SLAM 3.0 API

Serial communication

```
Serial port baud rate: 115200
Navigation host serial port: /dev/XX
**Command package:**
- The communication method is a serial asynchronous method, and the communication content between the ROS system and the host computer is an ASCII coded string.

| Frame Header (H) | Length (L) | Data (D) | Check Digit (S)|
| :--- | -----------:| :---------: |:------:|:------:|
|AA 54 | | | | |


**Instruction package data definition:**
- Frame header (H): AA 54
- Length (L): length of data (D)
- Data (D): byte array converted from string
- Parity bit (S): XOR calculation of length (L) and data (D)


**Response Packet:**
- The communication method is a serial asynchronous method, and the communication content between the ROS system and the host computer is an ASCII coded string.

| Frame Header (H) | Length (L) | Data (D) | Check Digit (S)|
| :___ | ___________:| :_________: |:______:|:______:|
| AA 54 | | | | |


**Response packet data definition:**
- Frame header (H): AA 54
- Length (L): length of data (D)
- Data (D): byte array converted from string
- Parity bit (S): XOR calculation of length (L) and data (D)
```

ROS actively reports

1. imu, laser, odometer, 3d camera status

```
response: check_sensors{imu lidar odom 3dcam ros}
mark:
imu: imu state;
lidar: Lidar status;
odom: odometer status;
3dcam: 3d camera status;
ros: ros service status;
0: Abnormal; 1: Normal;
The navigation system actively reports, once every 5s
```

2. navigation state

```
response:
nav_result{state code name dist_to_goal mileage}
mark:
   state:
         0: initial state;
         1: The machine starts to navigate;
         2: Pause;
         3: Navigation is complete;
         4: Cancellation of navigation;
         5: Navigation recovery;
         6: The upper computer sends the navigation command successfully, but the navigation has not started yet;
      Navigation process:
           send : nav_point[A]//send navigation target point
           resp : nav_result{6 0 A -1 0}//Start navigation
           resp : nav_result{1 0 A 0.562001 0}
           resp : nav_result{3 0 A 0 0}//navigation completed
           resp : nav_result{0 0 A -1 0}
code:
      When sending navigation commands:
      0 : success;
     -1 : docking charging pile;
     -2 : emergency stop switch is pressed;
     -3 : The adapter is charging;
    -4 : target point not found;
    -5: AGV docking failed;
    -6: Abnormal positioning;
    -7: The distance between fixed route points is too large;
    -8 : No fixed route found;
    -9: Failed to read point information;
   Send pause/resume navigation command:
```

```
        0 : success;
       -1 : failed;
    Send Cancel Navigation: 0;
    Navigation complete:
       0 : success;
     -1 : failed;
  name:
   Target point navigation: target point name;
    Coordinate Navigation :
       x: x-axis coordinates;
       y : y-axis coordinate;
        radian : radian
 dist_to_goal : the distance from the target point;
  mileage : navigation mileage;

```

3. Navigation error state

```
response: move_status:x
mark:
      x:
           3: Unable to plan the route;
            4: There are obstacles in the local path;
```

4. core state

```
response: core_data{data1 data2 data3 data4 data5}
 mark:
    data1: Collision:
          1: center; 2: back; 4: left; 8: right;
    data2: anti-drop:
          4: left; 2: center; 1: 3D;
    data3: emergency stop:
          0: press the release axis; 1: lift up the navigation;
    data4: Power:
          battery percentage;
    data5: charging status:
          1: Not charging; 2: Charger; 3: Adapter; 8: Docking; 9: The charging pile was not
found when docking; 11: The infrared code was not received after timeout; 12: The infrared
code was  not received in the second stage of the docking charging pile ; 13: The third
stage of docking charging pile did not receive the infrared code
```

