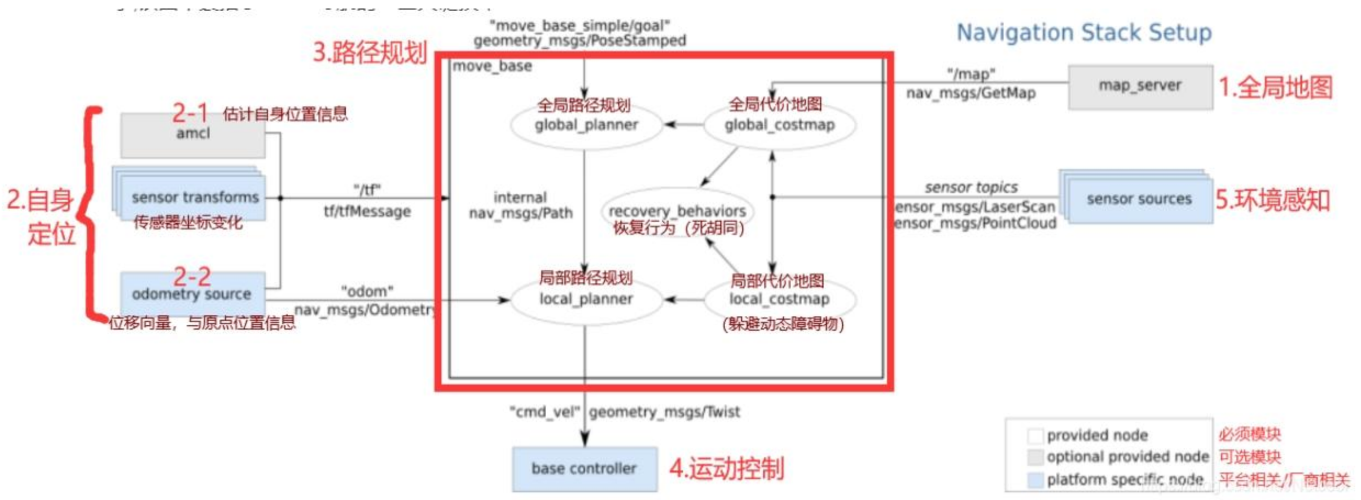
ROS Robot Autonomous Navigation Principle

1. Introduction to Autonomous Navigation

Autonomous navigation is the ability for a robot to move from point A to point B on its own. To realize such a function, the robot needs its own fixed components: global map, self-localization, path planning, motion control, and environment sensing. These components are used to realize the subsequent autonomous navigation function, which can be seen in the following pictures:



2. Principles of Autonomous Navigation

We already know that to achieve autonomous navigation, the robot needs to come with five components: **global map, self-localization, path planning, motion control,** and **environment sensing**, which correspond to the functional packages inside the robot.

The first one is the **global map**: it can provide a global map for robot navigation, and the corresponding map is constructed by SLAM.

Self-localization: an algorithm to calculate the robot's position in the map.
There are two general algorithms: one is to calculate the current position of the robot on the map by **using an odometer to calculate the current position and the origin position**; the other is to **use LIDAR sensors to sense the surrounding environmental information and then compare it with the feature points on the current map to derive the robot's position**.

Path planning: Paths are planned using global and local maps. The global map is obtained by SLAM construction, and the local map can be updated by real-time scanning with radar sensors to plan the path.

Motion Control: Controls the motion of the robot by sending messages to it.

Environment Sensing: Sensing the surrounding environment through radar sensors is helpful for SLAM to build maps, localize itself, and subsequent navigation functions.

Part 1 Adaptive Monte Carlo Localization

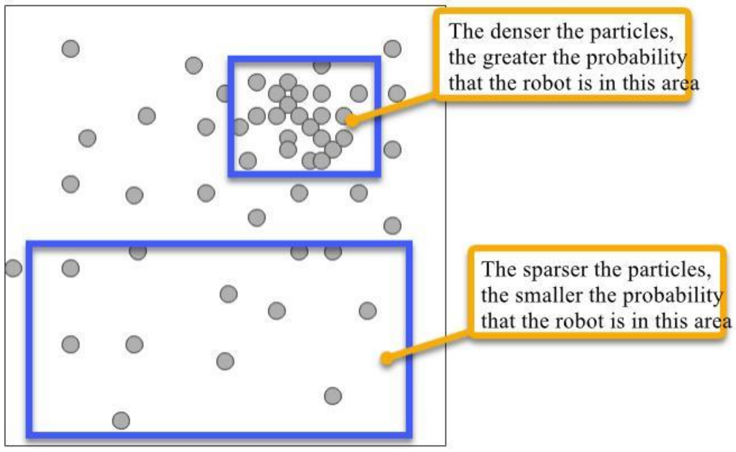
1. AMCL Definition

Adaptive Monte Carlo Localization is the official localization module for ROS/ ROS2, and the **only designated algorithm in the navigation module**. And it is based on **various Monte Carlo fusion algorithm**.

Adaptive Monte Carlo Localization is a probabilistic localization system of robot's movement in 2D space, and **it adopts the particle filter to track the position and orientation of a robot in the known map.**

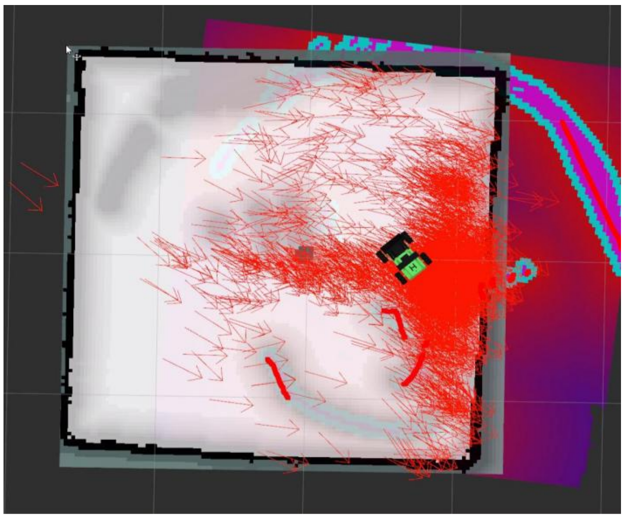
2. Working Principle

To **let the robot move to an accurate position**, you can **adopt Adaptive Monte Carlo Localization to adjust robot's position**, which works like that **put the particles uniformly on the map**, and **then these particles will gather in an area after algorithm calculation**. **The more the particles in an area, the greater the probability that the robot is in this area**.

