

Pepper: the 2024 updated developer guide

Authors: Jessica Frabotta, Alessio Ferrone



Table of contents

• Introduction	3
• Overview	4
• QiSDK	7
• Android Studio installation and configuration	11
• SDK installation and configuration	15
• Troubleshooting and Windows 11 workaround	23
• Creating an example application	25
• Python SDK	32
• SDK installation	34
• Communication with a real robot and Robot Viewer	38
• Example code	40
• References	42

Introduction

Pepper is a humanoid robot designed by SoftBank Robotics, formerly known as Aldebaran Robotics. Humanoid robots are playing increasingly important roles in real-life tasks, especially in indoor applications. They are often used in retail stores, banks and hospitality settings to greet customers, provide information and entertain. Additionally, they serve as educational tools in classrooms and companions in healthcare settings, offering company and support to patients.

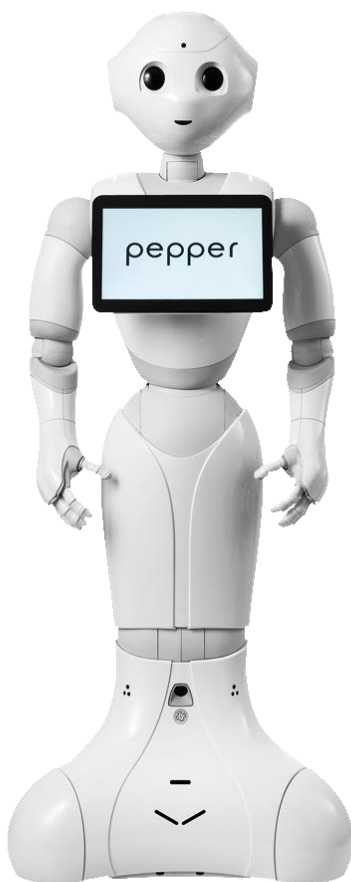


FIGURE 1. Pepper - humanoid robot created by Aldebaran-Softbank with the purpose of being a human companion.

Pepper is programmable, allowing developers to customize its behavior for specific applications. The objective of this guide is to make development for Pepper easier by providing step-by-step instructions on all the installations and configurations needed to develop applications with this robot.

Following the acquisition of Aldebaran Robotics by SoftBank Robotics, the SDKs used to develop applications for Pepper underwent changes, leading to more dispersed and fragmented documentation for developers. Unfortunately, Pepper is no longer being produced. In 2021, SoftBank Robotics announced that it had halted the production of Pepper robots. This discontinuation significantly impacted the maintenance of software updates and further worsened the issues with developer documentation.

This guide outlines the main steps to set up a functional development environment using the two primary SDKs for programming Pepper: QiSDK [1], which is Android-based and Python SDK [2].

Overview

Pepper is 1.2 meters tall and equipped with three wheels and 17 joints. The wheels allow for smooth movement. Due to the long life of its battery, it can work for 12 hours comfortably and be charged if needed. Pepper can perceive its environment through various sensors, which enable its numerous functionalities to assist humans.

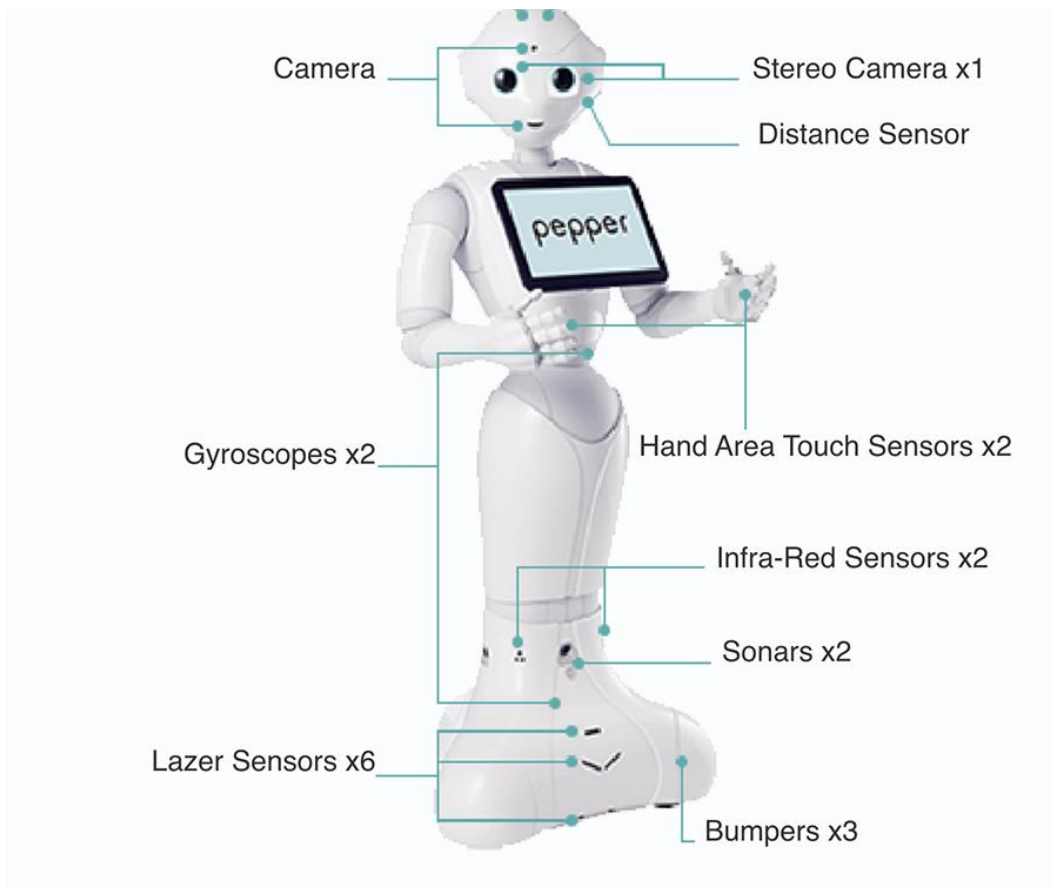


FIGURE 2. Pepper robot sensors.