5. wheel status

```
response:
    wheel_status{work_state current_left current_right temp_left temp_right driver_temp_left driver_temp_right code_left code_right model version}
mark:
    work_state : work state
        loose axis 0x00
        Shaft lock 0x01
        Wheel failure 0x02
        brake 0x04
        stop (stop) 0x08
        Offline status 0x80
current_left : left wheel current (mA);
current_right : right wheel current (mA);
temp_left : left wheel temperature (°C);
temp_right : right wheel temperature (°C);
driver_temp_left : left wheel driver temperature (°C);
driver_temp_right : right wheel drive temperature (°C);
code_left : left wheel error code;
code_right : right wheel error code;
model : wheel model;
version : wheel version number;
```

6. point update

```
response: waypoint:update
mark: report when the point is updated
```

7. Fixed route updates

```
response: pathmodel:update
mark: report when the fixed route is updated (not supported before 3.2.2)
```

8. shutdown

```
response: power_off:x
mark:
    x : 1 : software shutdown; 2 : restart; 3 : button shutdown; 4 : low power shutdown;
Android shutdown is required upon receipt of a shutdown report;
```

9. Report speed during navigation

```
response: base_vel[line_speed angular_speed]
mark:
    line_speed : line speed;
    angular_speed : angular speed;
```

10. Report the rear ultrasonic wave when docking with the charging pile

```
response:  range_sensor{data1 data2 data3}
mark:
      data1 : post-ultrasonic;
```

11. global path

```
response: global_path: & global_path1:
mark:
      global_path: global path, if it cannot be fully reported once, it will be reported twice (global_path will end with  +)
```

12. agv coordinate reporting

```
response: agv_tag_pose{tag_name pose1 pose2 pose3 poseX,poseY,poseAngle}
mark:
    tag_name:tag code name;
    poseX:X-axis offset;
    poseY:Y-axis offset;
    poseAngle:angle;
```

13. agv navigation results

When performing agv navigation, ros will first report nav_result, indicating that navigation to the target point is successful, then adjust the position according to the QR code, and then perform QR code navigation. Here is the result of QR code navigation;

```
agv navigation success:
      response: agv_success{tag}
          mark:
                  agv navigation success;
                  tag: target tag code;
 agv navigation failure:
       response:agv_fail
           mark:
              agv navigation failure;
```

14. agv navigation results

```
response: misspose:code
mark:
        code:
                0: reposition, position reset
                1: position lost
```

15. Generate fixed route

```
request: short_dij:& short_dij1:
mark:
        If the report cannot be completed in one time, it will be reported in multiple times (if the report is not completed, it will end with +)
```

16. Report special zones that global paths pass through

```
response: special_plan:{"sp":[{"n":"sp-b","c":[-6.03,-0,-8.66,-2.08],"type":0}]}
mark:
    n: special zone name;
    c: intersection coordinates of global route and special zone;
    type: special zone type:
                        0: default;
                        1: one-way zone;
                        2: gated zone;
                        3: turning zone;
                    4: custom;
```

17. Enter the special area

```
response: special_area[sp-a,0,-1.0]
mark:
    sp-a: special area name;
    0: special area type;
   -1.0: speed when passing through the special area, -1 is the default speed;
```

18. Leaving the Special Zone

```
response: special_area:out
```

Unique to Snail Sweeper

1. Cleaning progress

```
response: clean_room[running:progress,cleanroom_id]
mark:
    progress: progress;
```

2. Cleaning completed

```
response: clean_room[complete:code,clean_id]
mark:
        code:
                0:completed;
```

Unique to Forklift

1. Forklift related status

```
response: forklift{arm_location position_sensor dock_state control_state reserved}
          arm_location: forklift arm position
                0: middle;
                1: top;
                2: bottom;
                3: abnormal;
          position_sensor: pallet in place sensor
                0: not in place;
                1: in place;
    dock_state: docking state
                0: initial state;
                1: docking;
                2: docking successful:
                3: lateral deviation adjustment successful;
                4: visual recognition of pallet not available;
                5: near-end verification failed;
                6: collision with pallet;
                7: obstacle avoidance failed;
           control_state: forklift control state
                0: ros control;
                1: manual control;
           reserved: reserved position;
```

