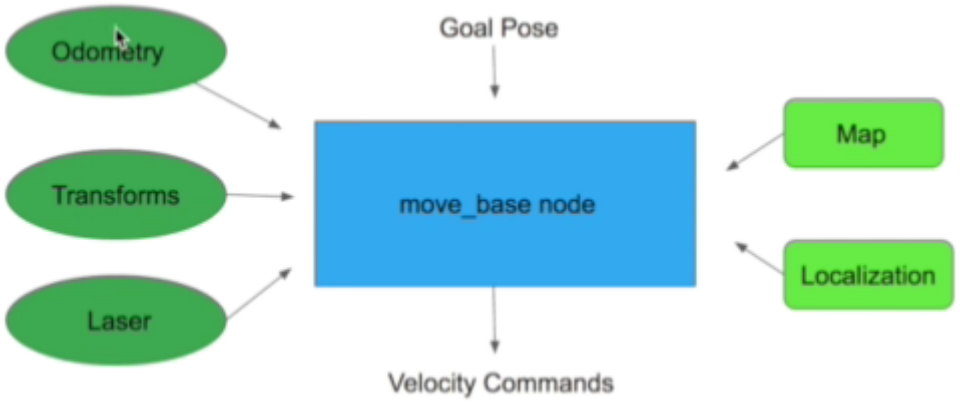
The move_base node

The list of packages which are linked by the move_base node are:

1. Global-planner
2. Local-planner
3. Rotate-recovery
4. Clear-costmap-recovery
5. costmap-2D

Other packages interfaced are:

1. Map-server
2. AMCL (Adaptive Monte Carlo Localization)
3. gmapping



Mapping

1. Laser Scan display
2. Map display

SLAM (Simultaneous Localization and Mapping)

Building a map of an unknown environment while simultaneously keeping track of the robot's location of the map being built.
The gmapping ROS package is an implementation of SLAM algo.

Gmapping

1. The gmapping package contains a node called slam_gmapping, which helps to create a 2D map.
2. Converts the laser signals and transforms from the rover and converts it into an OGM (Occupancy grid map)
3. Slam_gmapping node subscribes to the laser data topic (/kobuki/laser/scan) and to the /tf topics.
4. Generated map is published into the /map topic
5. /map topic uses a message of type nav_msgs/OccupancyGrid. (Occupancy is an integer 0-100, 0 for completely free, 100 for completely occupied and -1 for unknown)

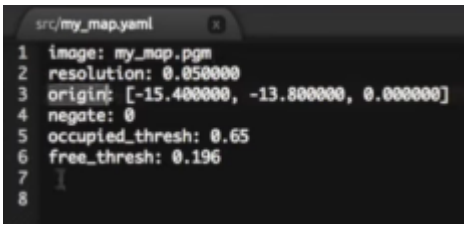
Saving the Map

Command : rosrun map_server map_saver -f <map name>

Map_server : topic
Map_saver : node
-f : is used to specify a file name

Two files get saved:

1. YAML file - Aint markup language file contains details about the image of the map.



2. PGM file - image of the map, a portable grey map.

Providing a Map

- Static_map service can be used to get a map.
- The topics used by this node to provide the map are : map_metadata. Map.
- Command : rosrun map_server map_server map_file.yaml (to launch the map server) (in project directory)
- Rostopic echo /map_metadata provides the metadata of the map loaded with the map_server node
- Rostopic echo /map prints the OGM
- Command : rosservice call /static_map "{}" (Gets the OGM of the map and the metadata) The map created is static and doesn't update with real time

- The map is 2D in nature

Transforms

- We need to set a transform between the robot base and the laser and add it to a transform tree.
- We need to provide two transforms:
    - Laser → base
    - Base → odom
- Command : rosrun tf view_frames
- Static and Dynamic transforms - for example. A sensor on the arm connected to the base link requires a dynamic transform

Transform Tree

A transform tree, also known as a tf tree (short for "transform tree"), is a data structure used in the Robot Operating System (ROS) to represent the relationships between different coordinate frames in a robotic system. It provides a way to keep track of the spatial relationships and transformations between various components, such as robot links, sensors, and world frames.

The transform tree is a directed acyclic graph (DAG) that organizes the coordinate frames and their relationships in a hierarchical manner. Each node in the tree represents a coordinate frame, and the edges between nodes represent the transformations between those frames.