The robot's head features four microphones and two HD cameras, located in the mouth and forehead. Depending on the robot model (Pepper 1.8 or Pepper 1.8a), there can be either a stereo camera or a 3-D depth [3] sensor behind the eyes. Additionally, the robot is equipped with a gyroscope in the torso and touch sensors in the head and hands. Its mobile base includes two sonars, six lasers, three bumper sensors and another gyroscope [4].

A very important component of the robot is the tablet, which can be used to make choices and to delight, entertain and inform users.

Pepper can support two types of operating systems: NAOqi 2.5 and NAOqi 2.9 [5]. Neither of these operating systems is hardware-dependent, and the choice of the operating system does not impact the robot's battery life. NAOqi 2.5 is a Linux-based system, whereas NAOqi 2.9 is Android-based. NAOqi 2.5 allows development through the Python SDK. Instead, NAOqi 2.9 adopts the Android ecosystem, using Java and Kotlin as the main programming languages within the QiSDK library.

NAOqi 2.5 is designed to use the robot as a development platform whereas with NAOqi 2.9, Pepper is meant to be a part of a solution. NAOqi 2.5 offers a very flexible open platform environment with more than 1000 APIs available, low-level programming and direct access to the sensors' and actuators' raw data. Meanwhile, NAOqi 2.9 provides a more stable and secure environment with only 20 different APIs for high-level development.

Last but not least, it is technically possible to upgrade from NAOqi 2.5 to NAOqi 2.9. However it is not possible to downgrade the other way.

		Pepper with NAOqi 2.5 OS	Pepper with NAOqi 2.9 OS
Specs	OS tech	Linux Based	Android Based
	Programming Language	Python, HTML, Javascript	Java, Kotlin
		C++ SDK available ROS compatible	
	SDK	Choregraphe	Android Studio Plugin
	API	Python SDK	
		1000+ APIs	Around 20 API (possible actions)

FIGURE 3. Comparison of Pepper's OS versions - Datasheet

NAOqi 2.5 is compatible with Choregraphe [6], a multi-platform desktop application which uses a box-based programming logic that allows to create animations, behaviors, dialogues and test them on a virtual or physical robot. The Choregraphe behaviors can be enriched with additional custom Python code.

The most common language used to program on NAOqi 2.5 is Python. HTML and JavaScript are also used for tablet displays.

On the other hand, Python SDK let you run scripts directly on Pepper. You can access all the API of NAOqi 2.5 via Choregraphe, therefore everything that is possible with the Python SDK is possible with Choregraphe and vice versa. We will see in detail in the next chapters how the Python SDK works, for additional knowledge about Choregraphe check this link:

http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html

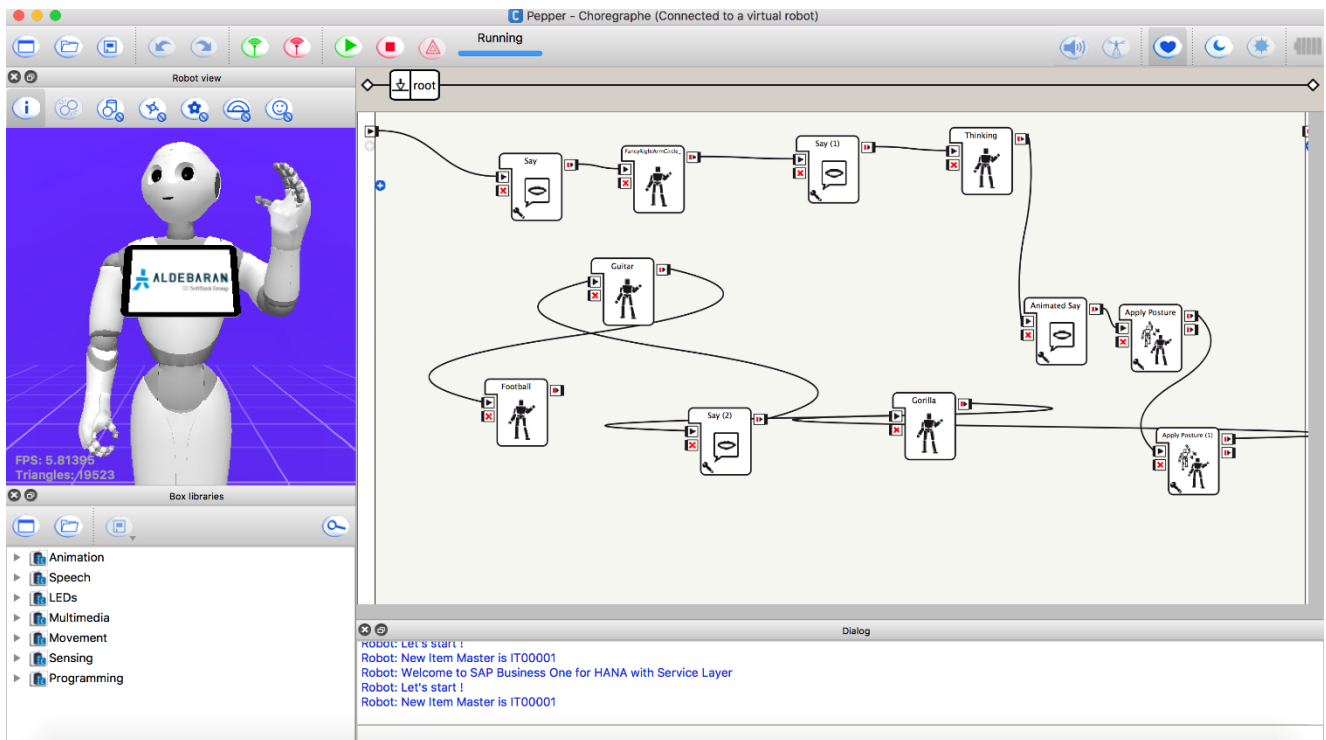


FIGURE 4. Choregraphe tool, available only for NAOqi 2.5

The NAOqi 2.9 Android Studio plugin provides a set of graphical tools and a Java library, the QiSDK. This allows you to make the robot move, talk, and interact with people in a simple way using a straightforward API, directly from your Android activity. It also includes a Layout Editor to work directly on the tablet display.