The host computer actively requests

1. heartbeat packet

```
request: keep_connect
response:
        hfls_version:HardwareVersion FirmwareVersion LoaderVersion SoftVersion
mark:
      HardwareVersion power board version number, FirmwareVersion power board firmware version number, LoaderVersion: loaderVersion, SoftVersion navigation system version number
Keep requesting once every 5 seconds, and the watchdog of the navigation system will check the connection status of the serial port
```

2. Shut down the whole machine

```
request: power_off
response: power_off: 1
```

3. restart navigation

```
request: sys:reboot
response: initpose:0, xy radian
mark:
      The navigation system will actively report when starting up/restarting the navigation/switching the map (in these cases, the navigation will be relocated;
xy radian: machine coordinates;
```

4. Laser data reporting

```
request: switch_lidar[open?on:off]
response: laser[distance]
mark:
    open: on : open; off : close;
    distance : distance, unit: m, accurate to cm;
```

5. Navigation host connected to wifi

```
request: wifi[ssid name;pwd password]
response:
        wifi:connect success The connection is successful
        wifi:connect fail connection failed
        wifi:connecting connection in progress
mark:
      name is the WIFI name
      password is the WIFI password
      WIFI only supports WPA/WPA2 Personal protocol
```

6. Get the navigation host Hostname

```
request: hostname:get
response: sys:boot:robot-18
mark:
      robot-18 is the Hostname of the navigation host
```

7. Get navigation system version

```
request: sys:version
response: ver:3.0.0
```

8. Get navigation wifi name and ip

```
request: ip:request
response: ip:ssid:xxxx/wlan:127.0.0.1
mark:
  ssid: wifi name
  xxxx: ip
  wlan:127.0.0.1: network not connected
```

9. Query network port ip

```
request: ip_lan: request
response: ip:enp2s0:192.168.10.2 / ip:lan:127.0.0.1
```

10. get current map

```
request: nav:current_map
response: current_map[map alias]
mark:
  map: current map md5
  alias: alias (if there is no alias response:current_map[map])
```

11. query navigation mode

```
request: model: request
response: model:x
mark:
    x: 1(navigation mode; 2(mapping mode; model:3(incremental mapping mode;
    After starting up, the navigation system will actively report once;
    The navigation system will automatically report when switching modes;
```

12. Toggle navigation mode

```
request: model: mode
response: model:x
mark:
    | mode | x | remark |
    | --------- | ------ | ---------------- |
    | navi | 1 | navigation mode |
    | mapping | 2 | Mapping mode |
    | remap | 3 | incremental mapping mode |
```

13. save map

```
request: save_map
response: model: 1
mark:
     Called after the drawing is completed
```

14. switch map

```
request: call_web[apply_map:map]
response: apply_map[map alias]
     apply_map: failed
mark:
     map : map md5
     alias : alias (if no alias response: apply_map[map])
     failed : switch map failed
     After the map switching is completed, the navigation will also be repositioned, and the repositioning is completed and reported to `initpose:0, xy radian`, which can be used as a sign that the  map is switched;
```

15. Get the coordinates of the marked point

```
request: nav:get_flag_point[name]
response:
     get_flag_point[x,y,radian,type,name]
     get_flag_point: -1
mark:
     x,y : coordinates
     radian : radian
     type : type
     name : point name
     -1 : The calibration point cannot be found
```

16. set calibration point

```
request: nav:set_flag_point[x,y,radian,type,name]
response:
     set_flag_point:result
mark:
     x,y : coordinates;
     radian : radian;
     type : charging pile type fixed: charge
     Delivery point type: delivery
     Product type: production
     Recycling point type: recycle
```

```
      Waypoint Type: normal
      Others: Types can be customized according to business needs
name : point name
result:
      0 : success
     -1 : the name length is greater than 50
     -2 : file parsing failed
     -3 : point repetition
```

17. Delete calibration point

```
request: nav:del_flag_point[name]
response:
      del_flag_point:result
mark:
      name : the point to delete
      result:
          -1 : file not found
          -2 : file parsing failed
          -3 : point not found
```