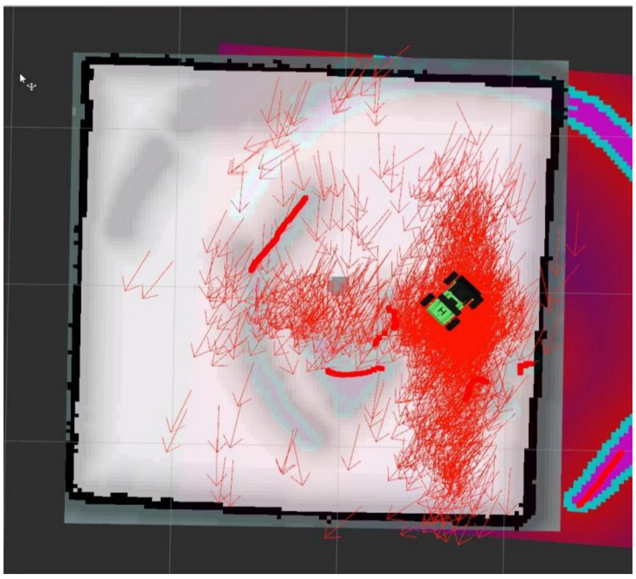

The denser the particles, the greater the probability that the robot is in this area
The sparser the particles, the smaller the probability that the robot is in this area

3. AMCL Effect

The red arrows indicate the estimated pose of the robot. **When multiple arrows are pointing in the same direction within a specific area, it is more likely that the robot is facing in that direction. Furthermore, if there are more arrows pointing in the same direction within a particular area, it is more likely that the robot is located there.**

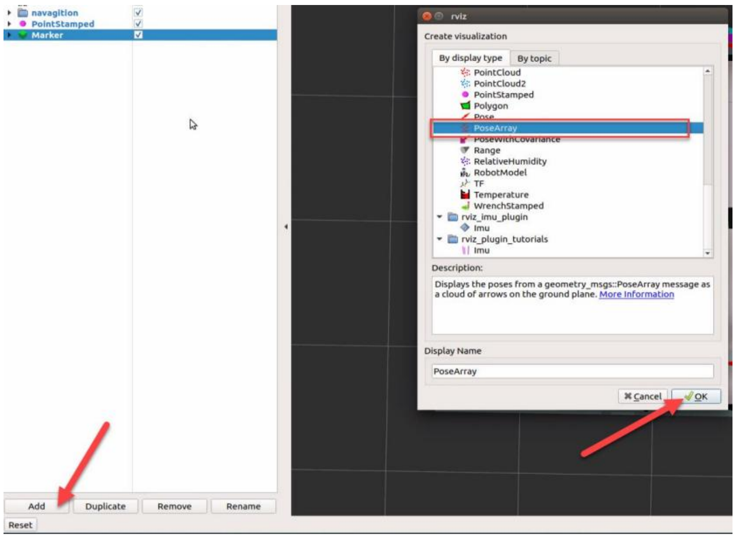


As you change the robot's position and direction, the direction arrows pointing to and their quantity will also change.



4. Display the Result of AMCL

Open RVIZ, then click on "**Add -> PoseArray -> OK**" in sequence to view the effect of AMCL..

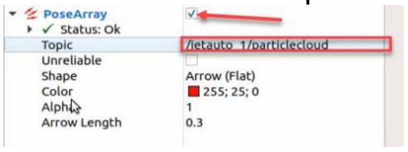


Description:

Displays the poses from a geometry_msgs::PoseArray message as a cloud of arrows on the ground plane. More Information

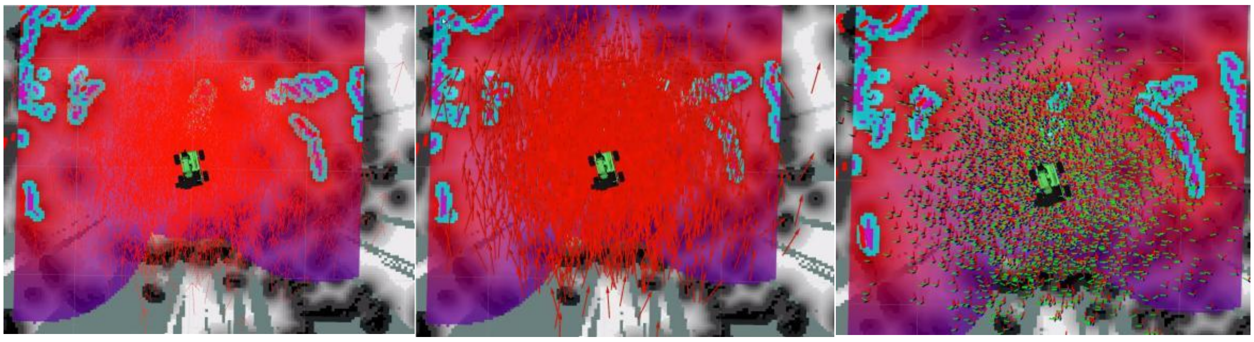
Display Name: PoseArray

To modify the arrow's display properties, you can adjust the relevant parameters. For instance, set data source node 'Topic' as "jetauto_1/particlecloud".

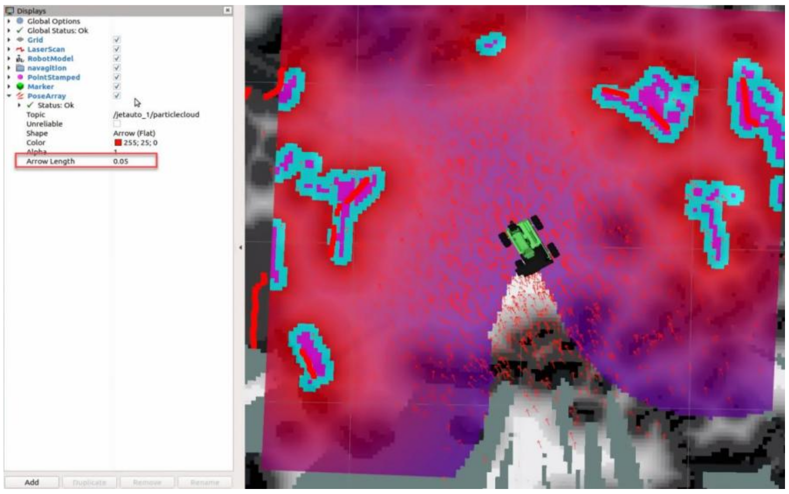


PoseArray	✓
Status: Ok	
Topic	/jetauto_1/particlecloud
Unreliable	
Shape	Arrow (Flat)
Color	🟥 255; 25; 0
Alpha	1
Arrow Length	0.3

The shape of arrows can be changed. Three pictures below show three styles of arrow.

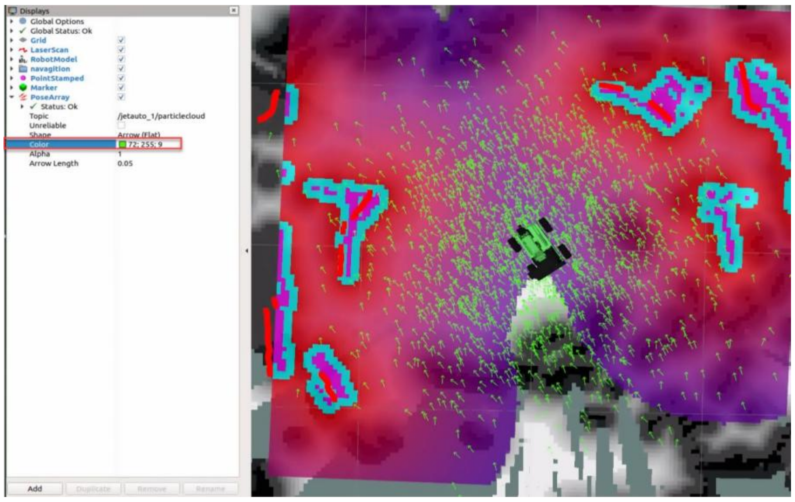


In the tap of “Arrow Length”, you can adjust the size of arrow.