In the next chapter, we will follow the step-by-step installation and configuration of Android Studio and the required plugin.

QiSDK

QiSDK allows developers to take control of the Pepper robot using its Android tablet. This makes developing applications for Pepper similar to developing apps for any Android device, such as a tablet or a smartphone.

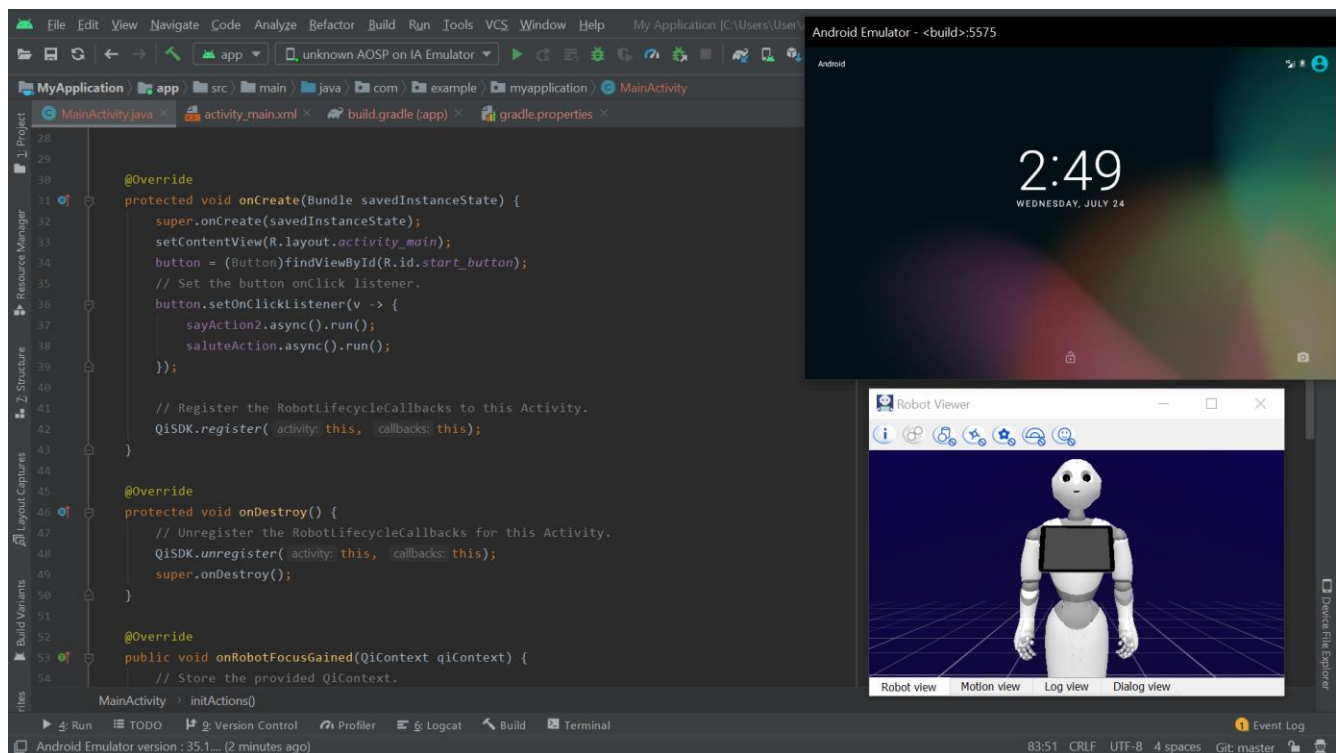


FIGURE 5. Pepper development environment in Android Studio

The APIs offered by QiSDK belong to four main areas:

- **Autonomous Abilities:** these allow the activation or deactivation of behaviors that the robot performs autonomously, such as blinking or following a specific person with its gaze.
- **Conversation:** these APIs enable the creation of a simple chatbot through the QiChat script language or the management of chat functionalities, including listening and speaking actions.
- **Motion:** these allow the use of animations to make Pepper execute a series of movements with its limbs and head, or follow a predefined trajectory. Some of these APIs also support basic functions of mapping and localization.
- **Perceptions:** these enable Pepper to approach a specific human to initiate interaction with them.

The Pepper plugin for Android Studio offers many tools that mainly focus on allowing the testing of applications developed for Pepper on emulators instead of directly on a physical robot. The main one is the **Robot Viewer**.

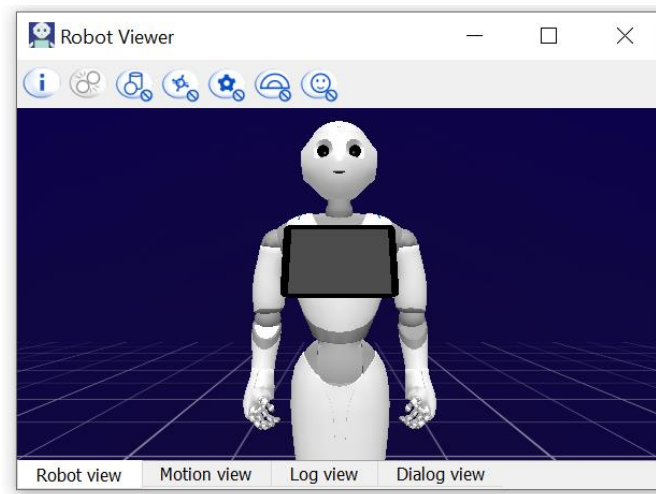


FIGURE 6. Robot Viewer

The Robot viewer gathers several tools allowing to monitor the robot on which the plugin is connected:

- **Robot View:** displays a 3D view of the robot (it is the view shown in figure 6). An emulated robot is displayed on a blue background, while the remote view of a real robot is displayed on a green background.
- **Motion View:** allows you to control the movements of the robot.
- **Dialog View:** allows you to watch dialog outputs and enter dialog inputs.
- **Log View:** allows you to watch the logs.