18. reset

```
request:
nav:reloc[x,y,radian] (relocate at (x,y,radian) coordinates;
nav:reloc_name[point] (relocate at the point of the specified name;
response: initpose:0, xy radian
mark:
The navigation system will actively report when starting up/restarting the navigation/switching the map (in these cases, the navigation will be relocated;
xy radian: machine coordinates;
```

19. Absolute relocation

(Note: Sending a name to relocate obtains the point position in the "calibration position" mode)

```
request:
        nav:reloc_absolute[point name]
response: initpose:0,x y radian
mark:
       point name:point name;
       Do not match the map environment, and force relocation to the specified point;
```

20. Absolute position relocation (coordinate relocation)

(Note: Only supports RSNF-3.2.2 and above)

```
request:
          nav:reloc_abpoint[x,y,radian]
response: initpose:0,x y radian
mark:
       point name:point name;
       Do not match the map environment, force relocation to the specified coordinates;
```

21. move robot

```
request:
      move[distance,angle] [int, int] type
      move[distance,angle,speed] [int, int, double] type
      response: move:done:xx
mark:
     1. One of distance and angle must be 0, that is, the robot can only walkin a straight
line or turn, and cannot walkin an arc
     2. If move[0,0] is sent, it means stop moving
     3. The unit of distance is mm
```

```
        4. The unit of angle is Euler angle, and the value range is between [-180,180]
        5. speed is the speed parameter, if not specified, the robot will walk at the default speed (0.3 meters per second for a straight line, 40 degrees per second for a turning angle)
  Second)


        6. speed must be greater than 0, otherwise it will be processed according to the default speed
        7. If speed is specified and distance is not 0, speed represents the speed of walking in a straight  line, and the unit is m/s
        8. If speed is specified and angle is not 0, speed represents the speed of the corner, and the  unit is degree/second
        9. Reply to move:done:16: Go straight and complete
       10. Reply to move:done:17: left turn completed
       11. Reply to move:done:18: turn right is completed
       12. move:done:19: Backward completed
       13. move:done:20: Cancel
       14.There is an obstacle: move_status:6
```

22. Navigate given target point name

```
request: nav_point[foreground]
response:
      Refer to `ROS active reporting - 2. Navigation status`
mark:
      When the sending target point type is `charge`, connect to the charging pile after successful navigation
      The Chinese target point needs to be set under the "Calibrate Position" option on the web page
      Chinese character encoding method utf8, non-Chinese character encoding method ASCII encoding
```

23. coordinate navigation

```
request: goal:nav[x,y,radian]
response:
      Refer to `ROS active reporting - 2. Navigation status`
mark:
      x,y are coordinates, radian is radians
```

24. pause navigation

```
request: nav_pause
response:
    Refer to `ROS active reporting - 2. Navigation status`
```

25. restore navigation

```
request: nav_resume
response:
Refer to `ROS active reporting - 2. Navigation status`
```

26. cancel navigation

```
request: nav_cancel
response:
Refer to `ROS active reporting - 2. Navigation status`
```

27. Get the current coordinates

```
request : nav:get_pose
response:
        nav:pose[x,y,radian] current robot coordinates
        nav:pose:notfound The robot is positioning
mark:
        x, y, radian are floating point numbers with two decimal places, where x, y are coordinates, radian is radian
```

28. Modify the navigation maximum speed

```
request: max_vel[0.6]
mark:
     Range: [0.3,1.0]
    Used to adjust the maximum speed of navigation
```

29. charging docking

```
request: dock:start
response:
    Refer to `ROS active reporting - 3. Core status`
```

30. Undock

```
request: dock: stop
response: none
    mark: This command is valid during the backward process of docking the charging pile
```

31. Get battery fixing information

```
request: get_battery_info
response:
    battery_info{manufacturer nominal_voltage temperature cycle_times rated_capacity full_capacity capacity health}
mark:
    manufacturer : manufacturer
    nominal_voltage : nominal voltage
    temperature : temperature
    cycle_times : cycle times
    rated_capacity : rated capacity
    full_capacity : full capacity
    capacity : current capacity
    health : health degree
```

