Version 1.0
Release Date: April 3rd, 2023

4th Kibo Robot Programming Challenge

Programming Manual

Version 1.0 (Release Date: April 3rd, 2023)

Japan Aerospace Exploration Agency (JAXA)

General Point of Contact: Z-KRPC@ml.jaxa.jp

List of Changes

All changes to paragraphs, tables, and figures in this document are shown below;

Release Date	Revision	Paragraph(s)	Rationale
April 3rd, 2023	1.0	All	-

Contents

1. Introduction

Let's start programming!

Astrobee can be controlled with an Android application called the Guest Science APK (GS APK). First, setup your machine to build your application according to the instructions in Chapter 2. Next, read Chapter 3 and create a GS APK. Then, try running the GS APK in the simulator environment. Chapter 4 describes how to use the simulator environment.

2. Setting up your machine

First of all, set up a machine for programming.

2.1. Requirements

The machine must meet the following requirements.

- 64-bit processor
- 4 GB RAM (8 GB RAM recommended)
- Ubuntu 20.04 (64-bit version) (http://releases.ubuntu.com/20.04/)
  or Windows 10 (64-bit version)

NOTE: If you want to run your program on your own PC, you need 16 GB of RAMand Ubuntu 20.04. For details, please refer to 4.6. Running on your own machine (optional).

2.2. Setting up Android Studio

2.2.1. Installing components (Only on Ubuntu)

If you use an Ubuntu machine, you need these components.

- openJDK8
- ADB (Android Debug Bridge)
- Gradle

Please install them with the following command.

```
sudo apt-get -y install openjdk-8-jdk adb gradle
```

2.2.2. Installing Android Studio

Please download Android Studio 3.4.1 from the Android Studio download archives page (https://developer.android.com/studio/archive) and extract it to your home directory.

2.2.3. Downloading additional components

To build the Guest Science APK, you need to download additional components as follows.

1) Launch Android Studio.
2) Select [Tools] -> [SDK Manager].
3) On the SDK Platforms tab, check “Show Package Details” and select “Android SDK Platform 25” and “Android SDK Platform 26.”
4) On the SDK Tools Tab, check “Show Package Details” and select “25.0.3”, “26.0.2” under Android SDK Build-Tools and “Android SDK Platform-Tools.”
5) Click the [Apply] button to install these components.

3. Creating your application

3.1. Creating an Android project

To create your application, prepare a new project with the following steps.

1) Download the APK template (Template APK) from the download page on the web site.
2) Extract the zip file to the directory where you want it.
3) Launch Android Studio.
4) Open the APK template folder with [File] -> [Open].
5) Open [app/java/jp.jaxa.iss.kibo.rpc.defaultapk /YourService.java] in Project view.
6) Write your code in runPlan1 – runPlan3 methods in the YourService.java file.

When you open the APK template folder, the "Android Gradle Plugin Update Recommended" dialog may appear. However, you must not update the plugin because of a dependency problem, so select "Don't remind me again for this project."

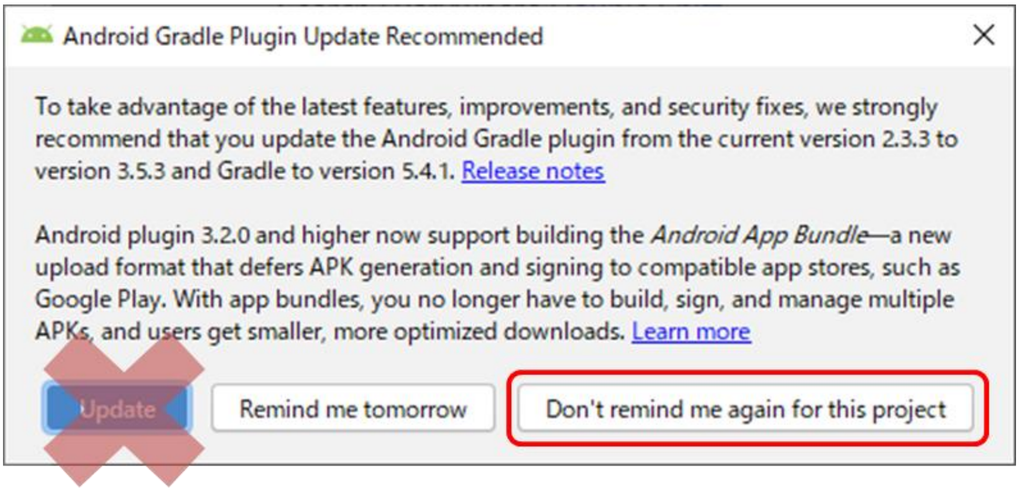


Figure 3-1 Android Gradle Plugin Update Recommended dialog

3.2. Writing the application

You can use the game APIs shown below in the YourService.java file.

"**runPlan1**" is executed on the web simulator. You can choose any plan when you run the application on your own machine.

```
public class YourService extends KiboRpcService {
    // write your plan 1 here
    @Override
    protected void runPlan1(){
        // the mission starts
        api.startMission();
        int loop_counter = 0;

        while (true){
            // get the list of active target id
            List<Integer> list = api.getActiveTargets();

            // move to a point
            Point point = new Point(10.4d, -10.1d, 4.47d);
            Quaternion quaternion = new Quaternion(0f, 0f, 0f, 1f);
            api.moveTo(point, quaternion, false);

            // get a camera image
            Mat image = api.getMatNavCam();

            // irradiate the laser
            api.laserControl(true);

            // take active target snapshots
            int target_id = 1;
            api.takeTargetSnapshot(target_id);

            /* ************************************************ */
            /* write your own code and repair the ammonia leak! */
            /* ************************************************ */

            // get remaining active time and mission time
            List<Long> timeRemaining = api.getTimeRemaining();

            // check the remaining milliseconds of mission time
            if (timeRemaining.get(1) < 60000){
```

```
                break;
            }

            loop_counter++;
            if (loop_counter == 2){
                break;
            }
        }
        // turn on the front flash light
        api.flashlightControlFront(0.05f);

        // get QR code content
        String mQrContent = yourMethod();

        // turn off the front flash light
        api.flashlightControlFront(0.00f);

        // notify that astrobee is heading to the goal
        api.notifyGoingToGoal();

        /* ********************************************************** */
        /* write your own code to move Astrobee to the goal positiion */
        /* ********************************************************** */

        // send mission completion
        api.reportMissionCompletion(mQrContent);
    }

    protected void runPlan2() {
        // You can write your other plan here, but it's not run on  the web simulator.
        // …
    }
}
```

You can find methods of the game APIs by using the code completion function of Android Studio.

Please refer to 7. Game API details for more information and you can download a sample APK from the download page on the web site.

3.3. Building your application

3.3.1. On Ubuntu

To build your application, use the command shown below.

NOTE: DO NOT build your application using Android Studio to change the build task, as this may cause an error.

```
$ cd <YOUR_APK_PATH>
$ ANDROID_HOME=$HOME/Android/Sdk ./gradlew assembleDebug
```

You can find the APK file here: “<YOUR_APK_PATH>/app/build/outputs/apk/app-debug.apk”.

3.3.2. On Windows

Please build your application with the following steps.

1) Launch the Android Studio.
2) Open <YOUR_APK_PATH>.
3) Click app on the [Project] window.
4) Select [Build] -> [Make Module ‘app’].

If you find errors, please build an APK file on an Ubuntu machine.

You can find the APK file here: “<YOUR_APK_PATH>\app\build\outputs\apk\app-debug.apk”.

3.3.3. Change the application ID, APK name and short name (optional)

You can change the application ID (*jp.jaxa.iss.kibo.rpc.sampleapk* or *jp.jaxa.iss.kibo.rpc.defaultapk* by default).

In this step, we change the application ID to “jp.jaxa.iss.kibo.rpc.<u>myteam</u>” and the APK name/short name to “myteam” with the SampleAPK project.

NOTE: This instruction is for the Final Round. Changing the application ID is not necessary and not recommended in the Preliminary Round.

NOTE: “jp.jaxa.iss.kibo.rpc” cannot be changed.

1) Launch Android Studio.
2) Open <YOUR_APK_PATH>.
3) Make sure you are viewing the project in Android View.

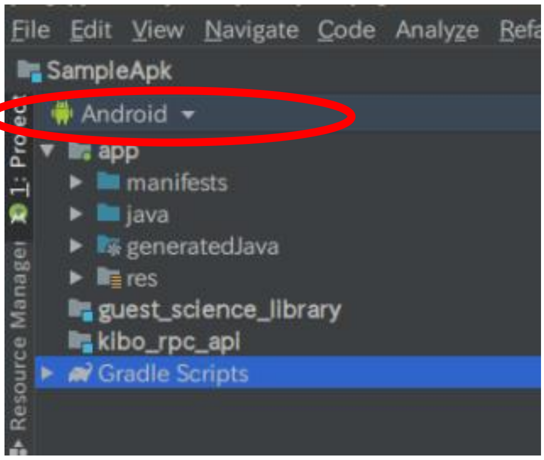


Figure 3-2 Android View

4) Click on settings (the gear icon) and deselect [Compact Middle Package].

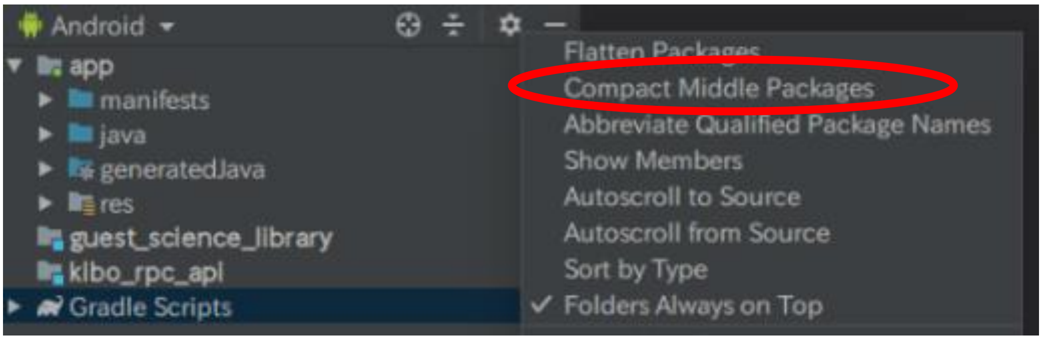


Figure 3-3 Unselect [Compact Middle Package]

5) Please expand the "java" folder.