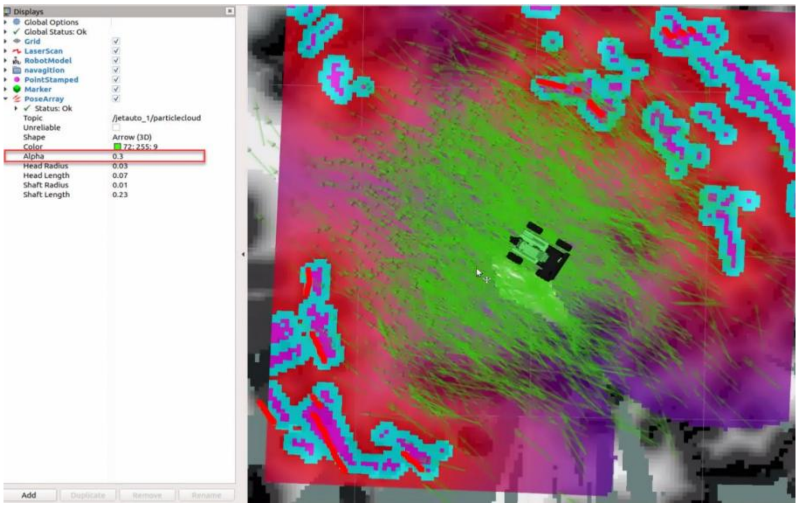
```
Displays
  Global Options
  Global Status: Ok
  Grid                  ✓
  LaserScan             ✓
  RobotModel            ✓
  navagition            ✓
  PointStamped          ✓
  Marker                ✓
  PoseArray             ✓
    Status: Ok
    Topic               /jetauto_1/particlecloud
    Unreliable
    Shape               Arrow (Flat)
    Color               🟥 255; 25; 0
    Alpha               1
    Arrow Length        0.05
```

To change color, you can make a adjustment in the tap of ‘Color’.



```
Displays
  Global Options
  Global Status: Ok
  Grid                  ✓
  LaserScan             ✓
  RobotModel            ✓
  navagition            ✓
  PointStamped          ✓
  Marker                ✓
  PoseArray             ✓
    Status: Ok
    Topic               /jetauto_1/particlecloud
    Unreliable
    Shape               Arrow (Flat)
    Color               🟩 72; 255; 9
    Alpha               1
    Arrow Length        0.05
```

Adjusting ‘Alpha’ enables you to change the transparency of arrows.



```
Displays
  Global Options
  Global Status: Ok
  Grid                  ✓
  LaserScan             ✓
  RobotModel            ✓
  navagition            ✓
  PointStamped          ✓
  Marker                ✓
  PoseArray             ✓
    Status: Ok
    Topic               /jetauto_1/particlecloud
    Unreliable
    Shape               Arrow (3D)
    Color               🟩 72; 255; 9
    Alpha               0.3
    Head Radius         0.03
    Head Length         0.07
    Shaft Radius        0.01
    Shaft Length        0.23
```

Making the aforementioned adjustment will not impact the performance of AMCL.

Part 2 TEB and DWA Path Planning

1. DWA Algorithm

1.1 DWA Algorithm Description

DWA (Dynamic Window Approach) is a algorithm that **mainly samples multiple sets of velocity in velocity space (v, w)**, and **predict their trajectories in the specific time according to the robot dynamics model.** Then **these trajectories will be scored by the evaluation function,** and **the optimal trajectory is picked to propel the robot to move**.

DWA algorithm **convert the position control of the robot into speed control. Before it predicts the robot motion trajectory under speed mode**, the **motion model of the robot should be analyzed firstly.**

v(t) and **w(t)** respectively refer to the **linear velocity** and **angular velocity** in the **world coordinate system**. In the **sampling period Δt**, if the **robot's displacement is small** and it executes **uniform linear motion**, the robot motion model is

.

$$\begin{cases} x(t) = x(t-1) + v(t)\cdot\Delta t\cdot cos\,(\theta(t-1)) \\ y(t) = y(t-1) + v(t)\cdot\Delta t\cdot sin\,(\theta(t-1)) \\ \qquad \theta(t) = \theta(t-1) + w(t)\cdot\Delta t \end{cases}$$

Among the formula, **x(t), y(t)** and **θ (t)** are the **pose of the robot** under the world coordinate system at **t** moment.

1.2 Velocity Space and Evaluation Function

DDW describes the **obstacle avoidance problem** as the **optimization problem with constraint in Velocity Space**. And, the constraints mainly contain **robot's incomplete constraint**, **environment obstacle constraint** and **robot structure dynamics constraint**.

Pick the optimal trajectory in the velocity space through designing the evaluation function. **In local path planning**, the conditions that **approaching the global path**, **finishing obstacle avoidance** and **moving toward the target rapidly** should all be satisfied so as to ensure the optimal trajectory is picked.

The evaluation function is defined as

$$G(v,w) = k(\alpha Heading(v,w) + \beta Goal(v,w) + \gamma Path(v,w) + \sigma Occ(v,w))$$

When G(v, w) is the minimum value, the optimal path is obtained. **k** is **a smooth function**, and **α, β, γ** and **σ** are **the weighting coefficients of each sub-function**. The detained explanation of the sub-function is as follow.

The first sub-function "**Heading**" works to **make the robot keep heading toward the ending point**. **The smaller the value of this function is, the smaller the azimuth of the robot to the end point is**.

The second sub-function "**Goal**" is employed to **evaluate the distance between the end of the local path and the ending point**, and **curtain the distance**(The distance to the end point is continuously reduced by this function).

The third sub-function “**Path**”is the path evaluation function used to **calculate the distance between the end of the trajectory and the global path**.

The fourth sub-function “**Occ**” is used to **evaluate the distance between the robot trajectory and the obstacle**, which represents **robot’s ability to avoid the obstacle**. If **the distance between the robot trajectory and the obstacle is greater than the radius of the robot, collision will not happen. Otherwise, the risk of collision is high, and this path should be abandoned.**

2. TEB Algorithm

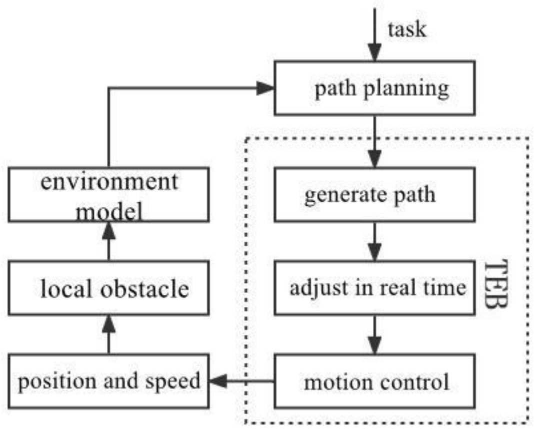
2.1 TEB Algorithm Description

TEB (Timed Elastic Band) algorithm **optimizes the motion trajectory through correcting the initial trajectory of the global path planning**. It mainly **optimizes the distance between the robot and the obstacle**, **the length of the path** and **execution time of the trajectory**.

TEB algorithm describes the path planning problem as a multi-objective optimization problem, which means that it will optimize the minimized execution time of the trajectory, and the constraint that keep certain distance between the robot and the obstacle, and follow the motion dynamics. As most of the optimization targets are local and only related to consecutive states of the robot, this optimization is targeted at sparse model.