32. Get battery dynamic information

```
request: get_current_info[open?1:0]
response: current_info{55 202 0 0 1}
        //Voltage current adapter current warning offline
mark:
        open: 1 : open; 0 : close;
        voltage : voltage;
        current : current;
        adapter_current : adapter current; (not yet implemented, requires hardware support)
        warning : warning;
        offline : 0: offline 1: online;
```

33. Get navigation dynamic parameters

```
request: update_dynamic
response:
    get_max_vel: speed
    get_stop_time : second
    get_global_p : p
    get_max_plan_dist :width
    get_stop_nearby : state
    get_use_speed_slow : m
mark:
    speed : current navigation speed
    second : the time to stay when encountering an obstacle
    p : whether the global path considers temporary barriers 1: true consider; 0: false
    width:fixed route obstacle avoidance width
    state:nearby parking status
    m:whether to adjust the minimum speed according to the curvature of the route
```

34. write navigation speed

```
request: write_max_vel[0.6]
response:
            get max vel:0.6 : where 0.6 is the current maximum speed
Mark:
      Range: [0.3,1.0]
      Used to adjust the maximum speed of navigation
```

```
response:
        get_max_vel:0.6 : where 0.6 is the current maximum speed
mark:
        Range: [0.3,1.0]
        Used to adjust the maximum speed of navigation
```

35. Set the dwell time when encountering obstacles

```
request: set_stop_time[5]
response:
    get_stop_time:5.0
mark:
    Range: [1,10]
```

36. Set whether the global path considers temporary obstacles

```
request: set_globalcost_p[1]
response:
    get_global_p:1.0
mark:
    1 : consider temporary barriers;
    0 : do not consider temporary barriers;
```

37. Automatically report coordinate control

```
request:nav:get_pose[open?on:off]
response:
      nav:pose[x,y,radian] current robot coordinates
      nav:pose:notfound The robot is positioning
mark:
    open: on: open; off: close
      x, y, radian are floating point numbers with two decimal places, where x, y are
  coordinates, radian is radian
```

38. Set the position and orientation of other robots on the local map

```
request: robot_cost[x,y,z,line_speed,hostname,radius]
mark:
    x,y : coordinates;
```

```
    z : radians;
    line_speed : line speed;
    hostname: machine name;
    radius : expansion radius (it is recommended to set 0.03 when the machine is stationary, and 0.04 when the machine is moving);
```

39. Set CPU high performance mode

```
request: cpu_performance
```

40. Set the nearest stop

```
request: set_tolerance[1/0]
mark:
    1: Enable nearby docking;
    2: Close the nearest stop;
```

41. Get the special area where the robot is currently located

```
request: sendWeb[in_polygon]
response: in_polygon:special_area_name
mark:
    When in a special area in_polygon:special_area_name;
    When not in the special area in_polygon:;
```

42. Take a temporary stop

```
request: get_nearest
response: answer_nearest{nofind/x,y,radian}
mark:
        nofind: cannot find the stop point;
        x,y are coordinates, radian is radians;
```

43. Get the ROS host startup time

```
request: get_poweron_time
response: power_on_t:yyyy-MM-dd HH:mm:ss
```

44. Filter Laser & 3D Noise Range

```
request: lidar_min[width]
        mark:
             width: width, floating point type, unit: meter;
```

45. Set whether the robot rotates (when the path, target point, etc. are blocked)

```
request: avoid_obstacle[on/off]
      mark:
             on: rotate;
             off: no rotation;
```

46. Set the robot length and width

```
request: footprint[width,length]
    mark:
         width: width value, floating point type, unit: meter;
         length: length value, floating point type, unit: meter;
         Default value: footprint[0,0];
```

47. Set the robot length and width(3.2.6)

```
request: footprint[front,left,back,right]
mark:
      front,left,back,right: the distance from the center of the machine to the front, left, back, right, floating point type, unit: meter;
```