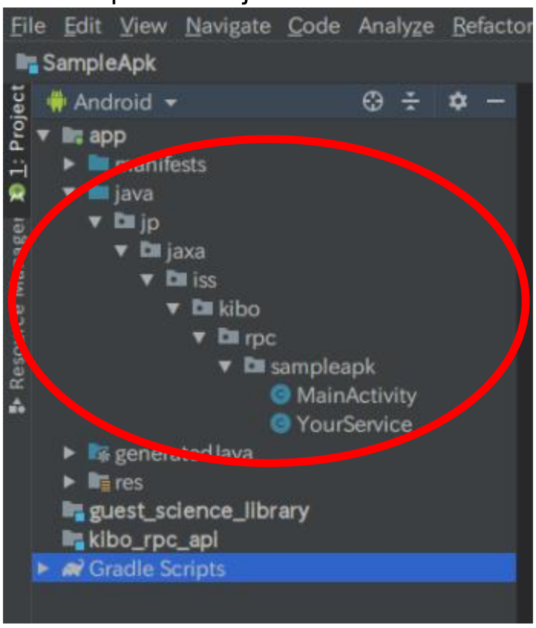


Figure 3-4 Expand the "java" folder

6) Right-click the "sampleapk" folder and select [refactor] -> [rename].

7) A warning will be displayed, but you want to go ahead and click [Rename Package]. After that, enter theAPKname that you want. (In the picture, we rename as it “myteam”.)

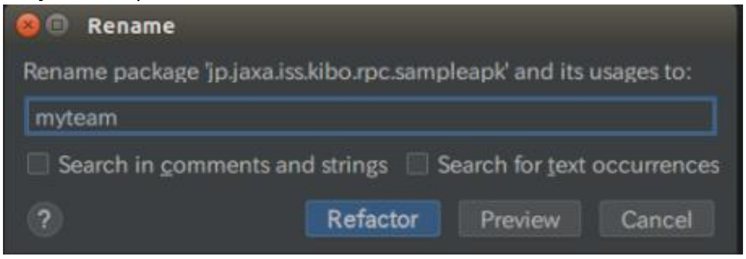


Figure 3-5 Rename dialog

8) At the bottom of Android Studio, “Refactoring Preview” will be displayed. Here, click [Do Refactor].

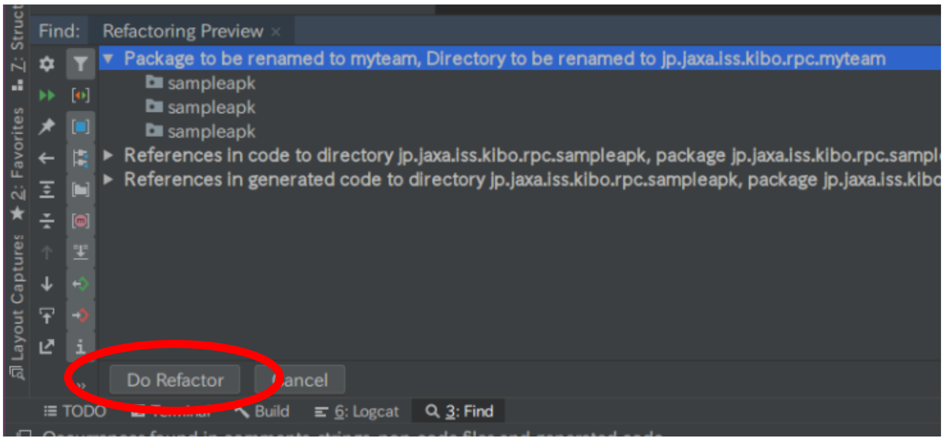


Figure 3-6 Refactoring Preview

9) Open **build.gradle (Module: app)** in Gradle Scripts on the left-side of the menu. Please change the application ID and click [Sync Now].

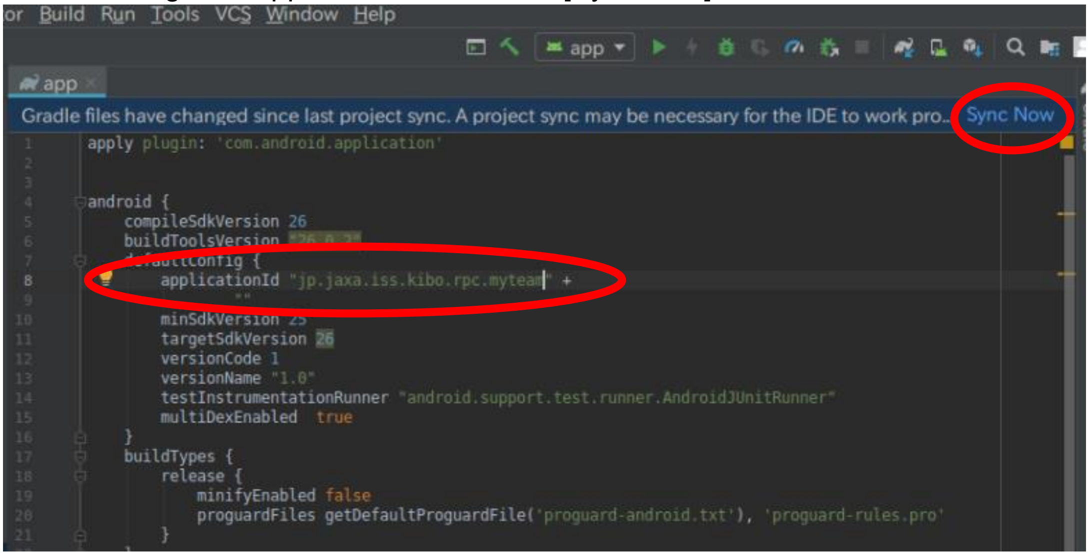


Figure 3-7 build.gradle (Module: app)

10) Open **strings.xml** in app -> src -> main -> res -> values on the left-side of the menu. Please change the APK name and save it.

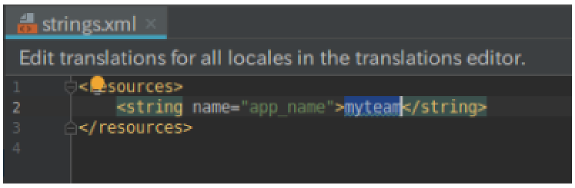


Figure 3-8 strings.xml

11) Open **commands.xml** in app -> src-> main -> res -> xml on the left-side of the menu. Please change the value of “shortName” tag and save it. If there is no “shortName” tag, please add it under the “apkInfo” tag and set your short name.

```
<shortName>myteam</shortName>
```

Figure 3-9 commands.xml

You have successfully changed the application ID in Android Studio. If you want to change the Android project name and its directory name, follow the following steps.

(1) On Ubuntu

12) Close Android Studio.

13) Please execute the following commands.

```
cd <YOUR_APK_PATH>
cd ../
mv SampleApk <YOUR_APK_NAME>
cd <YOUR_APK_NAME>
mv SampleApk.iml <YOUR_APK_NAME>.iml
```

(2) On Windows

12) Close the Android Studio.

13) Please rename a **SampleApk** folder to <YOUR_APK_NAME> with Windows Explorer.

14) Now rename **SampleApk.iml** to <YOUR_APK_NAME>.iml in the **SampleApk** folder with Windows Explorer.

4. Running your program on the simulator

4.1. Using the simulator server

Once you have built your application, you can run it on the web simulator provided by JAXA. To use the simulator, you need a user account issued by the Kibo-RPC secretariat. If you don't have one, please read the Kibo-RPC Guidebook to complete your application for participating in Kibo-RPC first.

Note that the actual Kibo environment is not exactly the same as the simulation environment, since there are many objects in the Kibo and the environment changes frequently. Please refer to the image and Google Map for the actual environment. (These are not the latest.)

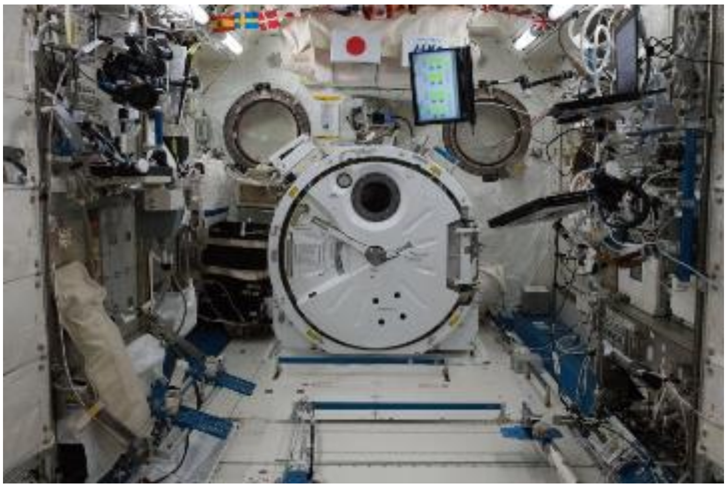


Figure 4-1 Image inside Kibo

Google Map: https://www.google.com/maps/@29.5604024,-95.0855631,2a,75y,205.79h,103.61t/data=!3m6!1e1!3m4!1sUA46_vIbk9kAAAQvxgbyMg!2e0!7i10000!8i5000

Also, the veiws of Astrobee's NavCam and DockCam are different between the simulator and the real robot. Refer to the image below.

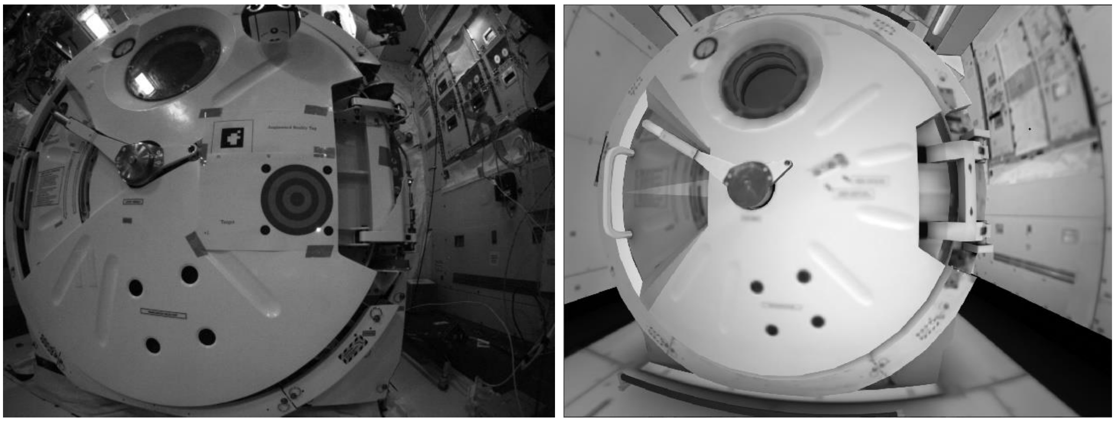


Figure 4-2 (Left) Actual snapshot (Right) Simulator image

4.2. Login

Access the Kibo-RPC web site (https://jaxa.krpc.jp/) and click “LOGIN.”