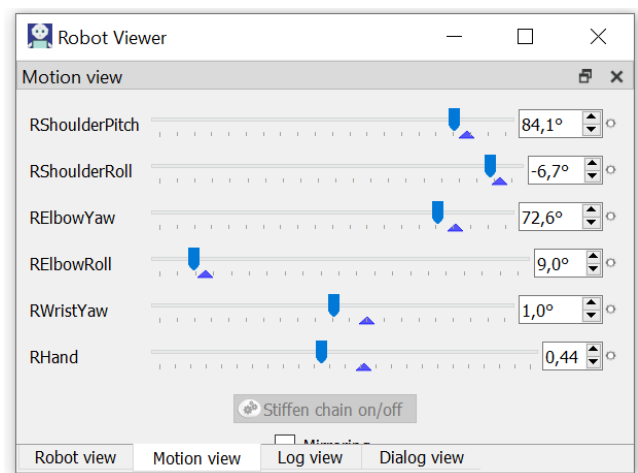
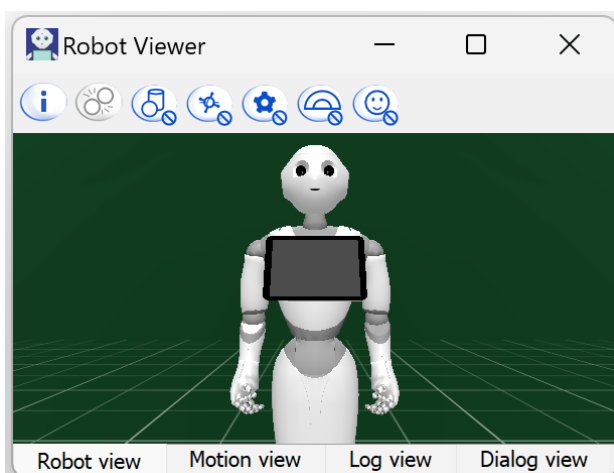


FIGURE 7. Robot View of a real robot and Motion View

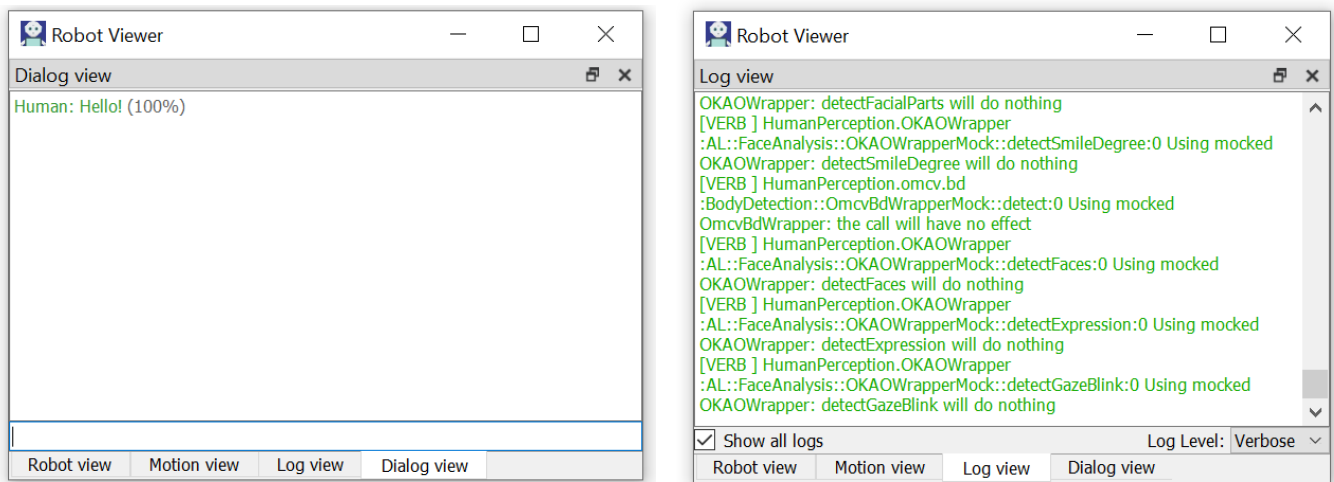


FIGURE 8. Dialog View and Log View

When the robot emulator is launched, both the Robot Viewer and an Android Virtual Device (AVD) are started. The Android Virtual Device is configured to match Pepper's tablet and can be used to preview how the application will work on Pepper's tablet and to fix layout issues.