2.2 TEB Model

The improved framework of the robot control system is as follow.



N discrete poses with time information constitute the trajectory generated by the TEB algorithm. And then **G2O (General Graph Optimization) algorithm is used to optimize these poses**, so that the **generated trajectory** is **the shortest**, **the least time-consuming**, and **the farthest from obstacles**. And **the speed and acceleration are limited** to make **the trajectory meet the requirements of robot kinematics.**

The coordinate of the center of the robot as well as **the turning direction determine its pose in the environment**.

Firstly, define the robot pose

$$X_a = (x_i, y_i, \beta_i)^T \in R^2 \times S^1$$

And **xi**, **yi**, and **βi** respectively correspond to the **position** and **posture orientation** in the map coordinate system.

Define the interval between the two pose as ΔTi to record the time sequence.

$$\tau = \{\Delta T_i\}_{i=0,1,\ldots,n-1}$$

Next, the pose and time sequence can be combined as

$$B := (Q, \tau)$$

The TEB algorithm **sets the posture and time interval as the variables to be optimized**, and **solves the optimal path under dynamic constraints.** These **dynamic constraints include velocity and acceleration limits, path lengths, distances between obstacles and the robot, and how long the robot will run on a trajectory**. Then, the **optimal path Q** is obtained **by setting the weighted multi-objective function.**

Part 3 Single-point Navigation and Obstacle Avoidance

1. Configuration

1) Start JetAuto, and then connect it to NoMachine.

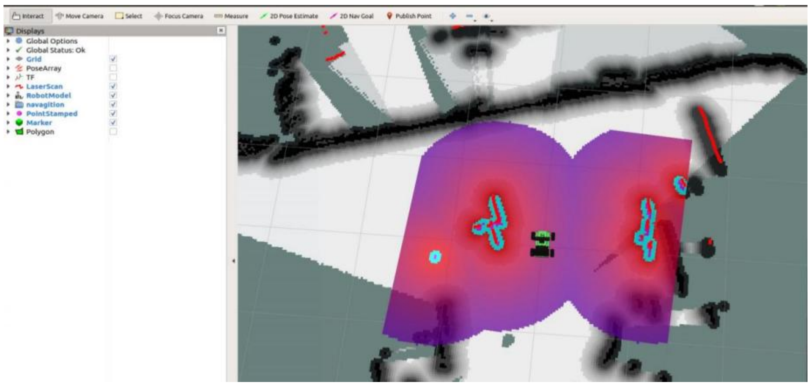
2) Open command line terminal.

3) Enter command "**sudo systemctl stop start_app_node.service**" and press Enter to stop APP service.

4) Open a new terminal, then enter command "**roslaunch jetauto_navigation navigation.launch map:=map_01**" and press Enter to enable navigation service.

"**map_01**" is the name of the map, and you can rename it. The map is saved in "**/home/jetauto_ws/src/jetauto_slam/maps**".

5) Open a new terminal, and input the command "**roslaunch jetauto_navigation rviz_navigation.launch**" and press Enter to open the software to view the model.



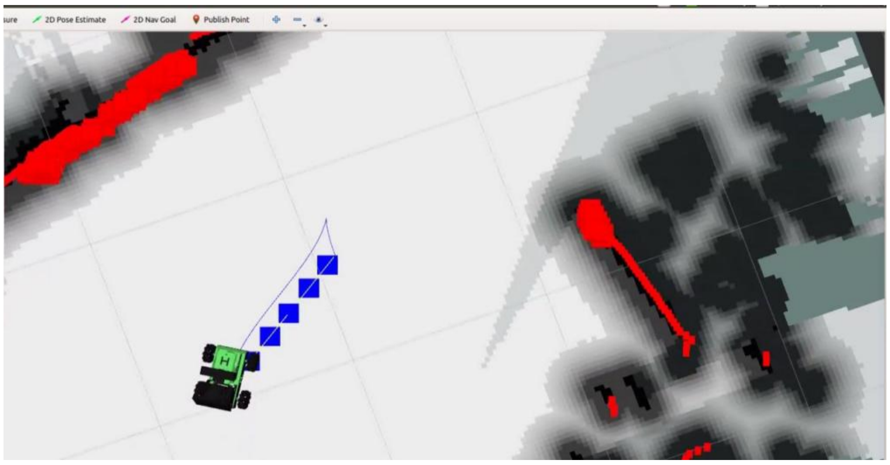
2. Navigation Obstacle Avoidance

There are three tools in the menu bar, including 2D Pose Estimate, 2D Nav Goal and Publish Point.

"**2D Pose Estimate**" is used to set the initial position of JetAuto, "**2D Nav Goal**" is used to set a target point and "**Publish Point**" is used to set multiple target points.

Click “**2D Nav Goal**” in the menu bar, and select one point by clicking the mouse as the target destination. After the point is set, JetAuto will automatically generate the route and move toward the point.

Two routes are displayed on the map. The line constructed by blue blocks is the straight path between the robot and the target point, and the dark blue line is JetAuto’s planned path.



Note: if you want to stop the navigation, set the current position of the robot as the target point with “2D Pose Estimate”. If the robot doesn’t touch the ground or is moved due to the external force, you need to reset the target point.

Part 4 Multi-point Navigation and Obstacle Avoidance

1. Configuration

1) Start JetAuto, and then connect it to NoMachine.

2) Open command line terminal.

3) Input command “**sudo systemctl stop start_app_node.service**” and press Enter to stop the APP service

4) Open a new terminal and input the command “**roslaunch jetauto_navigation navigation.launch map:=map_01**” and press Enter to enable navigation service.

“**map_01**” is the name of the map, and you can rename it. The map is saved in “**/home/jetauto_ws/src/jetauto_slam/maps**”.

5) Open a new terminal and input the command “**roslaunch jetauto_navigation rviz_navigation.launch**” and press Enter to open the software to view the model.

6) Open a new terminal and input the command “**roslaunch jetauto_navigation publish_point.launch**” and enable multi-point navigation service.

2. Navigation Obstacle Avoidance

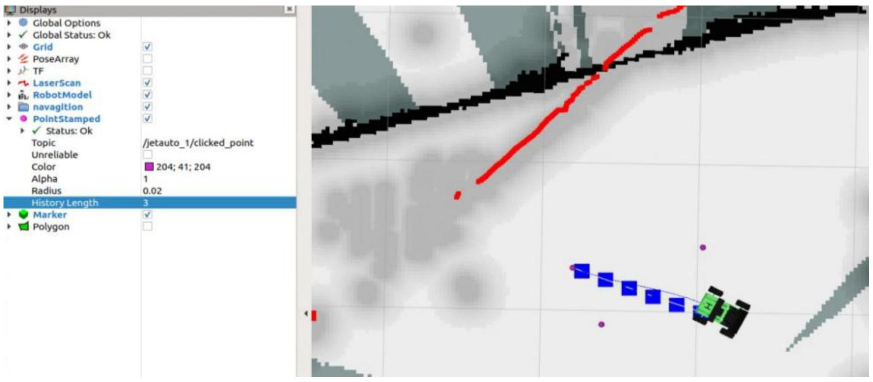
“**2D Pose Estimate**” is used to set the initial position of JetAuto, “**2D Nav Goal**” is used to set a target point and “**Publish Point**” is used to set multiple target points.