Figure 4-3 LOGIN tab

On the login page, enter the ID and password for your team’s account, and click the “LOGIN” button. If you have forgotten your ID, please contact the Kibo-RPC secretariat. You can reset your password by clicking the “Forgot your Password?” link.

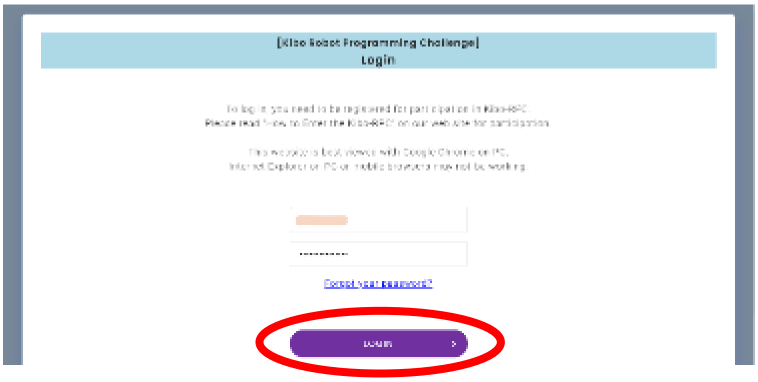


Figure 4-4 LOGIN button

On the home page, click “SIMULATION.”

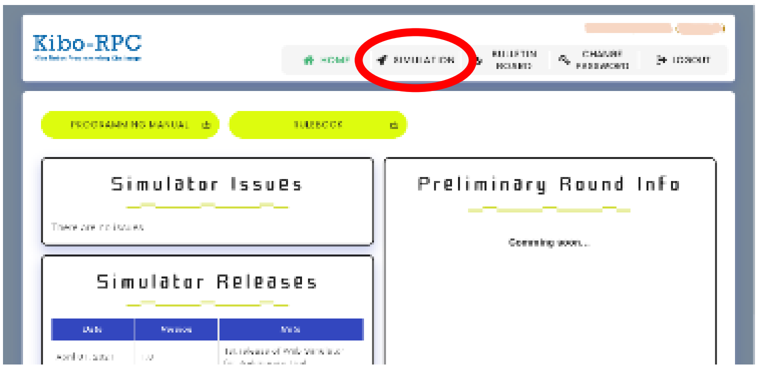


Figure 4-5 SIMULATION button

Now, you can access the web simulator from this page.

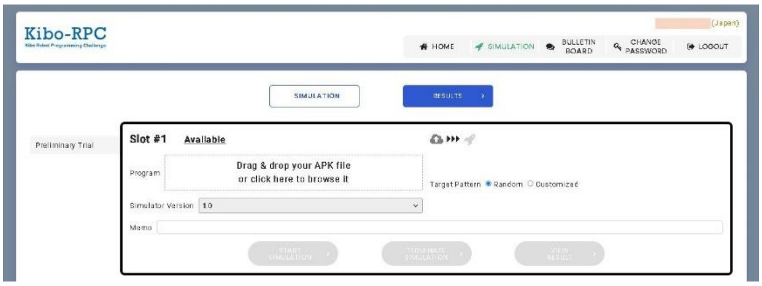


Figure 4-6 Web simulator page

4.3. Uploading the APK and running your program

On the simulation page, there are three slots for simulations, so that you can run at most three programs in parallel.

To start your simulation, select your APK file, the simulator version and the simulation conditions.

“Target Pattern” can be selected from “Random” and “Customized”. If you select Customized, you can specify which targets to activate for each phase.

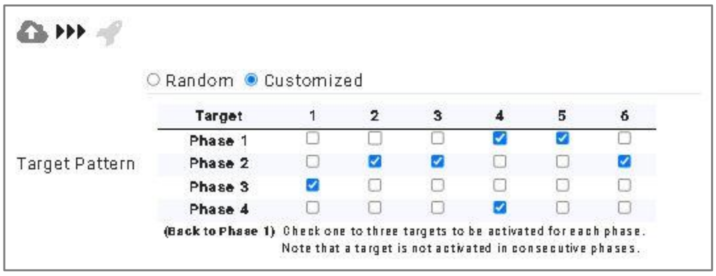


Figure 4-7 Customize target patterns

Eter a memo if desired, and click the “START SIMULATION” button.

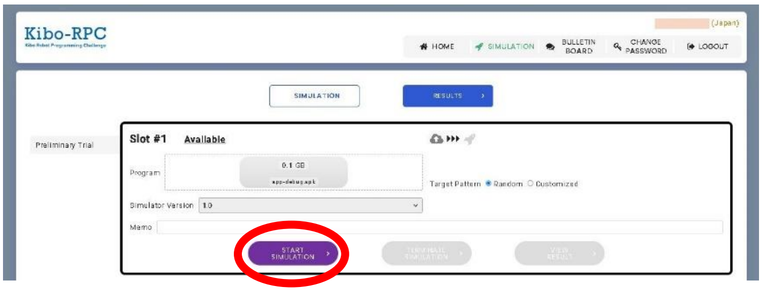


Figure 4-8 START SIMULATION button

A simulation may take longer than 20 minutes to run, and it does not need your attention while it runs. After starting your simulation, you can log out, get a cup of coffee, then go back to the web site.

When there is a simulation running, the slot displays its original information, and you cannot run another simulation in the same slot until it finishes.

If you want to stop your simulation, click the “TERMINATE SIMULATION” button. Note that terminating a simulation loses its game score and output files (such as rosbag and the Android Emulator’s log).

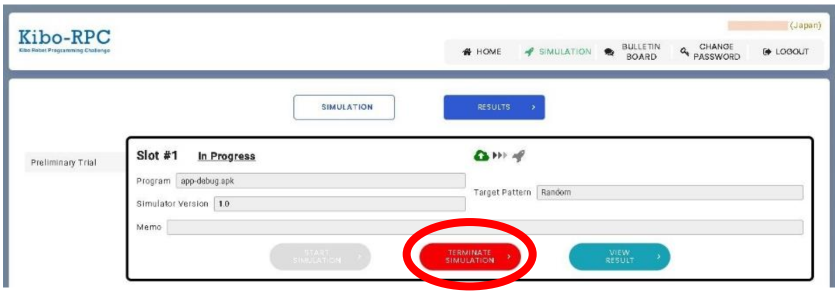


Figure 4-9 TERMINATE SIMULATION button

4.4. Checking simulation while running

When your simulation is running, you can log in to the simulator server (viewer) via your browser. Click the “SIMULATIOR VIEWER” button to show the information for a remote connection and open the viewer in another tab by clicking the “VIEW” button.

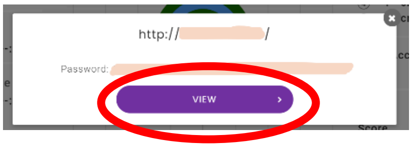


Figure 4-10 VIEW button

Enter the password for your remote connection to log in. Now you can use rviz to see how Astrobee moves in your simulation. This viewer is available until the simulation is finished.

The viewer displays a real-time simulation in the view-only mode for the simulation stability. You cannot operate the viewer.

4.5. Checking the result

4.5.1. Result summary

Once your simulation has started, you can check the results by clicking the “VIEW RESULT” button on the simulation page.

Figure 4-11 VIEW RESULT button

On the result page, you can see the details of your simulation, such as the game time, laser accuracy, and so on.

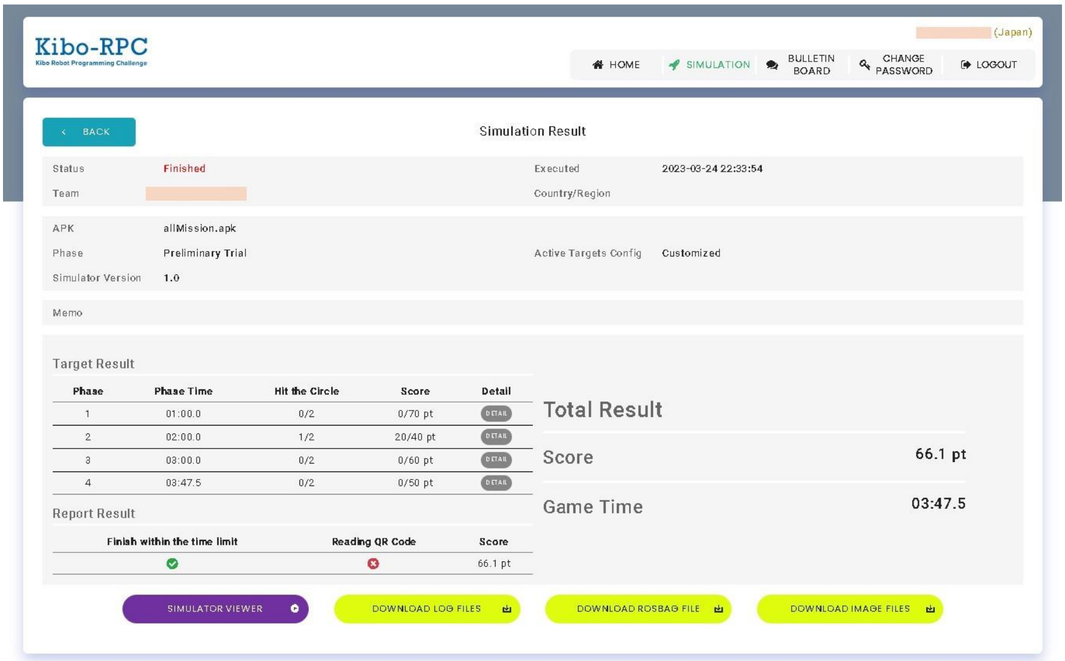


Figure 4-12 RESULT page

Table 4-1 Report Result

Finish within the time limit	Green check	Astrobee executed reportMissionCompletion API at the correct position within the time limit.
	Red cross	Astrobee executed reportMissionCompletion API at a wrong position. Or Astrobee did not try to execute reportMissionCompletion API.
Reading QR Code	Green check	A report message paired with QR code content is correct.
	Red cross	A report message is incorrect.

Refer to the rulebook for the defined report message.

Click “DETAIL” button, PHASE DETAIL” page is displayed.

Figure 4-13 PHASE DETAIL page

Table 4-2 Target Result

Hit the Target	Green check	When Astrobee took a snapshot, the laser shot intside of target's frame.
	Red cross	When Astrobee took a snapshot, the laser shot outside of target's frame. Or Astrobee did not take snapshot.