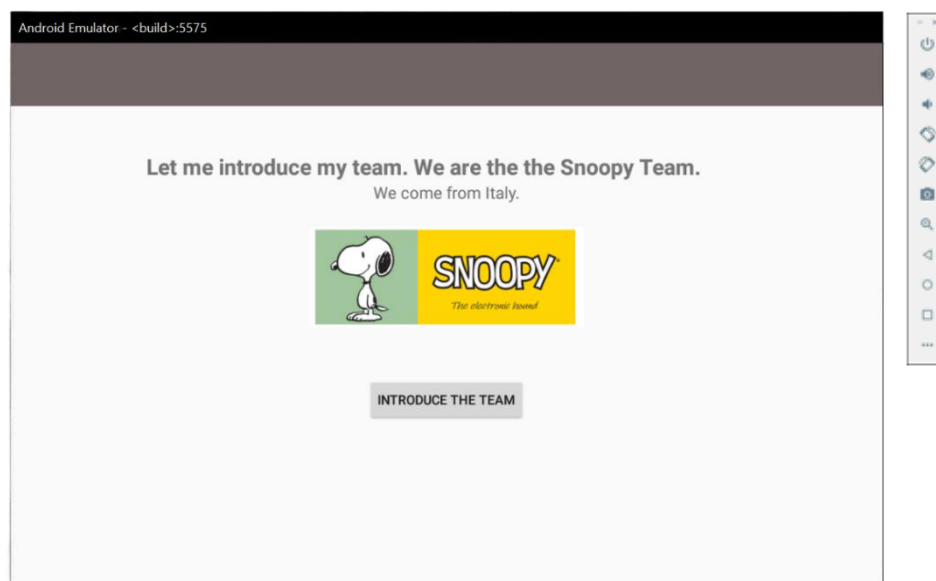
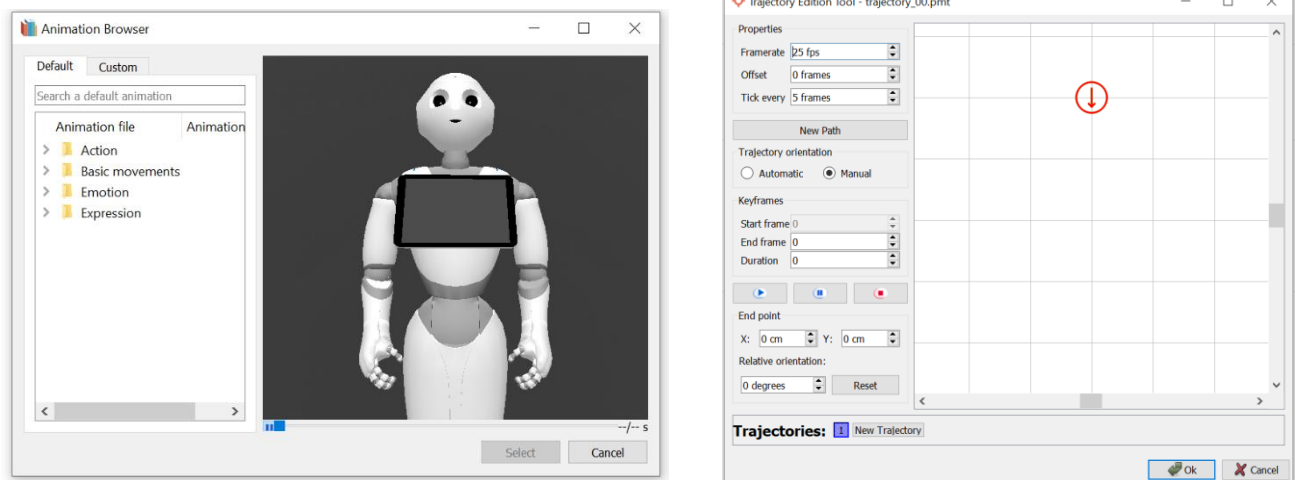


FIGURE 9. Android Virtual Device with a Pepper application

Additionally Android Studio provides a bundled Layout Editor that allows the developers to get a preview of what their layouts will look like on devices. Android Studio allows developers to preview their layouts on various screens and resolutions. To ensure an accurate display of your applications for Pepper, you need to configure Android Studio to use the Pepper Tablet settings. In the next chapter we will see how to correctly configure this aspect.

Additional tools offered by the Android Studio plugin are:

- **Animation Browser:** it is used to manage animations which are predefined or custom sets of movements and gestures that can be performed by the robot.
- **Trajectory Editor:** while animation lets you define the movement of the robot's actuators, trajectory lets you set the path that the robot will move along. With this tool you can define your trajectory file to make your robot move to a destination.
- **Chat Editor:** it is an extension of Android Studio's text editor designed to facilitate the creation and management of conversational content.



```
hello.top x
1  topic: ~hello()
2  # Defining extra concepts out of words or group of words
3  concept:(hello) [hello hi hey "good morning" greetings]
4
5  # Replying to speech
6  u:(~hello) ~hello
7
8
```

FIGURE 10. Animation Browser, Trajectory Editor and Chat Editor

Android Studio installation and configuration

Installing the Pepper plugin for Android Studio can be challenging due to its incompatibility with the latest versions of Android Studio [7]. The last compatible version is Android Studio 3.4 Bumblebee - 2021.1.1 Patch 3. So to use the plugin, you must install an older version of Android Studio.

You can find all the older versions here: <https://developer.android.com/studio/archive>

All the configurations and settings discussed in this and the next chapter have been tested only on Windows 10 and Windows 11. We do not guarantee that they will work on other operating systems. The installation of the plugin on Windows 10 is the most stable and we did not detect any bugs or issues with it. On the other hand, the installation of the plugin on Windows 11 can cause unexpected crashes of Android Studio and requires workarounds to allow the use of common tools like Robot Viewer or Animation Browser. Unfortunately, we could not find a solution to a crash of Android Studio that occurs when we try to deploy an application on a physical robot using the plugin on Windows 11. We will discuss these issues in detail later in this guide.

The steps we will follow are the same for Windows 10 and Windows 11, except where specified. The version of Android Studio we installed is **3.6.3 April 17, 2020** (it can be found in the Android Studio download archives linked above).

