RobotNavigationSDK Integration Guide

1 Environment Requirements

- Operating System: Android 8.1
- Development Software: Android Studio 4.2+
- JDK: 1.8
- SDK Version: Compilation and Target SDK version 27

2 RobotNavigationSDK Functionality

RobotNavigationSDK provides functionality for integration with navigation features. By invoking the interfaces provided by the SDK, you can control robot navigation functions. The SDK offers interfaces for navigation, cruising, positioning, map switching, and more. Map creation and editing functions are performed within the robot navigation component.

Note: RobotNavigationSDK can only be used with robots that have navigation capabilities.

3 RobotNavigationSDK Integration Steps

3.1 Install or Update Serial Port and Navigation Components

Open "System Settings" and click on the "Update Applications and Components" option. In the opened page, check if the "Serial Port Component" and "Navigation Component" are up to date. If there are updates available for these components, update them to the latest version.

3.2 Obtain the JAR Package

Before integrating the JAR package, you need to obtain the RobotNavigationSDK-*.jar. You can find the JAR package in the "libs" directory of the provided compressed package.

3.3 Integrate the JAR Package

Place RobotNavigationSDK-*.jar in the "libs" directory of your project, and add the configuration to your project's build.gradle file as follows:

```
dependencies {
  //...
  api files('libs/RobotNavigationSDK-1.3.0.jar')
}
```

When calling SDK interfaces, many data returns are asynchronous. In RobotNavigationSDK, robot navigation information is returned with the help of EventBus. Therefore, you also need to integrate dependencies such as EventBus into your project. The dependencies to be added to your project's build.gradle file are as follows:

```
dependencies {
  //...
  api 'org.greenrobot:eventbus:3.1.1'
  api 'com.google.code.gson:gson:2.9.0'
}
```

3.4 Connect to Navigation

Before calling navigation-related interfaces, you need to use a PadBotNavigationClient instance to connect to navigation, as shown below:

```
// Connect to navigation
PadBotNavigationClient.getInstance().connect(getApplicationContext());
```

3.5 Initialize Navigation

After successfully connecting to navigation, the OnNavigationClientConnectedEvent event is triggered via EventBus. In the method that listens to the OnNavigationClientConnectedEvent event, initialize navigation as shown below:

```
/**
 * Receive navigation connection success event
 * @param event Navigation connection success event
 */
@Subscribe(threadMode = ThreadMode.MAIN)
public void onEvent(OnNavigationClientConnectedEvent event) {
    try {
        // Initialize navigation
        PadBotNavigationClient.getInstance().initNavigation();

    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

3.6 Create a Map and Add Target Points

Before controlling robot navigation, you need to create a new map and add target points to it. These operations are performed within the navigation component. You can open the navigation component using the following code:

```
// Open the navigation component
PadBotNavigationClient.getInstance().startNavigationApp(Activity);
```

3.7 Control Robot Navigation

Before instructing the robot to move to a specific target point, you need to obtain map information, extract information about the target point from the map, and then call the navigation interface to move to the target point.

1、First, obtain a list of maps:

```
IndoorMapVoListResult indoorMapVoListResult =
PadBotNavigationClient.getInstance().getMapInfo();
if (null != indoorMapVoListResult) {
    // Get the list of maps
    List<IndoorMapVo> indoorMapVoList = indoorMapVoListResult.getMapVoList();
}
```

2、Find the map currently used by the robot from the list of maps:

```
IndoorMapVo currentIndoorMap = null;
// Traverse the list of maps to find the currently used map
for (IndoorMapVo indoorMapVo : indoorMapVoList) {
    if (indoorMapVo.isDefaultUse()) {
        currentIndoorMap = indoorMapVo;
        break;
    }
}
```

3、Retrieve information about the list of target points within the map:

```
// Get the list of target points
List<RobotTargetPointVo> targetPointVoList = currentIndoorMap.getTargetPointVoList();
```

4、Get a specific target point and call the navigation interface to move to that point:

```
RobotTargetPointVo targetPointVo = targetPointVoList.get(0);
// Navigate to the target point
PadBotNavigationClient.getInstance().startNavigateByPointId(targetPointVo.getId());
```

3.8 Receive Robot Information

1、First, you need to register EventBus so that you can receive events thrown by EventBus, as shown below:

```
// Register EventBus
EventBusUtils.register(this);
```

2、Then, add EventBus annotations to your methods to receive specific event objects. For example:

```
/**
 * Receive navigation information event
 * @param event Navigation information event object
 */
@Subscribe(threadMode = ThreadMode.MAIN)
public void onEvent(OnNavigateInfoChangedEvent event) {
    try {
        NavigateInfo navigateInfo = event.getNavigateInfo();
        NavigateStatus navigateStatus = navigateInfo.getNavigateStatus();
        RobotTargetPointVo targetPointVo = navigateInfo.getTargetPoint();
        NavigateStepType stepType = navigateInfo.getNavigateStepType();
        ActionStatus actionStatus =
PadBotNavigationClient.getInstance().getActionStatus();

        // If the current action is navigating to a target point
        if (actionStatus == ActionStatus.NAVIGATING) {
            // Start navigating to a new target point
            if (NavigateStatus.NEW_TARGET == navigateStatus) {
                if (null != targetPointVo) {
                    showToast("Navigate to:" + targetPointVo.getName());
                }
            }
            // If the final target point is reached
            else if (NavigateStatus.FINISHED == navigateStatus) {
                if (null != targetPointVo) {
                    showToast("Arrived:" + targetPointVo.getName());
                }
            }
         }
    } catch (RemoteException e) {
        e.printStackTrace();
        }
}
```