4.5.2. Download ZIP file

You can get a ZIP file by clicking the “DOWNLOAD LOG FILES” button. This ZIP file contains the game score and the Android Emulator’s log. The output log level is INFO. Note that some or all of these files will not be available unless your simulation finishes properly. Besides the result page, the game score also appears in a JSON file, which can be read using a text editor.

Table 4-3 Example of result.json

<table>
<tr><td><pre>
{
  "Mission Time": {
        "start": "20230315 203208003",
        "finish": "20230315 203656003"
  },
  "Phases": [
    {
      "phaseNumber": 1,
      "timestamp": "20230315 203223123",
      "activeTargets": [1, 2],
      "Targets": [
        {
            "targetId": 1,
            "direction": true,
            "x": 1.22,
            "y": -3.44,
            "r": 4.12,
            "timestamp": "20230315 203324123"
        },
        {
            "targetId": 2,
            "direction": true,
            "x": 1.22,
            "y": -3.44,
            "r": 4.12,
            "timestamp": "20230315 203425123"
        }
      ]
    },
    {
      "phaseNumber": 2,
      "timestamp": "20230315 203426123",
      "activeTargets": [3, 5],
      "Targets": [
        {
</pre></td>
<td>“Mission Time” is the difference between the “start” time and the “finish” time.

“direction” is true if the laser shot is on the Target plane.
“r” is the distance between the center of Target and the laser shot.</td></tr>
</table>

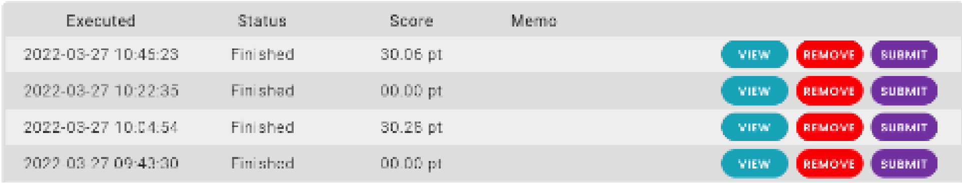
"targetId": 3,
"direction": true,
"x": 1.22,
"y": -3.44,
"r": 4.12,
"timestamp": "20230315 203527123"
},
{
"targetId": 5,
"direction": true,
"x": 1.22,
"y": -3.44,
"r": 4.12,
"timestamp": "20230315 203628123"
}
]
}
],
"Report": {
"arrival": {
"try": true,
"success": true,
"distance": 3.2
},
"qr": {
"try": true,
"success": true
}
},
"illegal": false
}
	"direction" is true if the laser shot is on the Target plane.
	"r" is the distance between the center of Target and the laser shot.
	
	"try" is true if you execute reportMissionCompletion API.
	"success" is true if you execute reportMissionCompletion API at the correct position.
	
	"success" is true if a report message paired with QR code content is correct.
	
	"illegal" is for server internal use.

You can also get a rosbag as a ZIP file by clicking the "DOWNLOAD ROSBAG FILE" button. The size of this file will be so large that it may take a long time to download it.

In addition, you can save any debug images with saveBitmapImage/saveMatImage API and download the images by clicking the "DOWNLOAD IMAGE FILES" button.

4.5.3. Check simulation after running

To check previous simulations, click the “Results” button on the simulation page. The results page lists your past simulations. This list can hold up to 20 simulations.



Executed	Status	Score	Memo			
2022-03-27 10:45:23	Finished	30.06 pt		VIEW	REMOVE	SUBMIT
2022-03-27 10:22:35	Finished	00.00 pt		VIEW	REMOVE	SUBMIT
2022-03-27 10:04:54	Finished	30.25 pt		VIEW	REMOVE	SUBMIT
2022-03-27 09:43:30	Finished	00.00 pt		VIEW	REMOVE	SUBMIT

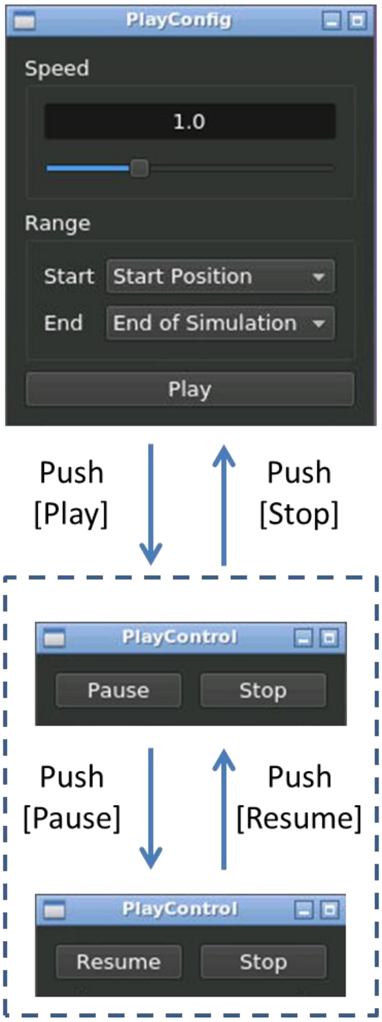
Figure 4-14 Results list page

The “VIEW RESULT” button is the same as the one on the simulation page. Please be careful when you click the “REMOVE RESULT” button; it removes the output files of the selected simulation and these results will be lost.

You can play the rosbag (simulation result) at 0.5x – 3x speed with the viewer. You can change rosbag replay settings and rviz settings. The details are, described in the sections below.

4.5.4. rosbag replay settings

You can change rosbag replay settings using Rosbag Player.



Type	Description
Speed Slider	Select replay speed.
Range Selector	Select replay range.
Play Button	Start replay and open rviz window. If rviz already has opened, it will restart.
Pause Button	Pause replay.
Resume Button	Resume replay.
Stop Button	Stop replay and back to PlayConfig window.

Figure 4-15 Rosbag Player

4.5.5. rviz settings

You can change the display settings for the rviz window.

Table 4-4 rviz configuration

Item	Check box in the "Displays" tab
Planning trajectory	[Visualize]->[PlanningTrajectory]
Trajectory	[Visualize]->[Trajectory]
KeepInZone/KeepOutZone	[Visualize]->[Zones]
NavCam	[Sensors]->[NavCam]
DockCam	[Sensors]->[DockCam]
HazCam	[Sensors]->[HazCam]

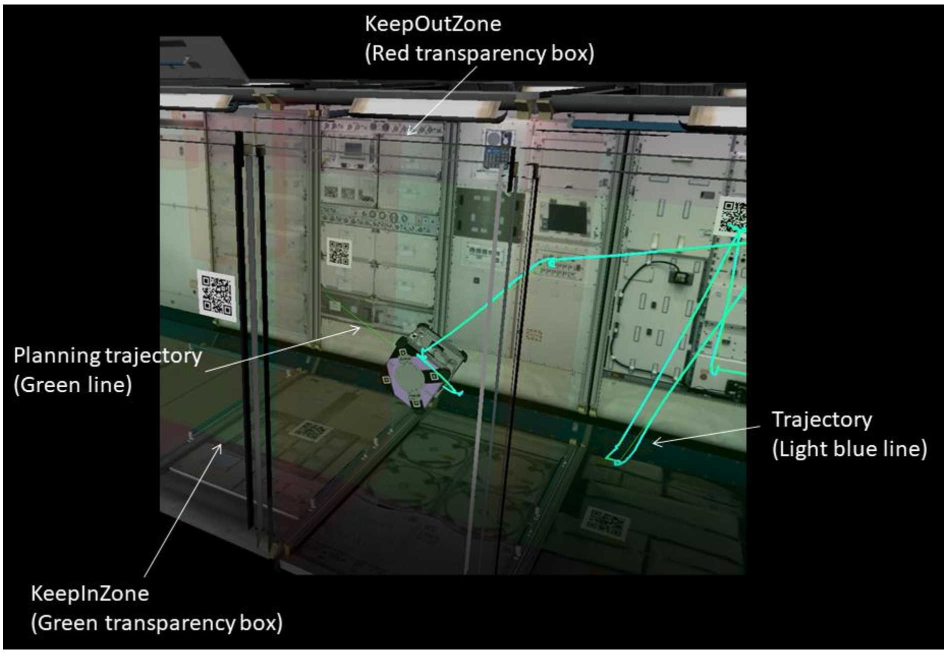


Figure 4-16 rviz configuration description

4.6. Running on your own machine (optional)

You can also run the program on your own machine. This chapter provides a procedure to set up the Astrobee Simulator. You get a simple simulation environment without randomness modules (Phases/targets randomness generator, air flow simulator and navigation errors) or a judge module.

4.6.1. Differences between web simulator and local simulator

Local Simulation Environment do not include random factor modules (object position, airflow, and navigation error).

You can test and debug your program using a local simulator, but you need to evaluate it on a web simulator server in order to obtain a high score in the Preliminary Round.

4.6.2. Requirements

The following requirements are needed to set up a simulation environment on your machine.

- 64-bit processor
- 16 GB RAM
- Ubuntu 20.04 (64-bit version) (http://releases.ubuntu.com/20.04/)
- Disk: 30GB of free space (SSD is highly recommended)

4.6.3. Overview

The overall procedure is as follows.

1. Installing Docker
2. Building the Kibo-RPC Simulator with Docker
3. Setting Android Emulator to run APK
4. Building the Guest Science Manager APK
5. Setting up the network to connect the Android emulator and the simulator
6. Installing your APK to Android Emulator
7. Launching the simulator and run your APK
    i. Launching the Android Emulator
    ii. Starting the Kibo-RPC Simulator
    iii. Running the Guest Science Manager, GDS Simulator and your APK

4.6.4. Installing Docker

Please install Docker engine before setting the simulator.

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg lsb-release
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

kr

Then, logout abd log back in.

For more details, see official instructions:

https://docs.docker.com/engine/install/ubuntu/

https://docs.docker.com/engine/install/linux-postinstall/

4.6.5. Building the Kibo-RPC Simulator with Docker

Download the Kibo-RPC Simulator Setting up scripts from our website.
(https://jaxa.krpc.jp/download.html)

This module contains the following scripts.

- build.sh
- run.sh

The build.sh clones the Astrobee Robot software from GitHub, applies patches of Kibo-RPC modules and builds docker images.

All you have to do is to execute the script as follows.

Note that the building sequence will take several hours to complete and needs large disk space (20-30 GB).

```
unzip krpc4_simulator.zip -d $HOME
cd ${HOME}/krpc4_simulator
bash build.sh
```

Once building the docker images is finished, you can launch the simulator.

Edit hosts file to set up the network.

```
sudo nano /etc/hosts
```

Add the following three lines and save the file.

```
127.0.0.1	hlp
127.0.0.1	mlp
127.0.0.1	llp
```

Finally, you can run a simulation.

```
bash run.sh
```

Is the image below displayed on your screen? If so, installation is complete!