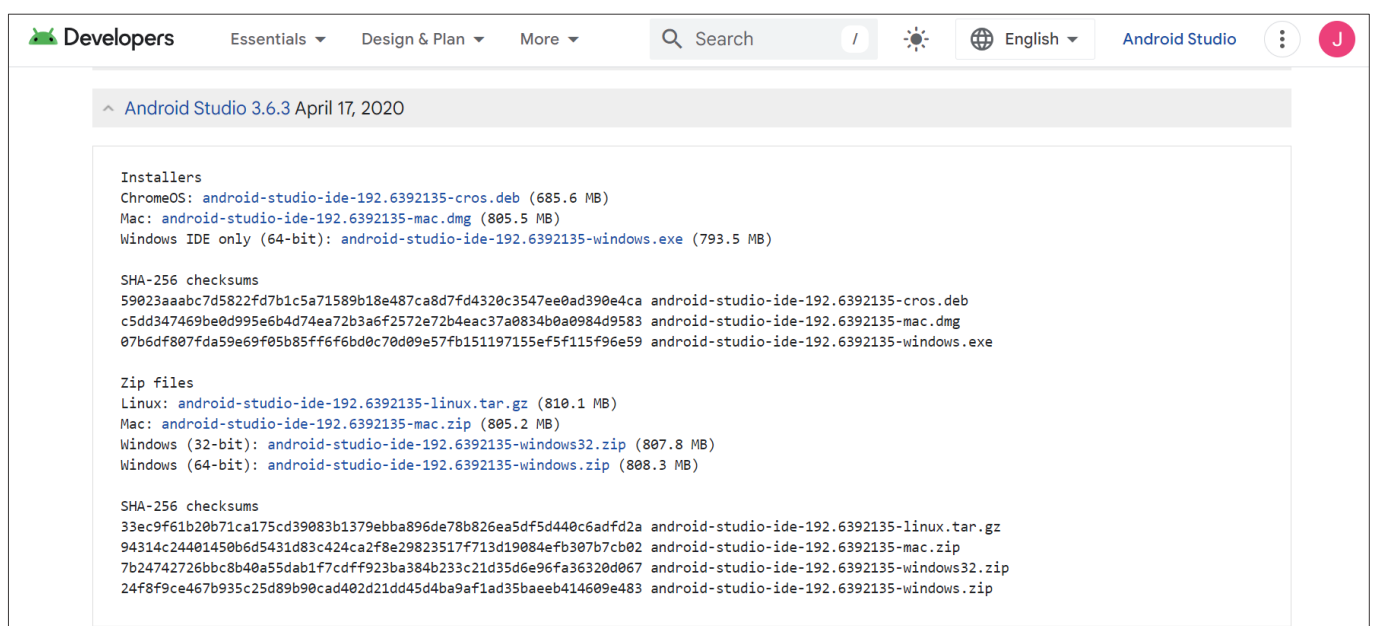


FIGURE 11. Android Studio download archives

After downloading, start the setup file. We confirmed all the default options displayed in the initial steps of the installation by clicking the “**Next**” button.

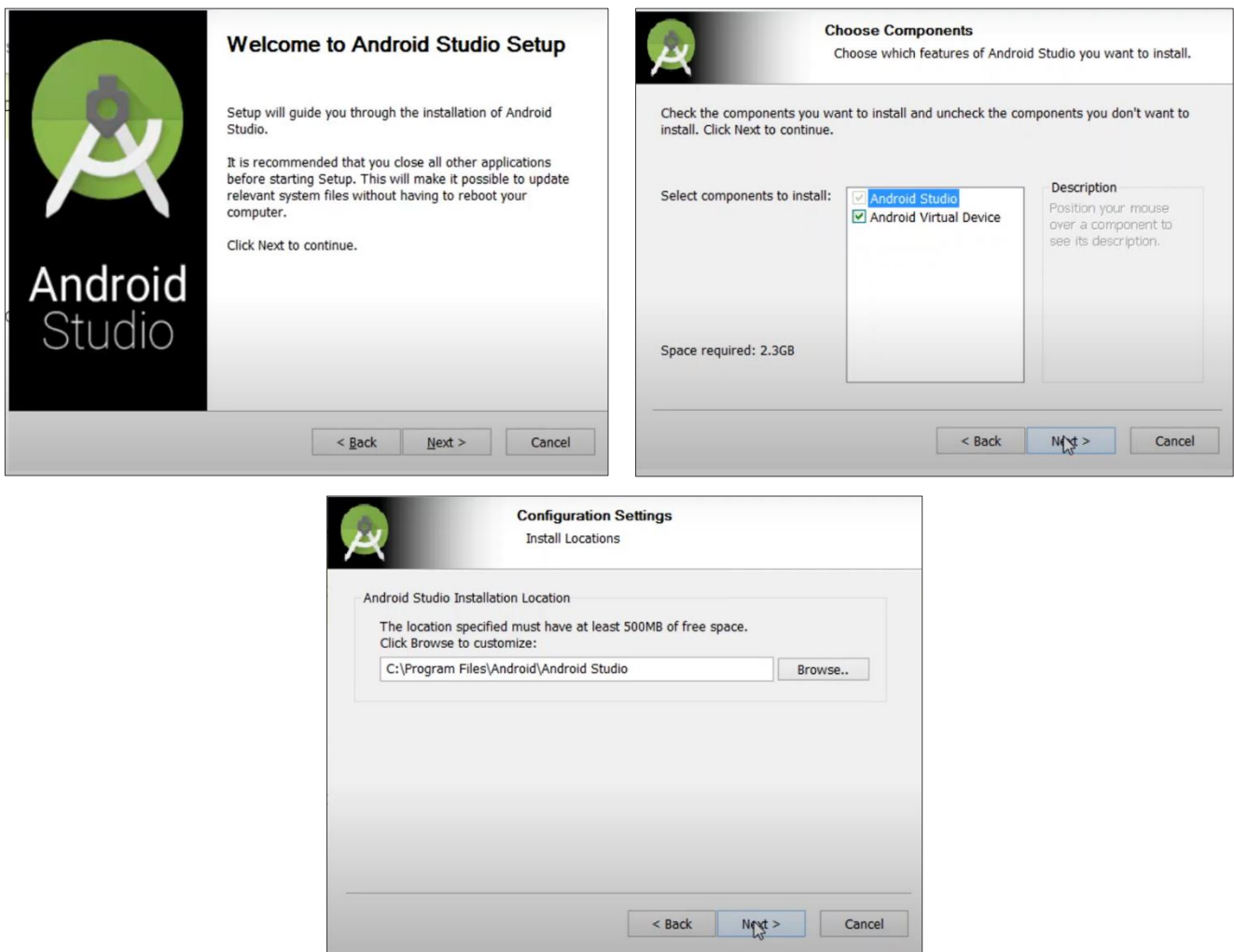


FIGURE 12. First three pages that appear after starting the setup file

At this point, simply click on the “**Install**” button. After the process ends, click on “**Finish**”.