In the transform tree, the root node represents the global or world frame, which serves as a reference coordinate system. Child nodes represent frames that are associated with different components or parts of the robot, such as the base, sensors, or end effectors. The edges between nodes represent the transformations, including translations and rotations, between the frames.

By maintaining the transform tree, the ROS system can provide real-time information about the relative poses of different components in the robot. This information is crucial for tasks such as sensor fusion, robot localization, motion planning, and coordination between different components.

The transform tree is managed by the tf library in ROS, which provides the infrastructure to publish, update, and query transformations between frames. It supports real-time transformation updates and handles complex scenarios where there are multiple frames and transformations involved.

Overall, the transform tree in ROS allows the robot and its components to maintain a consistent understanding of their spatial relationships, enabling coordinated operations and accurate sensor data interpretation in a multi-component robotic system.

How to configure your robot?

```
<joint name="laser_sensor_joint" type="fixed">
   <origin xyz="0.0 0.0 0.435" rpy="0 0 0"/>
   <parent link="base_link"/>
   <child link="laser_sensor_link"/>
</joint>

<link name="laser_sensor_link">
      <inertial>
            <mass value="1e-5"/>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"/>
      </inertial>
      <collision>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                  <box size="0.1 0.1 0.1"/>
            </geometry>
      </collision>
      <visual>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                  <mesh filename="package://hokuyo/meshes/hokuyo.dae"/>
            </geometry>
      </visual>
</link>
```

As you can see, it's defining several things regarding the laser:

- It defines the position and orientation of the laser regarding the base ("base_link") of the robot.
- It defines inertia related values
- It defines collision related values. These values will provide the real physics of the laser.
- It defines visual values. These values will provide the visual elements of the laser. This is just for visualization purposes.

Info of some rostopics used

```
user ~ $ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
 * /gazebo (http://ip-172-31-34-208:48232/)
```

Info /odom:

```
user ~ $ rostopic info /odom
Type: nav_msgs/Odometry

Publishers:
 * /gazebo (http://ip-172-31-34-208:48232/)

Subscribers: None
```

Info /kobuki/l

```
user ~ $ rostopic info /tf
Type: tf2_msgs/TFMessage

Publishers:
 * /robot_state_publisher (http://ip-172-31-34-208:48020/)
 * /gazebo (http://ip-172-31-34-208:48232/)

Subscribers: None
```

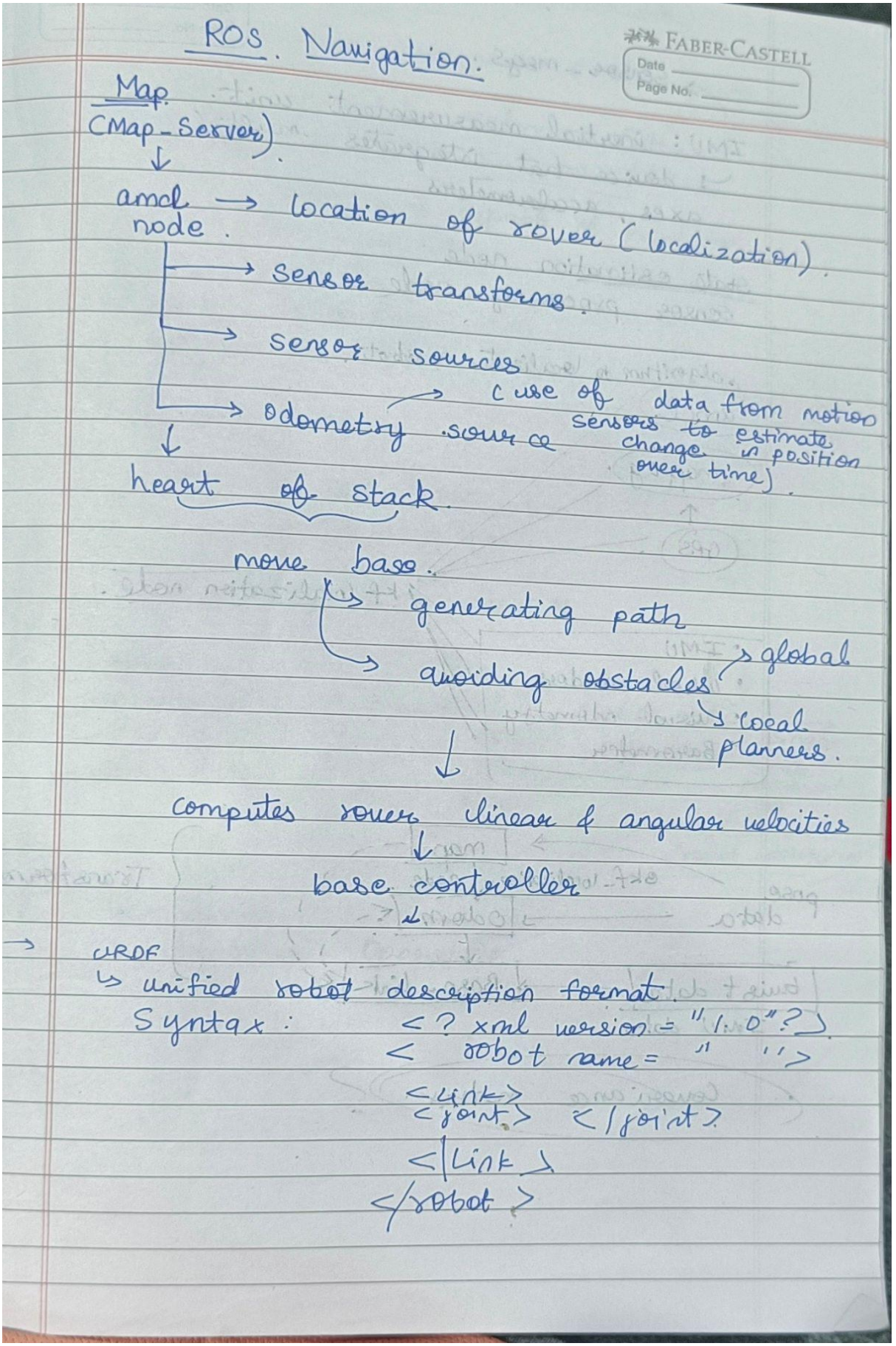
```
user ~ $ rostopic info /kobuki/laser/scan
Type: sensor_msgs/LaserScan

Publishers:
 * /gazebo (http://ip-172-31-34-208:48232/)

Subscribers: None
```