1) Click “**Publish Point**” in the menu bar, then select multiple points as the target destinations.

2) Among the target points, the latest point you set will be marked by purple dot, and only one purple dot is displayed on the map by default. And you can change the number under "**PointStamped->History Length**".

3) Click "**Publish Point**", then select the target points on the map. After all the points are set, the robot will generate the path automatically according to the sequence of the target points. Next, move to the target points in sequence.

Two routes are displayed on the map. The line constructed by blue blocks is the straight path between the robot and the target points, and the dark blue line is JetAuto's planned path.



```
Displays
▸ Global Options
▸ ✓ Global Status: Ok
▸ Grid                      ✓
▸ PoseArray                 ☐
▸ TF                        ☐
▸ LaserScan                 ✓
▸ RobotModel                ✓
▸ navagition                ✓
▾ PointStamped              ✓
    ▸ ✓ Status: Ok
    Topic                   /jetauto_1/clicked_point
    Unreliable              ☐
    Color                   204; 41; 204
    Alpha                   1
    Radius                  0.02
    History Length          3
▸ Marker                    ✓
▸ Polygon                   ☐
```

Note: if you want to stop the navigation, set the current position of the robot as the target point with "2D Pose Estimate". If the robot doesn't touch the ground or is moved due to the external force, you need to reset the target point.

Part 5 RTAB-VSLAM 3D Mapping and Navigation

1. RTAB-VSLAM Description

Real-Time Appearance-Based Mapping(RTAB-VSLAM) is an open-source library that adopts memory management approach for loop closure detection. It puts limit on the size of the map to make loop closure detection carried out in fixed time, so that **the requirements of long-period mapping and large-scale environment mapping can be met.**

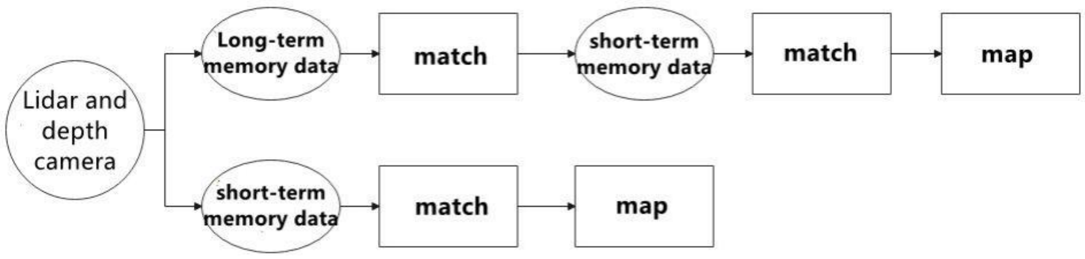
2. Logic of RTAB-VSLAM

RTAB-VSLAM is **feature-based mapping approach**. As the **general scenarios** can p**rovide abundant feature points**, this mapping method is **highly adaptive to diverse scenarios, and it can relocate with the feature points**. However it has defects. **It is time consuming to calculate the features.** Moreover, **the feature points contain little information and large part of information in the image are omitted.** Besides, **this mapping method doesn't work for weak texture areas.** And **the feature points are prone to mismatch**, which has a great impact on the results.

After the features in the picture are obtained, match the features at different times to implement loop closure detection. After the matching is completed, the data is divided into two types, namely **long-term memory data** and **short-term memory data**. **Long-term memory data is used to match future data**, and **short-term memory data is used to match current continuous data.**

During running, **RTAB-VSLAM algorithm will first uses short-term memory data to update the positioning point and mapping. When the data at a certain moment in the future can**

match the long-term memory data, **the corresponding long-term memory data is added to the short-term memory data to update the positioning point and mapping.**



3. Start Lidar Mapping

1) Start JetAuto and connect it to NoMachine

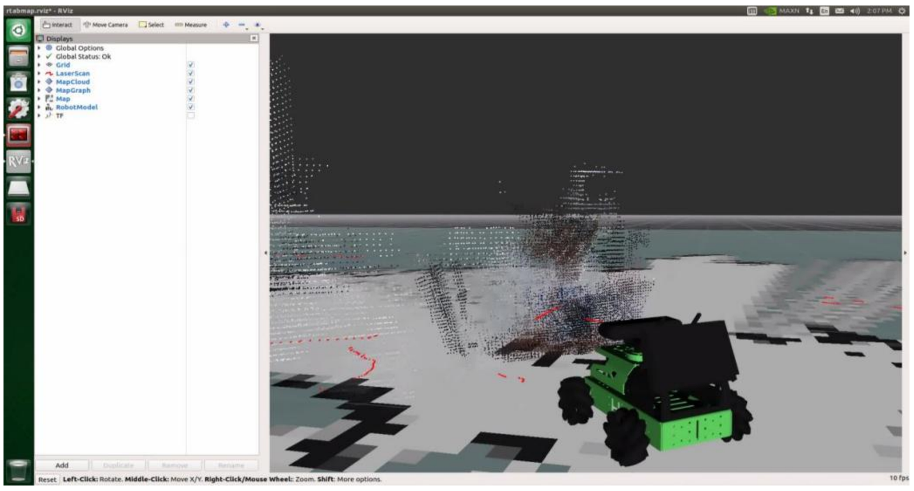
2) Open command line terminal.

3) Input command “**sudo systemctl stop start_app_node.service**” and press Enter to stop app service.

4) Open a new terminal. Input command “**roslaunch jetauto_slam slam.launch slam_methods:=rtabmap**” and press Enter to enable mapping service. If no error is reported, mapping service is enabled.

5) Open a new terminal. Input command “**roslaunch jetauto_slam rviz_slam.launch slam_methods:=rtabmap**” and press Enter to open model viewing software.

During JetAuto is mapping, you can also view the point cloud data obtained by the depth camera.



6) Open a new terminal, then input “**roslaunch jetauto_peripherals teleop_key_control.launch**” command to turn on keyboard control.

Keyboard control is enabled successfully when following message occurs.

7) Control robot to move with keys listed below so that robot can map entire surroundings.

8) After JetAuto finishes mapping, use shortcut ‘Ctrl+C’ to stop running the program.

Note: after you use shortcut ‘Ctrl+C’ to close mapping, the map will be automatically stored.

Having tried robot navigation, you can restart APP service by entering the appropriate command in the terminal or by restarting the robot. It is important to keep in mind that if the APP service is disabled, certain functions of the APP may become unavailable.

Enter command “**sudo systemctl restart start_app_node.service**” to restart APP service. When robot emits a sound, APP service is enabled successfully.