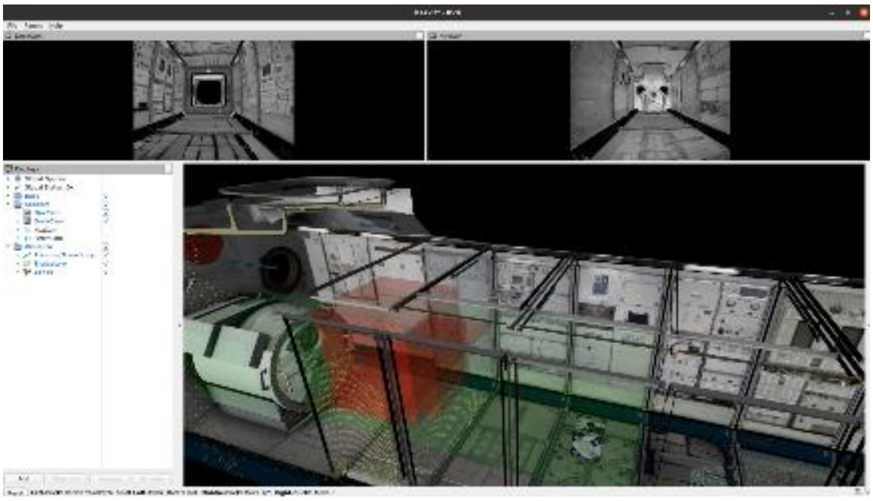


Figure 4-17 Setup result

Type Ctrl+C on the terminal to stop the simulator, and proceed to the next step.

4.6.6. Setting the Android Emulator to run APK

Create an AVD (Android Virtual Device) as follows.

1) Launch Android Studio.
2) Select [Tools] -> [AVDManager].
3) In the Android Virtual Device Manager window, click [+ Create Virtual Device …].
4) Select device **Nexus 5** (Resolution 1080x1920) and click [Next].
5) Select the [x86 Images] tab, choose **Nougat/API Level 25/ABI x86_64/Android 7.1.1(NO Google APIs)**, then click [Next].
   NOTE: Download the system image now if you need it.
6) Set the AVD name to “AstrobeeAndroidSim.”
7) Click [Finish].

In the Android Virtual Device Manager window, you will see “AstrobeeAndroidSim” in the list.

Click the Play button in the Action column. If the AVD launches successfully, you will see the following image.

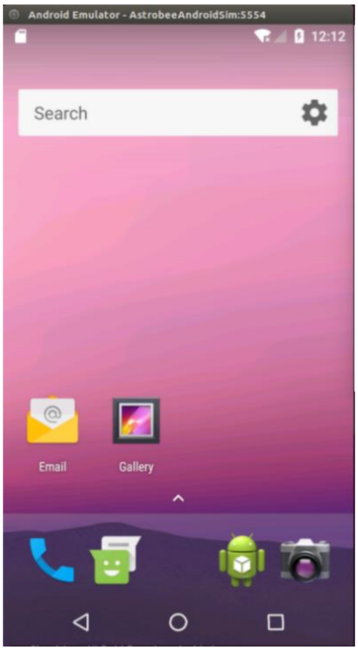


Figure 4-18 Android Emulator screen

See https://github.com/nasa/astrobee_android/blob/master/emulator.md for details.

Shutdown the emulator and let’s procced to the next step.

4.6.7. Building the Guest Science Manager APK

To run your program, you must install the Guest Science Manager APK. (see details at https://github.com/nasa/astrobee_android/blob/master/guest_science_readme.md)

At first, clone the Astrobee Android package from GitHub

```
export ANDROID_PATH=${HOME}/astrobee_android
git clone https://github.com/nasa/astrobee_android.git
git checkout a8560ab0270ac281d8eadeb48645f4224582985e
```

Execute the following commands to build the Guest Science Manager APK.

```
cd $ANDROID_PATH/core_apks/guest_science_manager
ANDROID_HOME=$HOME/Android/Sdk ./gradlew assembleDebug
```

4.6.8. Setting up the network

Setup the network between the Kibo-RPC Simulator and the Android Emulator.
See https://github.com/nasa/astrobee_android/blob/master/emulator.md for details.

(1) Setting the HOST network

Execute the following command to open the hosts file.

```
sudo nano /etc/hosts
```

Comment out the previous three lines, add new three lines below and save the hosts file.

```
#127.0.0.1      hlp
#127.0.0.1      mlp
#127.0.0.1      llp

10.42.0.36      hlp
10.42.0.35      mlp
10.42.0.34      llp
```

(2) Setting the environment variables

Execute the following commands to set the environment variables.

```
export ANDROID_PATH=${HOME}/astrobee_android
export EMULATOR=$HOME/Android/Sdk/tools/emulator
export AVD="AstrobeeAndroidSim"
```

Note that you need to execute the above commands whenever you open a terminal. If you write these commands in your .bashrc file, you don't have to execute them.

(3) Setting up the Android network and starting the Android Emulator

Execute the following commands to set up the Android network and launch the Android Emulator.

```
cd $ANDROID_PATH/scripts
./launch_emulator.sh -n
```

See https://github.com/nasa/astrobee_android/blob/master/emulator.md for details.

* If the Android network does not work, try turning off Wi-Fi in the Android Emulator.

4.6.9. Installing APKs

If the Android Emulator is not running, execute the following commands to start it.

```
cd $ANDROID_PATH/scripts
./launch_emulator.sh -n
```

In another terminal, execute the following commands to install the Guest Science Manager APK and your GS APK.

```
cd $ANDROID_PATH/core_apks/guest_science_manager
adb install -g -r activity/build/outputs/apk/activity-debug.apk
cd <YOUR_APK_PATH>
adb install -g -r app/build/outputs/apk/app-debug.apk
```

4.6.10. Running your program

It's time to run your program!

(1) Launching the Android Emulator

Execute the following commands to launch the Android Emulator if it is not running.

```
cd $ANDROID_PATH/scripts
./launch_emulator.sh -n
```

(2) Starting the Kibo-RPC Simulator

Execute the following command to start the Kibo-RPC Simulator.

```
cd ${HOME}/krpc4_simulator
bash run.sh
```

(3) Running the Guest Science Manager, GDS Simulator and your GS APK

Execute the following commands to start the Guest Science Manager APK and to launch the GDS simulator.

Note: It's required to re-execute these commands whenever you re-launch the simulator or re-install your APK.

```
ANDROID_PATH/scripts/gs_manager.sh start

docker exec -it astrobee bash
cd /src/astrobee/src/tools/gds_helper/src
python3 gds_simulator.py
```

Operate the GDS simulator to run your GS APK.

1) Press any key to grab control
2) Select your GS APK.
3) Type **b** and press **Enter** to start the GS APK.
4) Type **d** and press **Enter** to send a custom guest science command.

Now Astrobee starts to locate the leak!

5. Programming tips

5.1. Do NOT write infinite loops

You **must not** write any infinite loops in your code because no one can stop Astrobee while the loop is executing.

Double check that you use finite loops with a defined counter value, as shown below.

```
// NG
while(!result.hasSucceeded()){
	// do something
}

// OK
final int LOOP_MAX = 5;
int loopCounter = 0;
while(!result.hasSucceeded() && loopCounter < LOOP_MAX){
	// do something
	++loopCounter;
}
```

5.2. Debugging feature for image processing

You can save any Bitmap/Mat type images in the Android Emulator and download the images from the dashboard display. This feature should be useful to check intermediate images of your image processing algorithm.

To save an image, use saveBitmapImage or saveMatImage API as follows.

```
// get/process a bitmap image
Bitmap image = any_function();
// save the image
api.saveBitmapImage(image, “file_name_1”);

// get/process a mat image
Mat img = any_function_mat();
// save the image
api.saveMatImage(img, “file_name_2”);
```

If you are running APK on your local machine, the image can be obtained by the following command.

```
(If your APK is based on TemplateAPK)
adb pull /sdcard/data/jp.jaxa.iss.kibo.rpc.defaultapk/immediate/DebugImages

(If your APK is based on Sample APK)
adb pull /sdcard/data/jp.jaxa.iss.kibo.rpc.sampleapk/immediate/DebugImages
```

Up to 50 images can be saved per simulation, and the maximum image size is 1228800 pixels (1280 x 960 px).

5.3. Dealing with randomness

You must consider the randomness of the environment.

When you want to move the robot, refer to the commands below…

```
// move to point 1
api.moveTo(point1, quaternion1, true);
// move to point 2
api.moveTo(point2, quaternion2, true);
// move to point 3
api.moveTo(point3, quaternion3, true);
```

If there is no randomness in the environment, this code works well.

However, Astrobee may be faced with errors such as **tolerance violations** and **collision detection (*)**, and your code will not work, so you have to provide redundant code, as we see below.

Remember, **Do NOT** allow any infinite loops in your code!

* Tolerance violation error occurs when there is a discrepancy between Astrobee's pose (estimated) and the target pose. Collision detection occurs when Astrobee's HazCam detects any obstacles on the target path. Both errors can occur for a variety of causes including false detection, especially in the real environment.

```
Result result;
final int LOOP_MAX = 5;

// move to point 1(first try)
result = api.moveTo(point1, quaternion1, true);

// check result and loop while moveTo api is not succeeded.
// Do NOT write infinite loop.
int loopCounter = 0;
while(!result.hasSucceeded() && loopCounter < LOOP_MAX){

    // retry
    result = api.moveTo(point1, quaternion1, true);
    ++loopCounter;

}
// move to point 2
//…
```

5.4. About navigation errors

The real world always has uncertainties.Navigation error is one of them and the Kibo-RPC simulator server simulates it.

However, modeling and simulating navigation errors are highly complicated, and this increases the calculation load. Therefore, random error following Gaussian distribution is used generally.

The Kibo-RPC simulator also implements a Gaussian distribution and the parameters are as follows;

```
Regarding position;
   x: mean = 0 m and 3sigma = 0.1 m
   y: mean = 0 m and 3sigma = 0.1 m
   z: mean = 0 m and 3sigma = 0.1 m
  Regarding orientation;
   x: mean = 0 degree and 3sigma = 3 degree
   y: mean = 0 degree and 3sigma = 3 degree
   z: mean = 0 degree and 3sigma = 3 degree
```

You have to consider that self-positioning and self-orientation obtained from the APIs (getRobotKinematics) includes these errors.