Global Planner and Local Planner

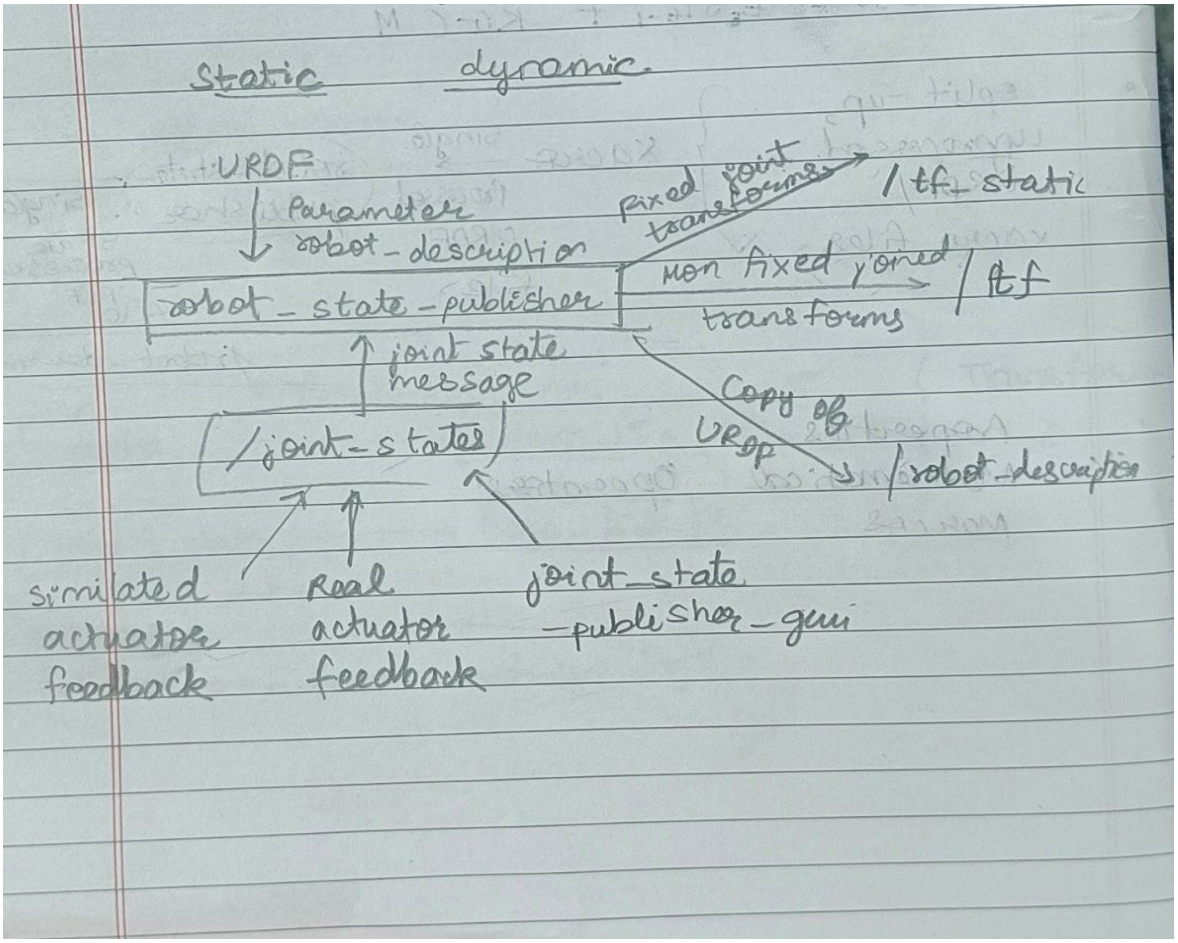
The global planner and local planner work together in a hierarchical manner to enable autonomous navigation:

- The global planner provides a long-term plan or path to reach the goal, considering the overall map and constraints of the environment.
- The local planner takes the global plan as input and generates short-term trajectories based on real-time information, adjusting the robot's trajectory to handle immediate obstacles and changes in the environment.
- The local planner continuously executes the short-term trajectory while ensuring that it aligns with the global plan, allowing the robot to follow the desired path while reacting to the immediate surroundings.

Flow diagram for ros navigation



Basic flow



Useful references:

Urdf Tutorial :
https://youtu.be/CwdbsvcpOHM

Reference for adding lidar ,cameras , depth cameras,adding simulation:https://www.youtube.com/watch?v=OWeLUSzxMsw&list=PLunhqkrRNRhYAffV8JDiFOatQXuU-NnxT&pp=iAQB

gazebo sensor plugins
https://classic.gazebosim.org/tutorials?tut=ros_gzplugins

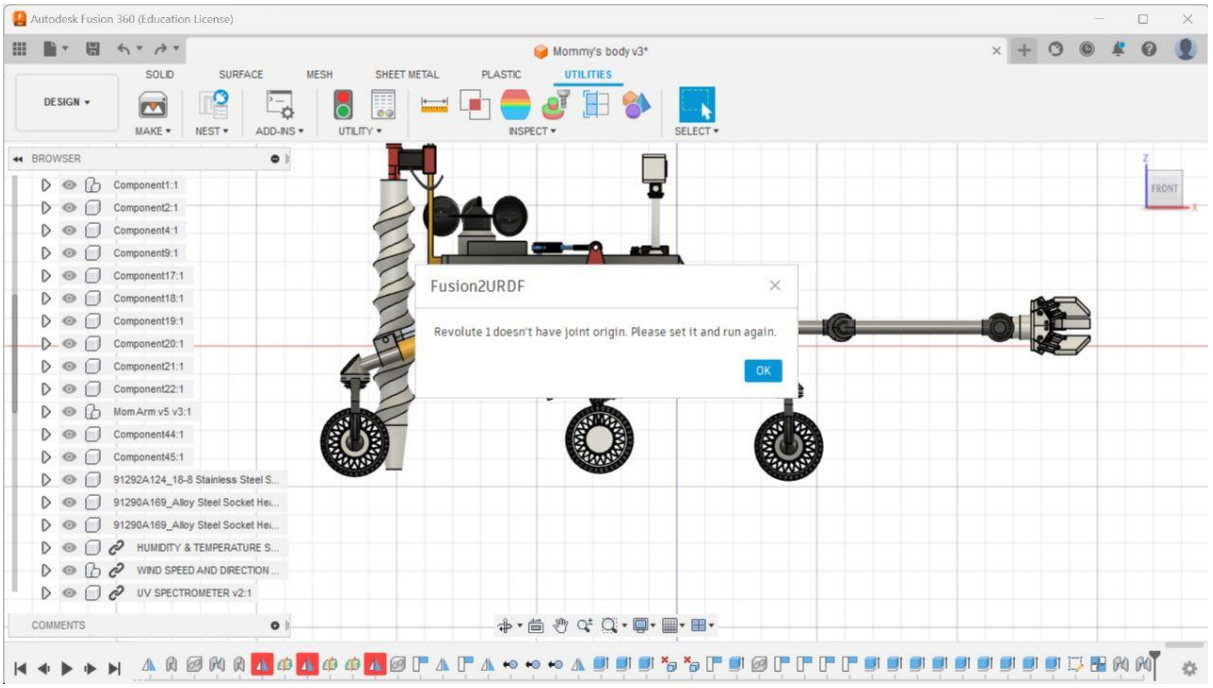
Mapping:
http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData

Rosnavigation:

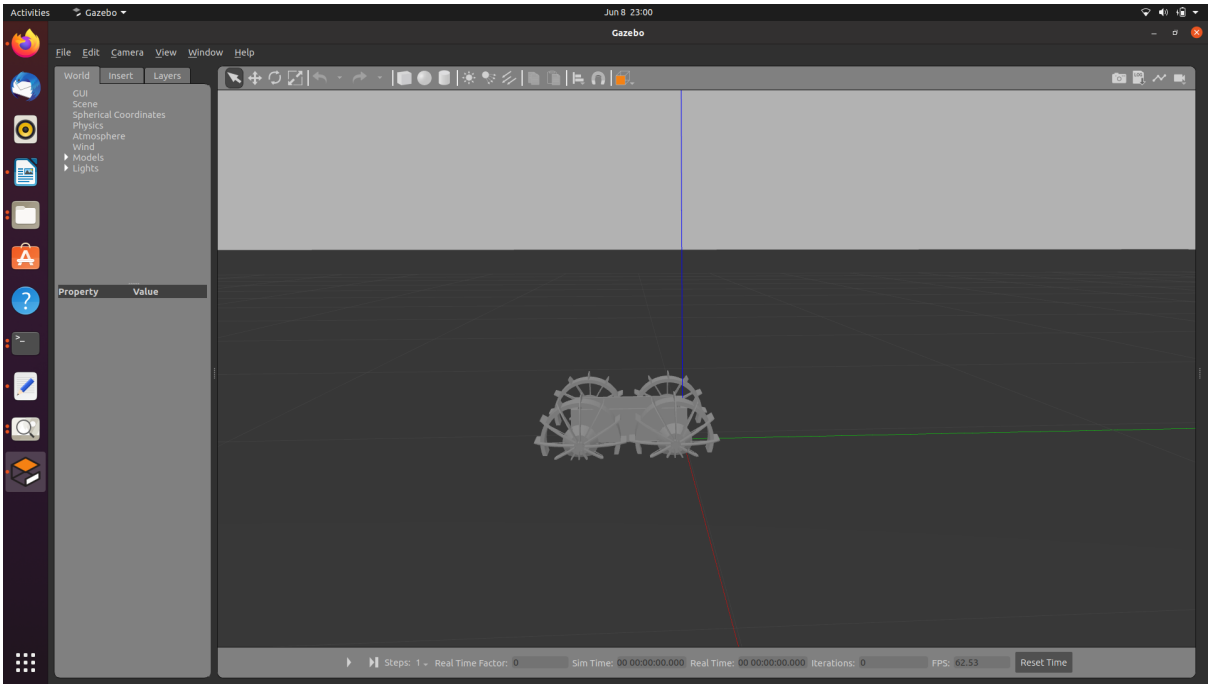
http://wiki.ros.org/navigation

Our work summary:

1. We first tried converting a fusion file of our mother body of irdc . It showed many errors as the orientation of the fusion file was not according to the specifications of urdf.



2. We then tried converting the fusion file of our daughter rover of irdc. It was converted but the wheels had negative coordinates . So the gazebo rover was not moving.



3. Then we decided to make a rover named trixy. It has a chassis, 2 wheels and chassis with a base_link

4. We made its urdf and added sensors like lidar,imu and depth camera to it in the trixy.xacro file . Their plugins are to be mentioned in the trixy.gazebo file.

5.We also created 3 different worlds: empty,house and mars_terrain

6.We also created a launch file launching gazebo and rviz and linking urdf with it.

7. We also made a basic controller to move the rover to desired coordinate.

Link for all files
https://drive.google.com/file/d/16mwLTS9Z-d5XmUTIoXddl6EknATOpFmL/view?usp=sharing