48. Set the robot radius

```
request: robot_radius[radius]
    mark:
        radius: radius value, floating point type, unit: meter;
        default value: robot_radius[0]
```

49. Set the direction and distance when the robot moves away from the QR code

```
request: agv_move[distance]
    mark:
        distance:
            0: the machine leaves directly;
            positive number: the machine starts navigating after moving forward a specified distance, floating point type, unit: meter;
            negative number: the machine starts navigating after moving backward a specified distance, floating point type, unit: meter;
            default value: 0.7;
```

50. Set the direction of the machine when navigating with a QR code

```
request: agv_mode[direction]
    mark:
        0: The machine performs QR code navigation in reverse;
        1: The machine performs QR code navigation in forward direction;
        Default value: 0;
```

51. Fixed route navigation

```
request: points_path[target_point_name]
response:
        Refer to `ROS Active Report - 2. Navigation Status`
        mark:
              target_point_name:target point;
```

52. Cancel tag code navigation

```
request: agv:stop
mark:cancel tag code navigation
```

53. Query fixed route path

```
request: get_defined_plan[point]
response: getplan_dij:nofind
            getplan_dij:path_point1 path_point2 path_point3...
mark:
      point: navigation target point
      nofind: unable to generate route
      path_point: route point
```

54. Fixed waypoint navigation

```
request: list_point[1.12,2.13,3.11,4.13,5.11,2.01]
list_point_pre[1.12,2.13,3.11,4.13,5.11,2.01]
mark: Combine the path points into a fixed route and navigate along the route. The host computer sends data through the serial port. Since the maximum length of the data is 256,
the coordinates of a point (x, y, theta) occupy at least 15 bits
Each data retains two decimal places (retains precision)
Therefore, a maximum of 14 coordinate points are transmitted at a time. The first point is the current coordinate, and the last point is the target point coordinate.
Ensure that the path does not pass through obstacles or sharp turns at small angles. The distance between path points can be larger in straight line space and smaller in complex environments,the distance is a little smaller.
Before sending, please calculate the length of the string, please do not exceed 230
If the route is long, more than 14 coordinate points, it needs to be sent in packets. The last packet uses the list_point command, and the previous routes use the list_point_pre command
```

55. Actively issue speed control

```
request: app_vel[0.5,0.3]
mark: The first digit is the linear velocity, and the second digit is the angular velocity, both of which have positive and negative values. Positive numbers represent forward or left turn, and negative numbers represent backward or right turn.
      Please ensure that the frequency of this interface is around 10hz
      When starting debugging, please start from 0.1 and gradually increase to the required test value, otherwise the speed may be too high and affect safety.
```

56. Set the fixed route obstacle avoidance width

```
request: max_plan_dist[width]
mark:  width: unit:  meter
```

57. Generate a route to a specified point

```
request:get_plan_name[point]
get_plan_point[x,y,radian]
response:get_plan:error
get_plan:x1,y1,radian1,x2,y2,radian2...
get_plan1:x1,y1,radian1,x2,y2,radian2...
mark: get_plan_name[point] This interface can only transmit point names that are not in fixed route mode;
        get_plan:error Failed to generate the route, usually because there are obstacles on the target point or path;
        get_plan:x1,y1,x2,y2,... get_plan: is followed by the route coordinates
        When the route is long, it will be reported in packets. The first packet starts with get_plan:, and the subsequent data packets start with get_plan1:.
        If the route is not reported, it ends with +;
```

58. Set whether the depth camera participates in dynamic obstacle avoidance (RSNF1 3.2.3)

```
request: pc_switch[switch]
mark: switch is 0 to turn off the depth camera obstacle avoidance, and 1 to turn on the depth camera obstacle avoidance
```

59. Set whether to adjust the minimum speed according to the curvature of the route (RSNF1 3.2.4)

```
request: use_slow[switch]
mark: switch If 1, use the dynamic minimum speed; if 0, do not use the dynamic minimum speed
```