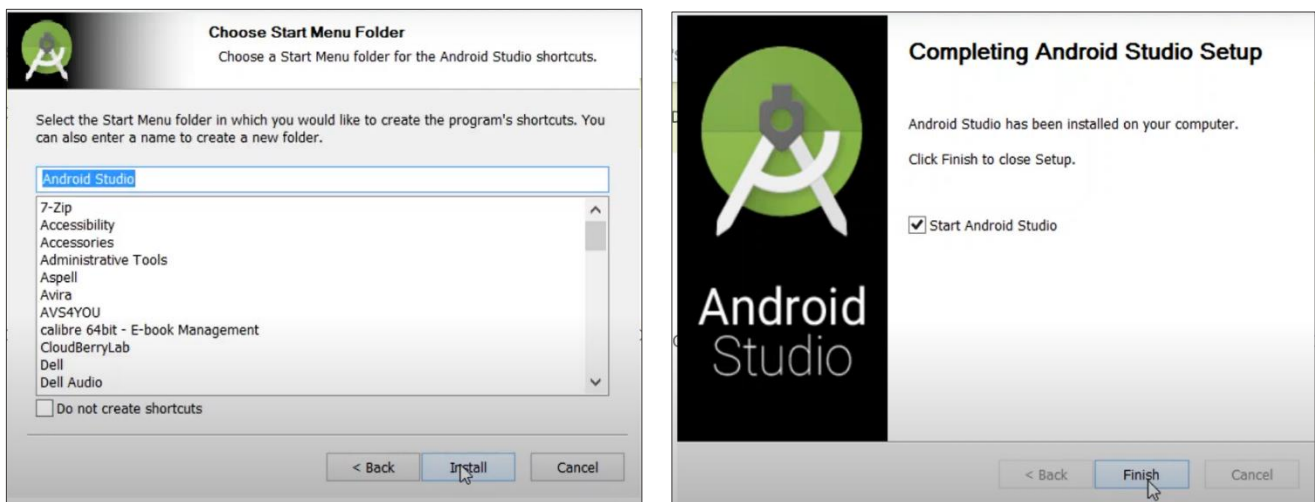


FIGURE 13. Last page before the installation and page that confirms the installation

At this point, the installation is complete, but the most challenging part begins: the configuration. Upon opening Android Studio for the first time, some settings will be requested, such as the UI theme. After that, we can finally choose the SDK settings, which include the SDK Platforms settings and the SDK Tools settings. These settings can be changed at any time, even after the initial choices, by accessing them from a created project through **File > Settings > Appearance & Behavior > System Settings > Android SDK** or from the **“Configure”** option on the Android Studio homepage.

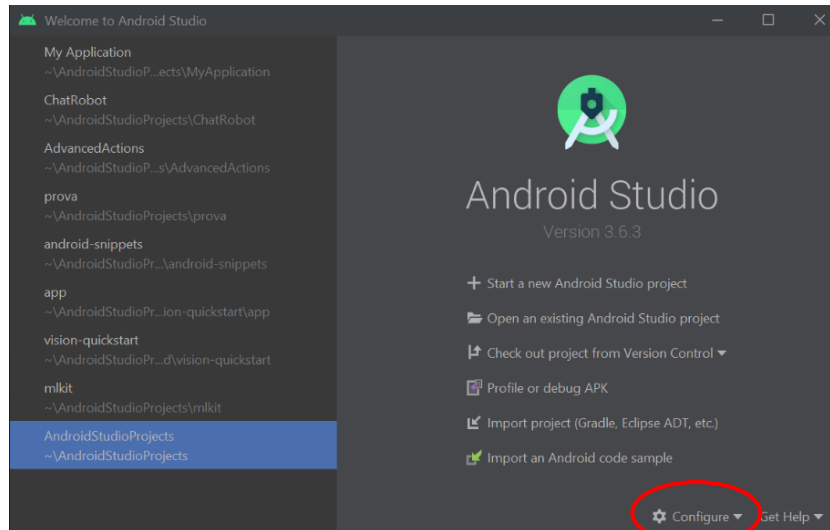


FIGURE 14. “Configure” option on the homepage

The SDK platform we need is **Android 10.0 (API level 29)**.