4. Navigation

1) Start JetAuto and connect it to NoMachine

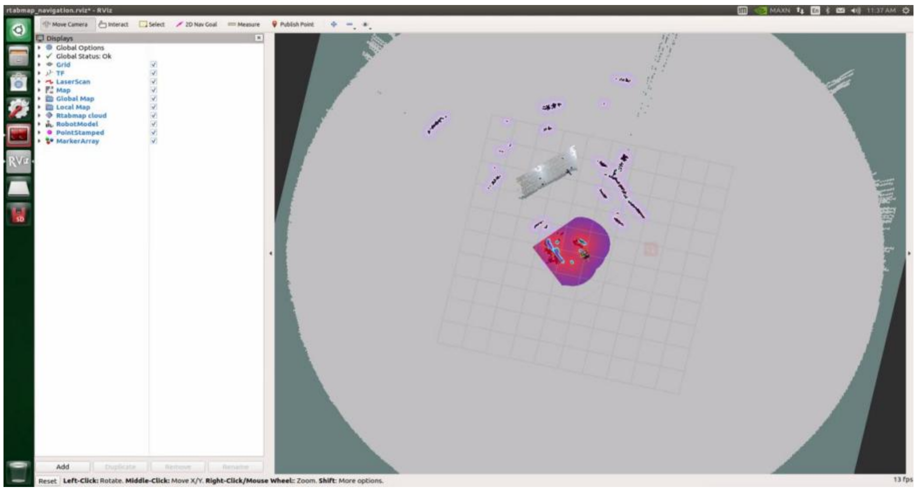
2) Open command line terminal.

3) Input command “**sudo systemctl stop start_app_node.service**” and press Enter to stop the APP service.

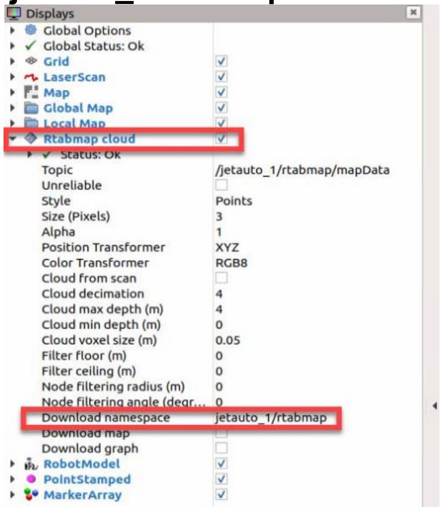
4) Open a new terminal and input the command “**roslaunch jetauto_navigation rtabmap_navigation.launch**” and press Enter to enable navigation service. If no error is reported, the service is enabled successfully. The last map will be used for 3D navigation.

5) Open a new terminal, then input command “**roslaunch jetauto_navigation rviz_rtabmap_navigation.launch**” to open the model viewing software.

During navigation, you can also view the point cloud data obtained by the depth camera as pictured.

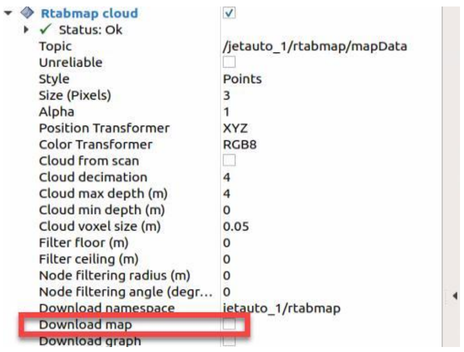


6) Click on ‘**Rtabmap_cloud**’ tap, then change the property of ‘**Download namespace**’ to ‘**jetauto_1/rtabmap**’.

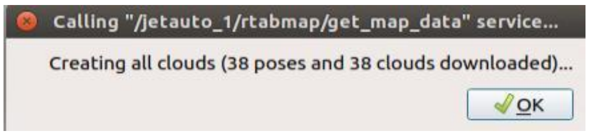

Displays

Global Options	
Global Status: Ok	
Grid	✓
LaserScan	✓
Map	✓
Global Map	✓
Local Map	✓
Rtabmap cloud	✓
Status: Ok	
Topic	/jetauto_1/rtabmap/mapData
Unreliable	☐
Style	Points
Size (Pixels)	3
Alpha	1
Position Transformer	XYZ
Color Transformer	RGB8
Cloud from scan	☐
Cloud decimation	4
Cloud max depth (m)	4
Cloud min depth (m)	0
Cloud voxel size (m)	0.05
Filter floor (m)	0
Filter ceiling (m)	0
Node filtering radius (m)	0
Node filtering angle (degr...	0
Download namespace	jetauto_1/rtabmap
Download map	☐
Download graph	☐
RobotModel	✓
PointStamped	✓
MarkerArray	✓

7) Tick ‘**Download map**’to load the map.

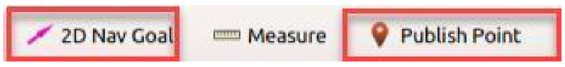


Rtabmap cloud	✓
Status: Ok	
Topic	/jetauto_1/rtabmap/mapData
Unreliable	☐
Style	Points
Size (Pixels)	3
Alpha	1
Position Transformer	XYZ
Color Transformer	RGB8
Cloud from scan	☐
Cloud decimation	4
Cloud max depth (m)	4
Cloud min depth (m)	0
Cloud voxel size (m)	0.05
Filter floor (m)	0
Filter ceiling (m)	0
Node filtering radius (m)	0
Node filtering angle (degr…	0
Download namespace	jetauto_1/rtabmap
Download map	☐
Download graph	☐



```
Calling "/jetauto_1/rtabmap/get_map_data" service...
Creating all clouds (38 poses and 38 clouds downloaded)...
OK
```

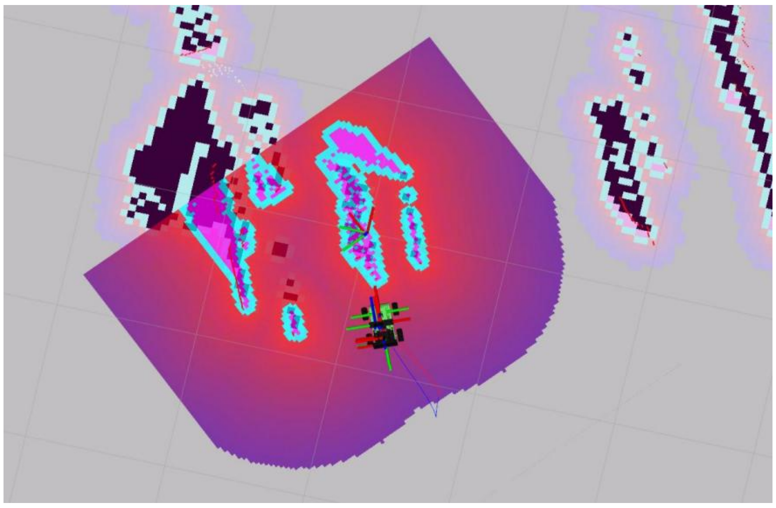
There are two tools in the menu bar, including **2D Nav Goal** and **Publish Point.**



2D Pose Estimate" is used to set the initial position of JetAuto, "**2D Nav Goal**" is used to set a target point and "**Publish Point**" is used to set multiple target points.

Click "**2D Nav Goal**" in the menu bar, and select one point by clicking the mouse as the target destination. After the point is set, JetAuto will automatically generate the route and move toward the point.

Two routes are displayed on the map. The red line is the straight path between the robot and the target point, and the blue line is JetAuto's planned path.



Enter command "**sudo systemctl restart start_app_node.service**" to restart APP service. When robot emits a sound, APP service is enabled successfully.