5.5. Flashlight

You must consider the effect of light conditions on reading a QR code in the real environment. Flashlight might help you to improve the reading accuracy. In our past experiments, the best brightness value was 0.05.

```
// turn on the front flashlight
api.flashlightControlFront(0.05f);
// you might need some sleep…
Thread.sleep(2000);
// get a camera image
Mat image = api.getMatNavCam();
// turn off the front flashlight
api.flashlightControlFront(0.0f);
```

5.6. Error handling

Various errors can occur due to external and internal causes during program run. Considering all kinds of situations and handling the errors make your program more robust to environmental changes.

Null check: Ensure variable is not null.

```
Mat image = api.getMatNavCam();
// You must handle null pointer
```

```
If (image == null) {
    /** Error handling**/
} else {
    readAR(image);
}
```

Size of list/array: Check the length of list/array when access its element.

```
List<String> list = getStringList();

// Bad
for (int i=0; i<5; i++) {
    // this code cause an exception when the list size is less than 5.
    String str = list.get(i);
}

// Good
for (int i=0; i<list.size(); i++) {
    String str = list.get(i);
}
```

try-catch: Handle exceptions with try-catch statements.

```
try {
    String decodedStr = someDecoder();
    int value = Integer.parseInt(decodedStr);
    // …
} catch (NumberFormatException e) {
    /** Handle the exception occurs when the string is not number **/
} catch (Exception e) {
    /** Handle unexpected exceptions to prevent program down **/
}
```

5.7. Attention to computing resources

If the computing loads are high, Astrobee might be overloaded and not work on orbit. The specifications of Astrobee real robot's HLP are as follows. Note that available resources are different by other software working on HLP. Multithreading is not recommended.

CPU: Qualcomm Snapdragon 820 (4 cores, 2.2GHz)

RAM: 4GB

5.8. Cautions when irradiating laser

Please use the laser only to illuminate the Target. Due to safety reasons, it is prohibited to point the laser at the crew. It is required to inform the intended timing and target of laser irradiation in advance to both flight controllers and the crew. Therefore, create a program where the laser does not hit the crew and does not illuminate a place unnecessarily. DO NOT irradiate the laser blindly, for example, when reading QR/AR is failed. Please check your code not to proceed the processing despite the direction to aim is uncertain.

5.9. Performance of Localization

There is a possibility of losing Astrobee's self-position on-orbit. Once the self-position is lost, Astrobee may not be able to recover on its own. In this case, it means it is a game over. Be aware that this incident is not in the simulator. It is well known that the technology Astrobee uses is likely to lose its self-position when the navigation camera view gets too close to wall, floor, airlock, and so on because the camera cannot capture enough features at those places. Please note the above when creating your program.

For more information, mapped landmark (ML) features coverage heat map is shown in the figures below.

The localization performance depends on how many ML features NavCam captures. The heat map represents the number of mapped landmark (ML) features in a volume of 30 cm^3 along an imaginary grid on the overhead, aft, forward, deck walls and the front of the airlock.

The more ML features in a given volume the more stable Astrobee localization will be.

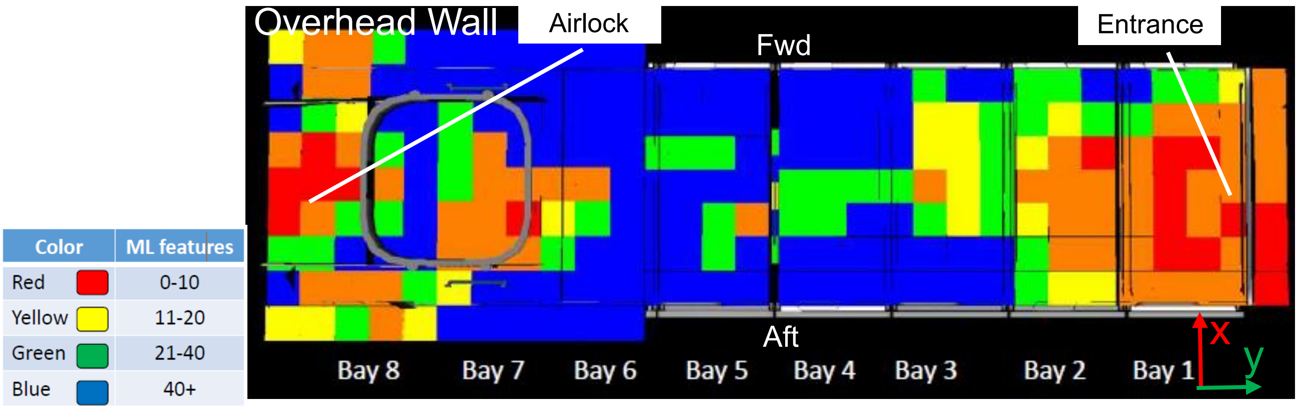


Figure 5-1 ML features on overhead wall

Color	ML features
Red	0-10
Yellow	11-20
Green	21-40
Blue	40+

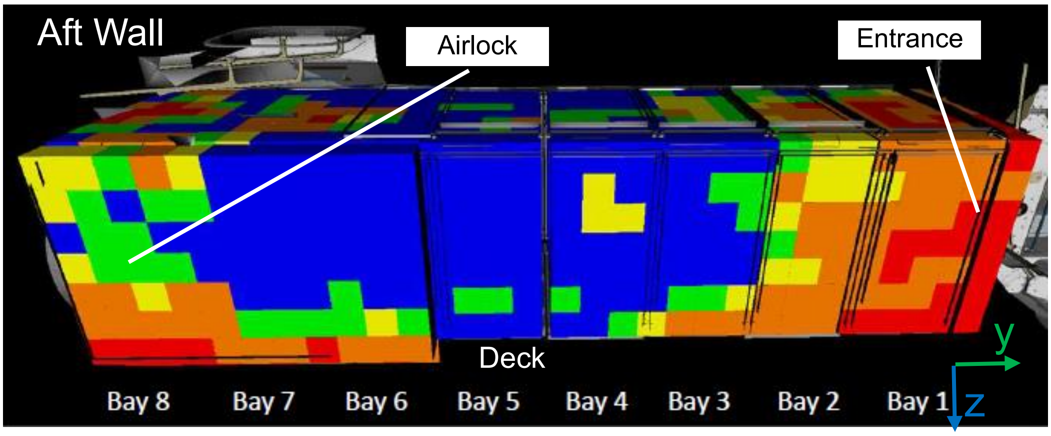


Figure 5-2 ML features on aft wall

Color	ML features
Red	0-10
Yellow	11-20
Green	21-40
Blue	40+

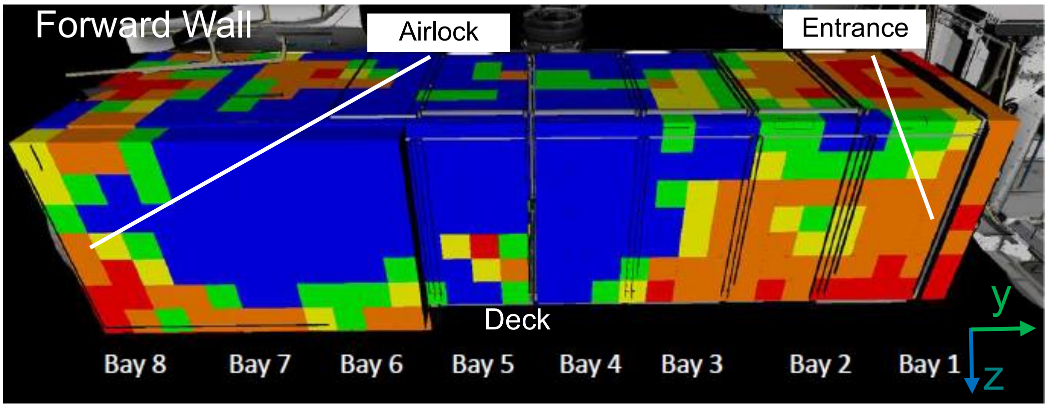


Figure 5-3 ML features on fwd wall

Color	ML features
Red	0-10
Yellow	11-20
Green	21-40
Blue	40+

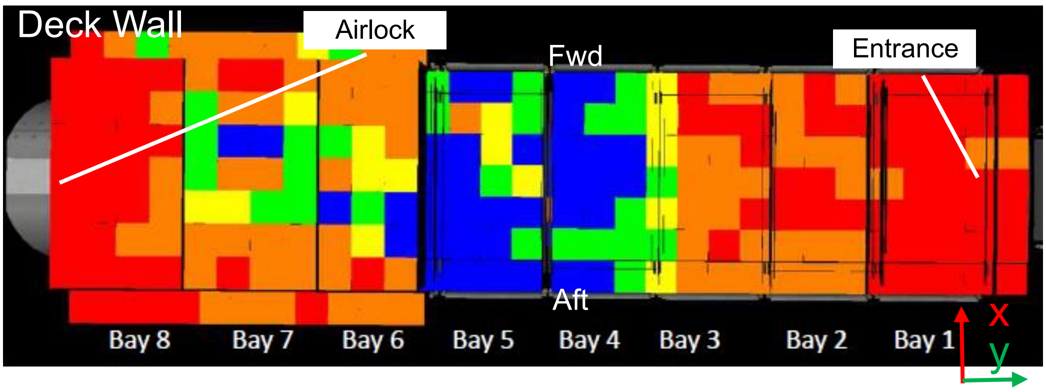


Figure 5-4 ML features on deck wall

Color	ML features
Red	0-10
Yellow	11-20
Green	21-40
Blue	40+

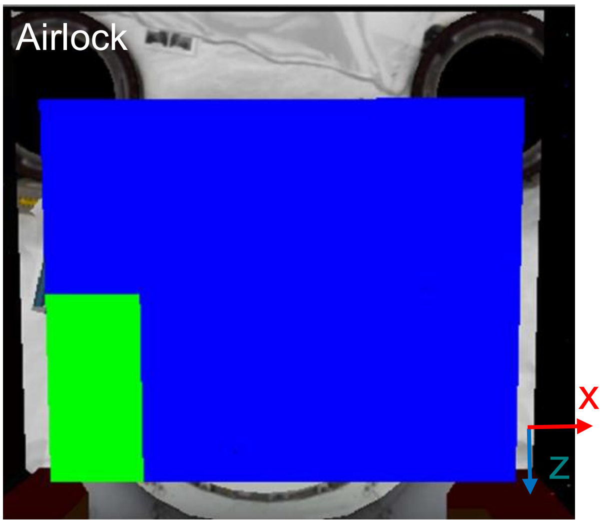


Figure 5-5 ML features on Airlock

5.10. Code review

In the Final Round, before the submitted APK is uplinked to Astrobee on orbit, JAXA / ARC performs a code review for safety confirmation in advance. In the code review, if there is an inappropriate code in the submitted APK, we might delete it or instruct the participants to rewrite it.