60. Set the move to target point (RSNF1 3.2.5)

```
request: move_goal[x,y,yaw]
response: move:done:15
mark: The parameter is the coordinate of the target point. Please keep two decimal places. Move forward at a speed of 0.1 and rotate at a speed of 0.1. Notify when reaching the target point
If there is an obstacle 10cm ahead, it will be considered to have arrived
```

61. Set whether local path obstacle avoidance takes map layer into account (RSNF1 3.2.5)

```
request: local_static[on]
mark: The parameter is on or off, on means using the map, off means not using the map
Considering the map layer will be safer, but correspondingly, due to the change in wall thickness, the width requirement is larger
```

62. Start docking QR code

```
request: agv:start[point_name]
mark: point_name: the point name of the QR code to be docked;
```

63. Query robot type

```
request: robot_type
response: robot_type:type
mark: type:
      1: Hussar chassis
      2: Moonknight chassis
      3: How wheels chassis
      4: Flybot with label code
      5: Big dog
      6: Flybot without label code
      7: Flybot (double lidar)
      8: Big dog (double lidar)
      Snail sweeper and Forklift determine the robot type by ros version number
```

64. ROS wifi switch

```
request: robot_type
response: robot_type:type
mark: type:
      1: Hussar chassis
      2: Moonknight chassis
      3: How wheels chassis
      4: Flybot with label code
      5: Big dog
      6: Flybot without label code
      7: Flybot (double lidar)
      8: Big dog (double lidar)
      Snail sweeper and Forklift determine the robot type by ros version number
```

65. Set anti-fall parameters

```
request: cliff_params[height,range,num,frequency]
mark: height: anti-fall height
        range: anti-fall distance
        num: the number of points that meet the anti-fall triggering requirement
        frequency: filter anti-fall frequency
```

Unique to Snail Sweeper

1. Set anti-fall parameters

```
request: clean_room[start:areaName]
      mark:
            areaName: clean area name
response:
      clean_room[start:code,clean_id]
      mark:
             code:
                   -1: failed, room name not found;
                   -2: failed, unable to generate any route, starting point has obstacles or area is less than 1 square;
             -3: failed, room not found, at least three points;
                   -4: failed, calculating route, unable to execute
```

Unique to Forklift

1. Take the pallet (adjust the lateral position deviation)

```
request: pallet:start[x,y,radians]
mark: x,y,radians are the coordinates of the pallet point
```

2. Place the pallet (adjust the lateral position deviation)

```
request: unload:start[x,y,radians]
mark: x,y,radians are the coordinates of the pallet point
```

3. Entering the elevator (adjusting the lateral position deviation before entering the elevator)

```
request: elevator_in:start[x,y,radians]
mark: x,y,radians are the coordinate of the elevator inside point
```

4. Exit the elevator

```
request: elevator_out:start[x,y,radians]
mark: x,y,radians are the coordinate of the elevator out point
```

5. Control Forklift arms

```
request: forklift_arm[1/0]
mark: 1: raise the forklift arm; 0: lower the forklift arm;
```

6. Exit the pallet docking state

```
request: pallet:stop
```

7. Set obstacle avoidance distance

```
request: avoid:distance[0.3,0.3,0.2,0.2]
mark: The parameters are front, back, left, and right;
```

8. Set obstacle avoidance distance

```
request: goal_back_point[x,y,theta]
response: same as normal navigation
mark: The forklift will not adjust the angle when reversing, so need to turn the rear end of the vehicle toward the goal point first;
```

9. Setting the laser angle at the forklift arm

```
request: set_arm_laser[angle]
mark:angle range:[20,160]
```

10. Detect whether there are obstacles at the target point

```
request:check:area_reachable[positionCenter[0],positionCenter[1],positionCenter[2],distanceToFront,distanceToBack,distanceToLeft,distanceToRight]
response: area_reachable:x
mark: positionCenter is the coordinate of the target point;
        distanceToXX is the distance to determine whether there is an obstacle within a certain range of the target point;
        area_reachable:x: x is 1, indicating no obstacle, and 0 indicates an obstacle;
```