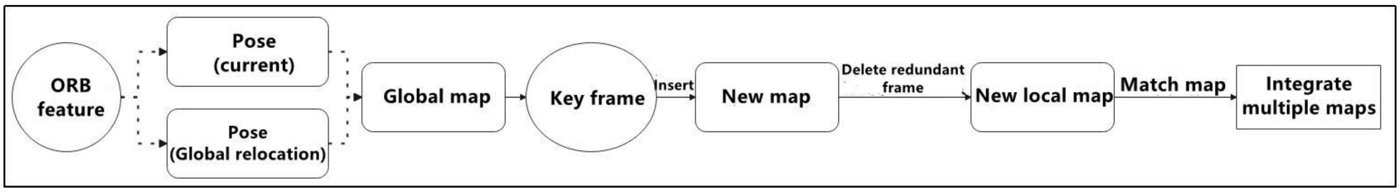
Part 6 ORBSLAM2 and ORBSLAM3 Mapping

1. ORBSLAM Description

ORB-SLAM2 is **a real-time feature-based SLAM system for Monocular cameras, which can operate in large, small, indoor and outdoor environment. This system contains the modules shared by all the SLAM systems, including tracking, mapping** and **relocating**. As it is a feature-based SLAM system, **it is able to compute the camera trajectory and realize a sparse 3D reconstruction.**

2. Logic of ORBSLAM

Firstly, it will extract the ORB feature from the image, and perform pose estimation or initialize the pose through global relocation.
Then track the local map constructed to optimize the pose, and confirm a new key frame.
Next, insert the new key frame into the map to generate new map point. After the map is verified, the redundant key frames will be deleted.
Lastly, match the obtained local map, and integrate the map after matching to adjust the global map.



3. Start ORBSLAM2 Modeling

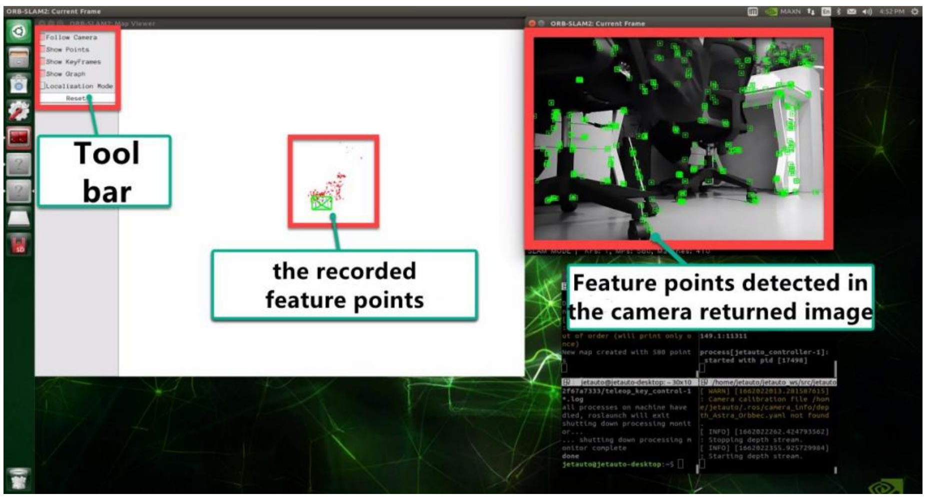
1) Start JetAuto and connect it to NoMachine

2) Open command line terminal.

3) Input command “**sudo systemctl stop start_app_node.service**” and press Enter to stop app service.

4) Open a new terminal. Input command “**roslaunch jetauto_example orb_slam2_rgbd.launch**” and press Enter to enable mapping service. If no error is reported, the mapping service is enabled successfully.

When the following window pops up, the service is enabled successfully


Follow Camera
Show Points
Show KeyFrames
Show Graph
Localization Mode
Reset
Tool bar
the recorded feature points
Feature points detected in the camera returned image

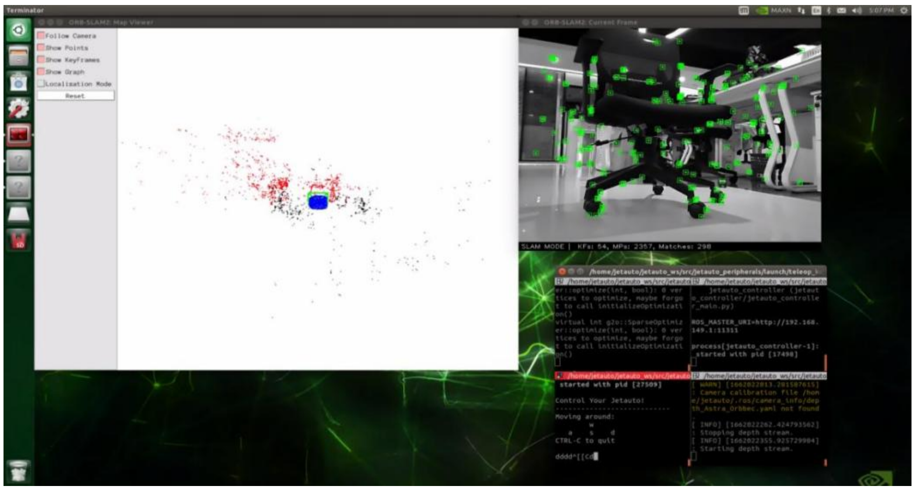
```
ut of order (will print only o    149.1:11311
nce)
New map created with 500 point    process[jetauto_controller-1]:
s                                  started with pid [17498]

jetauto@jetauto-desktop: ~ 30x10  /home/jetauto/jetauto_ws/src/jetauto
2f67a7333/teleop_key_control-1    [ WARN] [1662022013.281587615]
*.log                             : Camera calibration file /hom
all processes on machine have     e/jetauto/.ros/camera_info/dep
died, roslaunch will exit         th_Astra_Orbbec.yaml not found
shutting down processing monit    .
or...                             [ INFO] [1662022262.424793562]
... shutting down processing m    : Stopping depth stream.
onitor complete                   [ INFO] [1662022355.925729984]
done                              : Starting depth stream.
jetauto@jetauto-desktop:~$
```

5) Open a new terminal, and input command "**roslaunch jetauto_peripherals teleop_key_control.launch**" and press Enter to enable the keyboard control service.

If you receive the following hint, the keyboard control service is enabled successfully

6) Control the robot to move around to map by pressing the corresponding keys.

7) As JetAuto is moving, it collects lots of feature points which can be used to construct 3D model



```
SLAM MODE |  KFs: 54, MPs: 2357, Matches: 298
```

```
er::optimize(int, bool): 0 ver
tices to optimize, maybe forgo
t to call initializeOptimizati
on()
virtual int g2o::SparseOptimiz
er::optimize(int, bool): 0 ver
tices to optimize, maybe forgo
t to call initializeOptimizati
on()
```

```
    jetauto_controller (jetaut
o_controller/jetauto_controlle
r_main.py)

ROS_MASTER_URI=http://192.168.
149.1:11311

process[jetauto_controller-1]:
 started with pid [17498]
```

```
 started with pid [27509]

Control Your Jetauto!
---------------------------
Moving around:
        w
   a    s    d
CTRL-C to quit

dddd^[[Cd
```

```
[ WARN] [1662022013.281587615]
: Camera calibration file /hom
e/jetauto/.ros/camera_info/dep
th_Astra_Orbbec.yaml not found
.
[ INFO] [1662022262.424793562]
: Stopping depth stream.
[ INFO] [1662022355.925729984]
: Starting depth stream.
```

8) If you need to exit the current program, press "**Ctrl+C**".