5.11. Setting the application ID

Each Final Round APK must have a unique application ID to avoid conflict when installing on Astrobee in the ISS. The application ID will be specified for the finalists later.

5.12. Questions and information exchange

You can post questions, share programming tips and exchange information with other teams on the bulletin board. Make effective use of it to create your program!

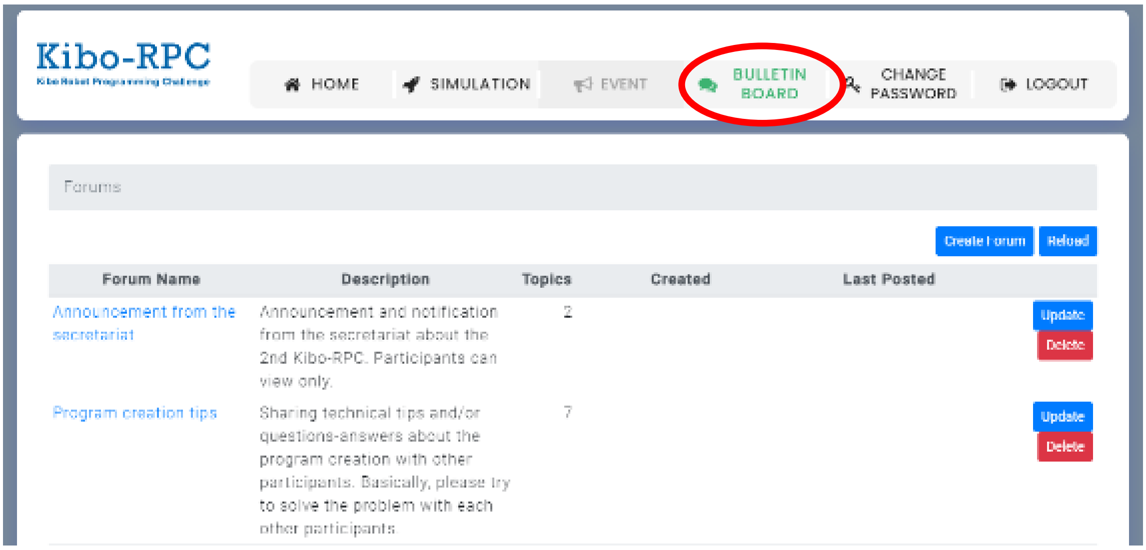


Figure 5-6 BULLETIN BOARD page

6. Simulator change log

Ver.1.0	Initial Release

7. Game API details

Details of the Kibo-RPC's game APIs are listed below.

7.1. Method summary

Table 7-1 Method summary

Modifier and Type	Method and Description
Result	flashlightControlBack(float brightness) Set the brightness of the back flash light.
Result	flashlightControlFront(float brightness) Set the brightness of the front flash light.
List<Integer>	getActiveTargets() Get the current active target IDs.
Bitmap	getBitmapDockCam() Gets Bitmap image of DockCam.
Bitmap	getBitmapNavCam() Get Bitmap image of NavCam.
double[][]	getDockCamIntrinsics() Get camera matrix and distortion coefficients of DockCam.
Mat	getMatDockCam() Gets Mat image of DockCam.
Mat	getMatNavCam() Gets Mat image of NavCam.
double[][]	getNavCamIntrinsics() Get camera matrix and distortion coefficients of NavCam.
Kinematics	getRobotKinematics() Gets current data related to positioning and orientation for Astrobee.
List<Long>	getTimeRemaining() Get the remaining time of the currently active phase and the remaining time of the mission.
Result	laserControl(boolean state) Turns power on/off Laser Pointer.

Modifier and Type	Method and Description
Result	**moveTo**(Point goalPoint, Quaternion orientation, boolean printRobotPosition) Moves Astrobee to the given point and rotates it to the given orientation.
void	**notifyGoingToGoal**() Notice by ROS message that participant is heading to the goal.
Result	**relativeMoveTo**(Point goalPoint, Quaternion orientation, boolean printRobotPosition) Moves Astrobee to the given point using a relative reference and rotates it to the given orientation.
boolean	**reportMissionCompletion**(java.lang.String report) Report mission completion and blink the lights.
void	**saveBitmapImage**(Bitmap image, java.lang.String imageName) Save a bitmap image for debug.
void	**saveMatImage**(org.opencv.core.Mat image, java.lang.String imageName) Save a mat image for debug.
boolean	**startMission**() Astrobee starts counting the mission time.
void	**takeTargetSnapshot**(int targetId) Take a snapshot of Target.

7.2. Method details

- **getRobotKinematics**

```
public Kinematics getRobotKinematics()
```

Gets current data related to positioning and orientation for Astrobee. Note that the data cannot be trusted when the confidence is POOR or LOST.

Returns:

Current Kinematics.

- **getBitmapNavCam**

```
public Bitmap getBitmapNavCam()
```

Get Bitmap image of NavCam.

Returns:

Bitmap image of NavCam(1280 px x 960 px) or null if an internal error occurs. Format:Bitmap.Config.ARGB_8888

- **getBitmapDockCam**

```
public Bitmap getBitmapDockCam()
```

Gets Bitmap image of DockCam.

Returns:

Bitmap image of DockCam(1280 px x 960 px) or null if an internal error occurs. Format:Bitmap.Config.ARGB_8888

- **getMatNavCam**

```
public Mat getMatNavCam()
```

Gets Mat image of NavCam.

Returns:

Mat image of NavCam(1280 px x 960 px) or null if an internal error occurs. Format:CV8UC1

- **getMatDockCam**

```
public Mat getMatDockCam()
```

Gets Mat image of DockCam.

Returns:

Mat image of DockCam(1280 px x 960 px) or null if an internal error occurs. Format:CV8UC1

- **flashlightControlFront**

```
public Result flashlightControlFront(float brightness)
```

Set the brightness of the front flash light.

Parameters:

brightness - Brightness percentage between 0 - 1.

Returns:

A Result instance carrying data related to the execution. Returns null if the command is NOT executed because of an error

- **flashlightControlBack**

```
public Result flashlightControlBack(float brightness)
```

Set the brightness of the back flash light.

Parameters:

brightness - Brightness percentage between 0 - 1.

Returns:

```
A Result instance carrying data related to the execution. Returns null
if the command is NOT executed because of an error.
```

- **moveTo**

```
public Result moveTo(Point goalPoint,
                     Quaternion orientation,
                     boolean printRobotPosition)
```

Moves Astrobee to the given point and rotates it to the given orientation.

Parameters:

`goalPoint` - Absolute cardinal point (xyz)

`orientation` - An instance of the Quaternion class. You may want to use CENTER_US_LAB or CENTER_JEM as an example depending on your initial position.

`printRobotPosition` - Flag whether to print robot positions in log or not.

Returns:

A Result instance carrying data related to the execution. Returns null if the command is NOT executed because of an error.

- **relativeMoveTo**

```
public Result relativeMoveTo(Point goalPoint,
                             Quaternion orientation,
                             boolean printRobotPosition)
```

Moves Astrobee to the given point using a relative reference and rotates it to the given orientation. This api can take up to 30 seconds when localization is not accurate enough.

Parameters:

`goalPoint` - The relative end point (relative to Astrobee)

`orientation` - The absolute orientation

`printRobotPosition` - Flag whether to print robot positions in log or not.

Returns:

A Result instance carrying data related to the execution. Returns null if the command is NOT executed because of an error.

- **laserControl**

```
public Result laserControl(boolean state)
```

Turns power on/off Laser Pointer. If it is same state as input parameter, nothing happens.

Parameters:

`state` - Set a laser pointer true:power on / false:power off.

Returns:

A Result instance carrying data related to the execution. Returns null if the command is NOT executed because of an error

- **startMission**

```
public boolean startMission()
```

Undock Astrobee from docking station. Astrobee then starts counting the mission time.

Returns:

Returns True if the execution is successful. Returns false if the command is NOT executed because of an error.

• notifyGoingToGoal

```
public void notifyGoingToGoal()
```

Notifies that Astrobee is moving toward the goal. This method must be executed, but whether it is called does not affect the simulator results.

• reportMissionCompletion

```
public boolean reportMissionCompletion(java.lang.String report)
```

Report mission completion and blink the lights.

Parameters:

`report` - a report message paired with QR code content.

Returns:

Returns True if the execution is successful. Returns false if the command is NOT executed because of an error.

• takeTargetSnapshot

```
public void takeTargetSnapshot(int targetId)
```

Take a snapshot of Target. The laser is turned off at the end of this API.

Parameters:

`targetId` - Target ID to take snapshot

• getNavCamIntrinsics

```
public double[][] getNavCamIntrinsics()
```

Get camera matrix and distortion coefficients of NavCam. Different values are returned on orbit and in the simulator. *The parameter values are different between the simulator and real Astrobee in the ISS as shown below. Therefore this API returns different value depending on whether the APK is running on simulator or on real Astrobee.

Returns:

Array of camera parameters [camera matrix, distortion coefficients] for NavCam. The array of the camera matrix and distortion coefficients is as follows.

・Simulator

Camera matrix: [523.105750, 0.000000, 635.434258,0.000000, 534.765913, 500.335102,0.000000, 0.000000, 1.000000]

Distortion coefficients: [-0.164787, 0.020375, -0.001572, -0.000369, 0.000000]

・Real Astrobee

Camera matrix: [608.8073, 0.0, 632.53684, 0.0, 607.61439, 549.08386, 0.0, 0.0, 1.0]

Distortion coefficients: [-0.212191, 0.073843, -0.000918, 0.001890, 0.0]

• getDockCamIntrinsics

```
public double[][] getDockCamIntrinsics()
```

Get camera matrix and distortion coefficients of DockCam. Different values are returned on orbit and in the simulator. *The parameter values are different between the simulator and real Astrobee in the ISS as shown below. Therefore this API returns different value depending on whether the APK is running on simulator or on real Astrobee.

Returns:

Array of camera parameters [camera matrix, distortion coefficients] for DockCam. The array of the camera matrix and distortion coefficients is as follows.

・Simulator

Camera matrix: [661.783002, 0.000000, 595.212041,0.000000, 671.508662, 489.094196,0.000000, 0.000000, 1.000000]

Distortion coefficients: [-0.215168, 0.044354, 0.003615, 0.005093, 0.000000]

・Real Astrobee

Camera matrix: [753.51021, 0.0, 631.11512, 0.0, 751.3611, 508.69621, 0.0, 0.0, 1.0]

Distortion coefficients: [-0.411405, 0.177240, -0.017145, 0.006421, 0.000000]

• saveBitmapImage

```
public void saveBitmapImage(Bitmap image,
                            java.lang.String imageName)
```

Save a bitmap image for debug. The maximum pixel size of an image is 1228800 (height x width) and up to 50 images can be saved per simulation. The image is saved in Android Emulator (/sdcard/data/) as a png file and can be download on the dashboard display.