11. Detect whether the robot can rotate safely

```
request: check:turn_angle[angle]
response: turn_angle_check:x
mark: angle is the angle that the robot will rotate (left rotation is positive, right rotation is negative), for example -90 means right rotation 90°;
turn_angle_check:x When x is 1, it means that the vehicle can rotate, and 0 means that there is an obstacle and the robot cannot rotate;
```

Forklift pallet placement process

```
After reaching the designated position, take the pallet
android send to ros -> pallet:start[x,y,radians]
Start adjusting the lateral deviation                forklift{1 0 1 0 0} <- ros send to android
The lateral deviation adjustment is completed        forklift{1 0 3 0 0} <- ros send to android
Android sends to lower the forklift arm              android send to ros -> forklift_arm[0]
After lowering the forklift arm, start docking the pallet
forklift{0 0 3 0 0} <- ros send to android
The docking of the pallet is completed               forklift{2 0 2 0 0} <- ros send to android
Raise the forklift arm                               android send to ros -> forklift_arm[1]
The forklift arm has been raised                     forklift{1 0 0 0 0} <- ros send to android
Leave the pallet position                            android send to ros -> move[1400,0]
Move completed                                       move:done:16 <- ros send to android
mark:move[1400,0] means moving forward 1.4m. This is for reference only. The actual distance between the pallet point and the pallet docking point should be calculated.
```

Transparent command

```
request: send_to_base[data]
response: base_data:{}
mark: navigation forwarding, sent to the power board
```

1. Get the name of the available power supply for the power supply board

android request power information

index	content	Description
0	command	0x0D
1	Length	data_cmd+data
2	data_cmd	0x01
3	data	0x00 reserved bit

```
request: send_to_base[208 2 1 0]
```

Power board feedback power supply name

Index	content	Description
0	command	0x0D
1	Length	data_cmd+data1...
2	data_cmd	0x01
3	data1	Power Name 1 (low -> high) format: string

4	data2	Power Name 2 (low - > high) format: string

```
response:
base_data:{ D0 1F 01 32 34 76 00 62 61 74 74 65 72 79 5F 31 00 64 6F 6F 72 5F 6C 6F 63 6B 5F 70 6F 77 65 72 00 }
mark:
Power 1 : 32 34 76 00
- > 50 52 118 0 (convert to decimal, used when sending switching power supply)
- > 24v (converted to a string)
Power supply 2: 62 61 74 74 65 72 79 5F 31 00
Power supply 3: 64 6F 6F 72 5F 6C 6F 63 6B 5F 70 6F 77 65 72 00
```

2. switching power supply

Index	content	Description
0	command	0x0D
1	Length	data_cmd+data1...
2	data_cmd	0x02
3	data1	enable bit 0x00 power off 0x01 Turn on the power
4	data2	power supply name (low - > high) format: string

```
request:send_to_base[208 6 2 1 50 52 118 0]
mark:
Turn on the power supply 24v
```

36V Power Control

```
request:
        Open: The public power board needs to send the command first: send_to_base[208 12 02 01 98 97 116 116 101 114 121 95 49 00]
            send_to_base[208 16 02 01 103 101 110 101 114 97 108 95 112 111 119 101 114 00]
        Close: send_to_base[208 16 02 00 103 101 110 101 114 97 108 95 112 111 119 101 114 00]
```

24V Power Control

```
request:
     Open: send_to_base[208 06 02 01 50 52 118 00]
     Close: send_to_base[208 06 02 00 50 52 118 00]
```

Power Board Upgrade

1. Upload upgrade file

```
The web side will upload the upgrade file;
```

2. enter loading mode

```
request: base_upgrade[start]
response: base_upgrade:loading
```

3. upgrade

```
request: base_upgrade[affirm]
response: base_upgrade: success
```

test odom

```
roslaunch yoyo_node test_get_odom.launch
rostopic echo /mobile_base/debug/odom_pose2d
```

test imu

```
roslaunch yoyo_node test_get_yaw.launch
rostopic echo /mobile_base/debug/imu_angle
```