4. Start ORBSLAM3 Modeling

1) Start JetAuto and connect it to NoMachine

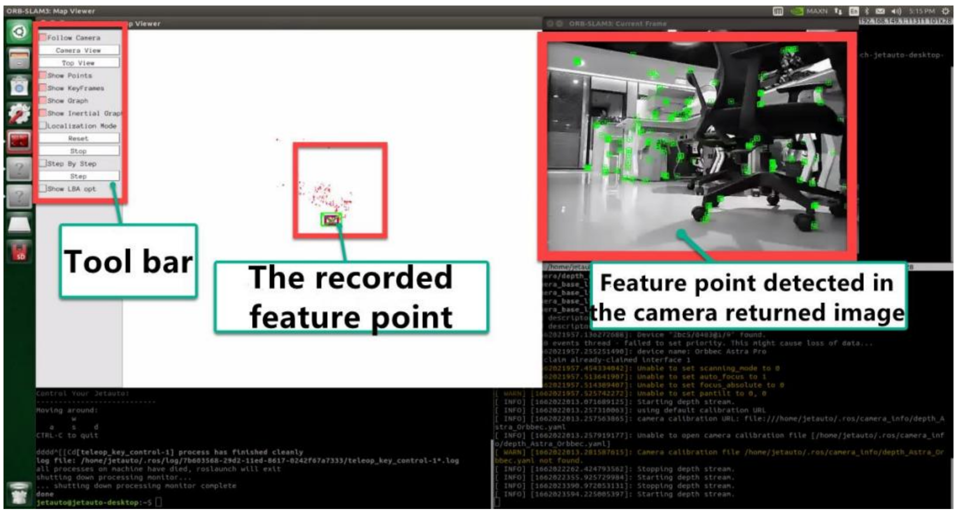
2) Open command line terminal.

3) Input command "**sudo systemctl stop start_app_node.service**" and press Enter to stop app service.

4) Input command "**roslaunch jetauto_peripherals astrapro.launch**" to enable basic Lidar service.

5) Open a new terminal and input command "**roslaunch jetauto_example orb_slam3_rgbd.launch**" and press Enter to enable modeling service. If no error is reported, the modeling service is enabled successfully.

When the following window pops up, the service is enabled successfully.



```
Control Your Jetauto!
---------------------------
Moving around:
        w
   a    s    d
CTRL-C to quit

dddd^[[Cd[teleop_key_control-1] process has finished cleanly
log file: /home/jetauto/.ros/log/7b603568-29d2-11ed-8617-0242f67a7333/teleop_key_control-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
jetauto@jetauto-desktop:~$ 
```

```
62021957.136272686]: Device '2bc5/0403@1/9' found.
0 events thread - failed to set priority. This might cause loss of data...
62021957.255251490]: device name: Orbbec Astra Pro
claim already-claimed interface 1
62021957.454334042]: Unable to set scanning_mode to 0
62021957.513641907]: Unable to set auto_focus to 1
62021957.514389407]: Unable to set focus_absolute to 0
[ WARN] [1662021957.525742272]: Unable to set pantilt to 0, 0
[ INFO] [1662022013.071689125]: Starting depth stream.
[ INFO] [1662022013.257310063]: using default calibration URL
[ INFO] [1662022013.257563865]: camera calibration URL: file:///home/jetauto/.ros/camera_info/depth_A
stra_Orbbec.yaml
[ INFO] [1662022013.257919177]: Unable to open camera calibration file [/home/jetauto/.ros/camera_inf
o/depth_Astra_Orbbec.yaml]
[ WARN] [1662022013.281587615]: Camera calibration file /home/jetauto/.ros/camera_info/depth_Astra_Or
bbec.yaml not found.
[ INFO] [1662022262.424793562]: Stopping depth stream.
[ INFO] [1662022355.925729984]: Starting depth stream.
[ INFO] [1662023390.972053131]: Stopping depth stream.
[ INFO] [1662023594.225805397]: Starting depth stream.
```

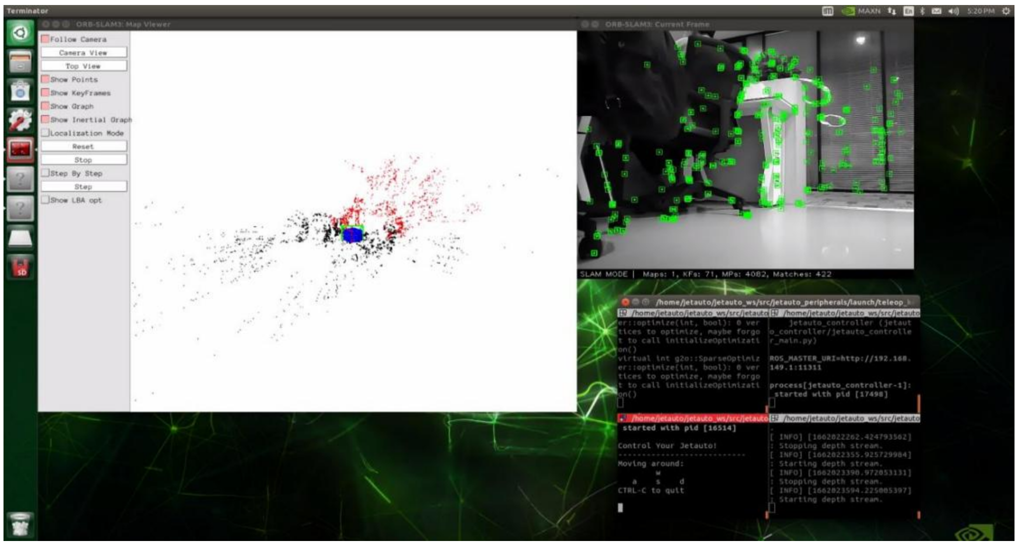
6) Open a new terminal, and input command “**roslaunch jetauto_controller jetauto_controller.launch**” and press Enter to enable motion control service.

7) Open a new terminal, and input command “**roslaunch jetauto_peripherals teleop_key_control.launch**” and press Enter to enable the keyboard control service.

If you receive the following hint, the keyboard control service is enabled successfully.

8) Control the robot to move around to map by pressing the corresponding keys.

9) As JetAuto is moving, it collects lots of feature points which can be used to construct 3D model



```
SLAM MODE |  Maps: 1, KFs: 71, MPs: 4082, Matches: 422
```

```
/home/jetauto/jetauto_ws/src/jetauto_peripherals/launch/teleop_
```

```
er::optimize(int, bool): 0 ver
tices to optimize, maybe forgo
t to call initializeOptimizati
on()
virtual int g2o::SparseOptimiz
er::optimize(int, bool): 0 ver
tices to optimize, maybe forgo
t to call initializeOptimizati
on()
```

```
    jetauto_controller (jetaut
o_controller/jetauto_controlle
r_main.py)

ROS_MASTER_URI=http://192.168.
149.1:11311

process[jetauto_controller-1]:
 started with pid [17498]
```

```
 started with pid [16514]

Control Your Jetauto!
---------------------------
Moving around:
       w
   a   s   d
CTRL-C to quit
```

```
.
[ INFO] [1662022262.424793562]
: Stopping depth stream.
[ INFO] [1662022355.925729984]
: Starting depth stream.
[ INFO] [1662023390.972853131]
: Stopping depth stream.
[ INFO] [1662023594.225005397]
: Starting depth stream.
```

10) If you need to exit the current program, press “**Ctrl+C**”