Parameters:

`image` - Bitmap Images to save.

`imageName` - string Image name to save.

• saveMatImage

```
public void saveMatImage(Mat image,
                         java.lang.String imageName)
```

Save a mat image for debug. The maximum pixel size of an image is 1228800 (height x width) and up to 50 images can be saved per simulation. The image is saved in Android Emulator (/sdcard/data/) as a png file and can be download on the dashboard display.

Parameters:

```
image - Mat Images to save.

imageName - string Image name to save.
```

- **getTimeRemaining**

```
public java.util.List<java.lang.Long> getTimeRemaining()
```

Get the remaining time of the currently active phase and the remaining time of the mission.

Returns:

```
Active Time left and Mission Time left.
(0): Remaining milliseconds of Active Time.
(1): Remaining milliseconds of Mission Time.
```

- **getActiveTargets**

```
public java.util.List<java.lang.Integer> getActiveTargets()
```

Get the current active target IDs.

Returns:

```
List of active target IDs.
```

7.2.1. Type information

Please refer the following links for information about Types implemented in astrobee_android.

Table 7-2 Type information

Type	URL
gov.nasa.arc.astrobee.Kinematis	https://github.com/nasa/astrobee_android/blob/a8560ab0270ac281d8eadeb48645f4224582985e/astrobee_api/api/src/main/java/gov/nasa/arc/astrobee/Kinematics.java
gov.nasa.arc.astrobee.Result	https://github.com/nasa/astrobee_android/blob/a8560ab0270ac281d8eadeb48645f4224582985e/astrobee_api/api/src/main/java/gov/nasa/arc/astrobee/Result.java
gov.nasa.arc.astrobee.types.Vec3d	https://github.com/nasa/astrobee_android/blob/a8560ab0270ac281d8eadeb48645f4224582985e/astrobee_api/api/src/main/java/gov/nasa/arc/astrobee/types/Vec3d.java
gov.nasa.arc.astrobee.types.Quaternion	https://github.com/nasa/astrobee_android/blob/a8560ab0270ac281d8eadeb48645f4224582985e/astrobee_api/a

Type	URL
	pi/src/main/java/gov/nasa/arc/astrobee/types/Quaternion.java
gov.nasa.arc.astrobee.types.Point	https://github.com/nasa/astrobee_android/blob/a8560ab0270ac281d8eadeb48645f4224582985e/astrobee_api/api/src/main/java/gov/nasa/arc/astrobee/types/Point.java

8. Tips for Astrobee Characteristics

The tips for Astrobee characteristics will be provided in this section to be considered for successful runs.

8.1. Rendering of Astrobee

Figure 8-1 indicates Astrobee is equipped with some external hardware components. The hardware surrounded with one-foot cube (about 32 cm wide) are used in the simulator and ISS Final Round.

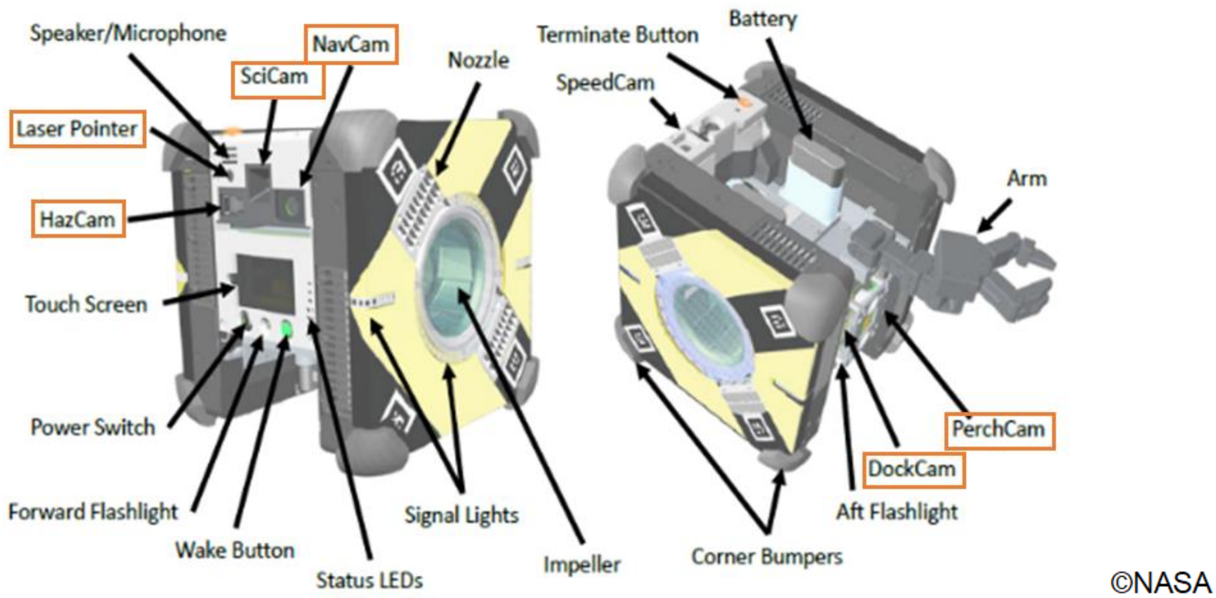


Figure 8-1 External hardware components

Table 8-1 External hardware components

Name	Explanation
HazCam	A monochrome camera for detecting obstacles within 30 cm
NavCam	A monochrome camera for image data processing and taking a photo after sending finish command
SciCam	A color camera for taking a video (The participants cannot utilize this.)
DockCam	A monochrome camera for docking to the docking station
PerchCam	A monochrome camera for grabbing a handrail
Laser Pointer	Irradiating the target
Flashlight	Use this when reading QR code
Speaker	Playing audio files. In the 4th Kibo-RPC, the speaker will not be used.

Figures and tables below show the positions of the hardware components.

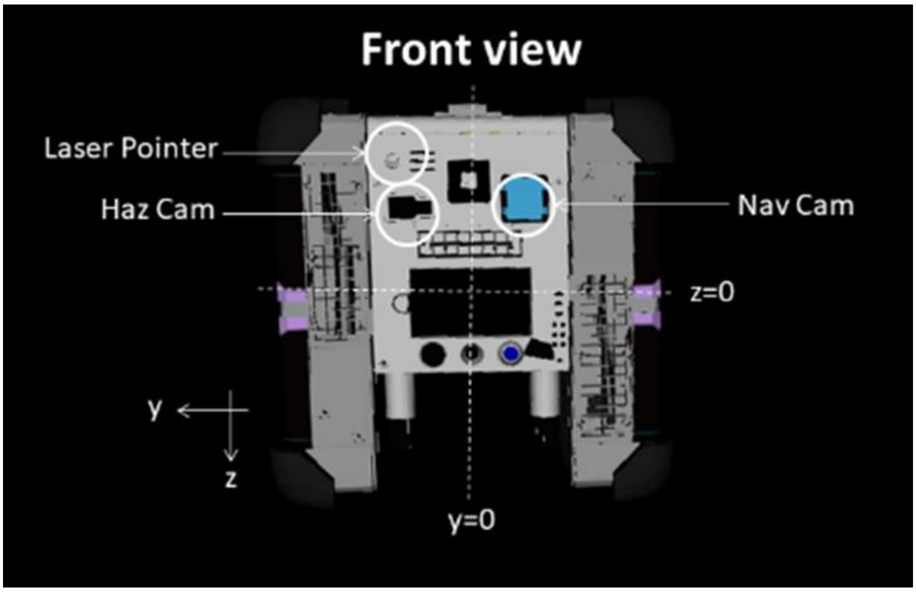


Figure 8-2 Astrobee Front View

Table 8-2 Distances from center point

	x[m]	y[m]	z[m]
Nav Cam	0.1177	-0.0422	-0.0826
Haz Cam	0.1328	0.0362	-0.0826
Laser Pointer	0.1302	0.0572	-0.1111

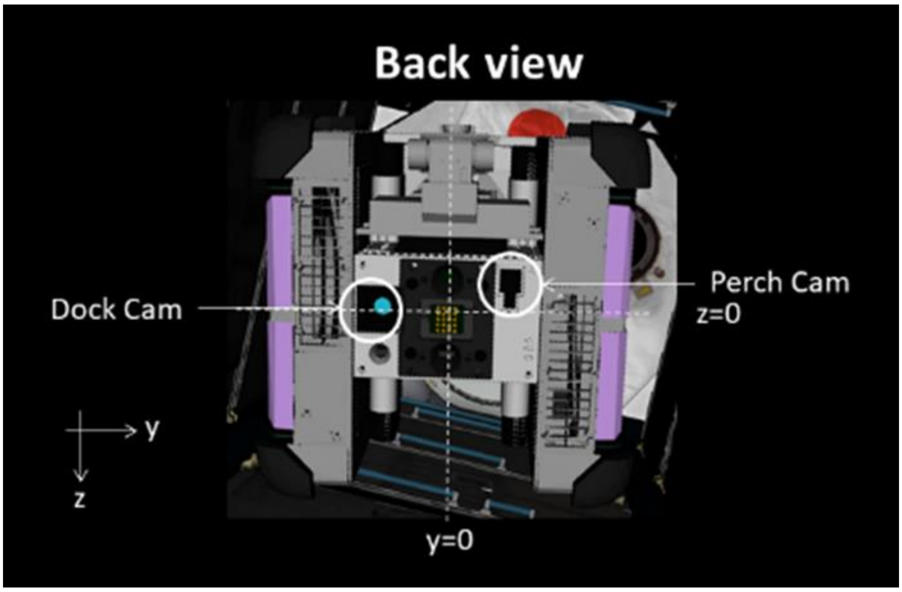


Figure 8-3 Astrobee Back View

Table 8-3 Distances from center point

	x[m]	y[m]	z[m]
Dock Cam	-0.1061	-0.054	-0.0064
Perch Cam	-0.1331	0.0509	-0.0166

8.2. Specification of Astrobee

- Mass: 10kg (Installed only two of the four batteries)
- Maximum Velocity: 0.5 m/s
- Maximum Thrust (X axis): 0.6 N
- Minimum moving distance: 0.05 m.
- Minimum rotating angle: 7.5 degrees.
- If the Astrobee detects the actual obstacles in front, it will automatically stop and then maintain its position and orientation. The moving path it is on will also be discarded.

8.3. References of Astrobee

- GitHub-1 (https://github.com/nasa/astrobee)
- GitHub-2 (https://github.com/nasa/astrobee_android)
- Website of Astrobee (https://www.nasa.gov/astrobee)