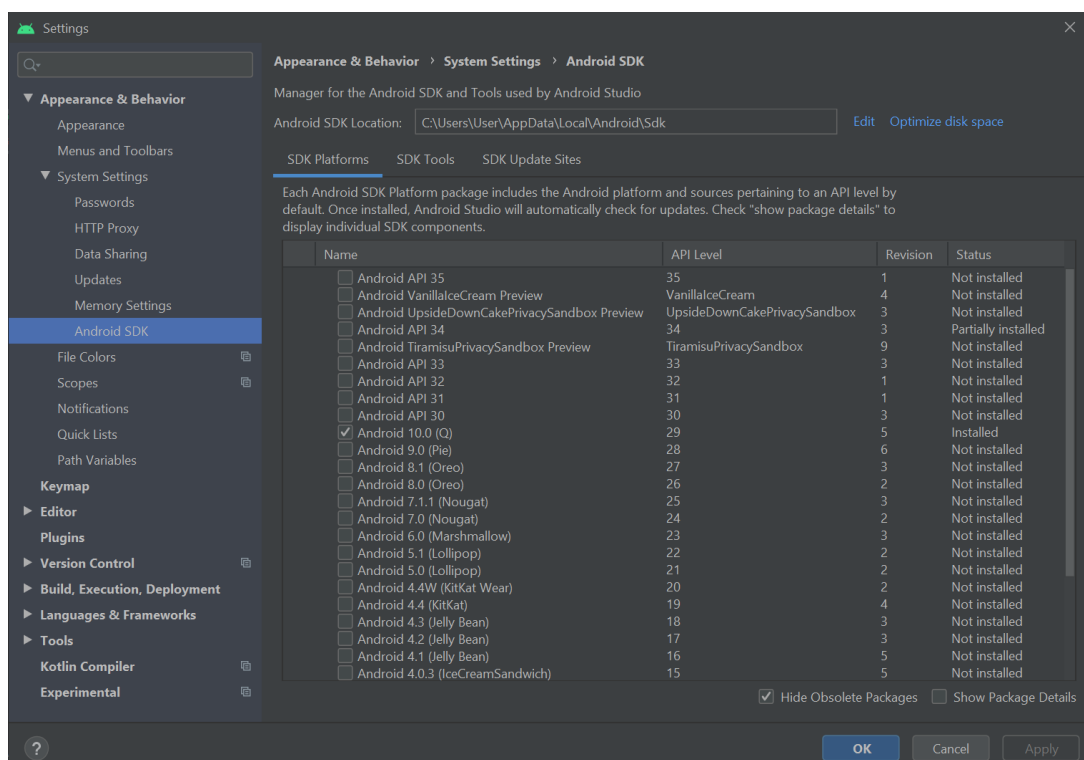


FIGURE 15. SDK Platforms settings

Navigate to the **SDK Tools** tab on the same page, check the **Show Package Details** option, and select both **Android SDK Build-Tools** and **Android SDK Platform-Tools** with the version **29.0.0**. Then click the “**Apply**” button. If you cannot find the requested version of the Android SDK Platform-Tools, you can download it from this link:

https://dl.google.com/android/repository/platform-tools_r29.0.0-windows.zip

If you need to download the requested Android SDK Platform-Tools from the link above, follow these steps after the download:

1. Delete the **platform-tools** folder located at `C:\Users\<your_username>\AppData\Local\Android\Sdk` on your computer.
2. Extract the downloaded file and copy its contents to the same path, replacing the deleted folder.

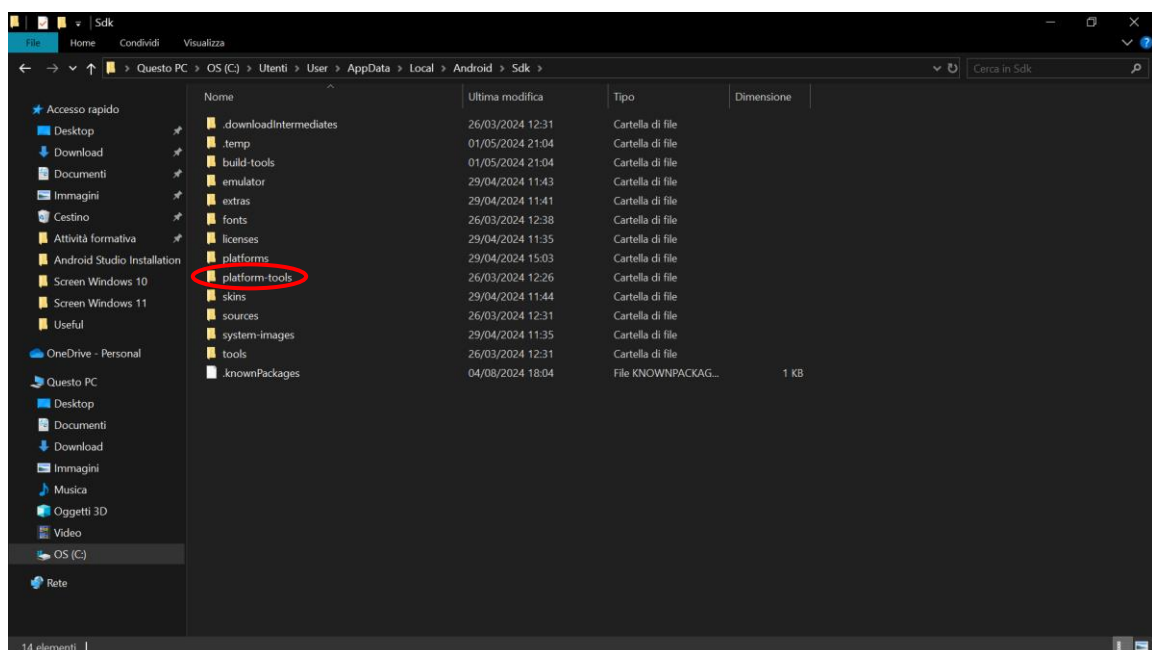


FIGURE 16. Folder that should be replaced

On Windows make sure you have installed:

1. Microsoft Visual Studio C++ 2010 x64 Redistributable Package. You can download the `vc_redist_x64.exe` package here:
<https://www.microsoft.com/en-us/download/details.aspx?id=26999>
2. Microsoft Visual Studio C++ 2013 Redistributable Package (x64). You can download the `vc_redist_x64.exe` package here:
<https://www.microsoft.com/en-us/download/details.aspx?id=40784>

SDK installation and configuration

Finally, we can download the Pepper SDK Plugin. To do this, go to the Plugins tab. You can access it from a created project by navigating to **File > Settings > Plugins** or from the **“Configure”** option on the Android Studio homepage. Enter “Pepper” in the search bar, in order to find Pepper SDK. Select Pepper SDK and click the **“Install”** button.

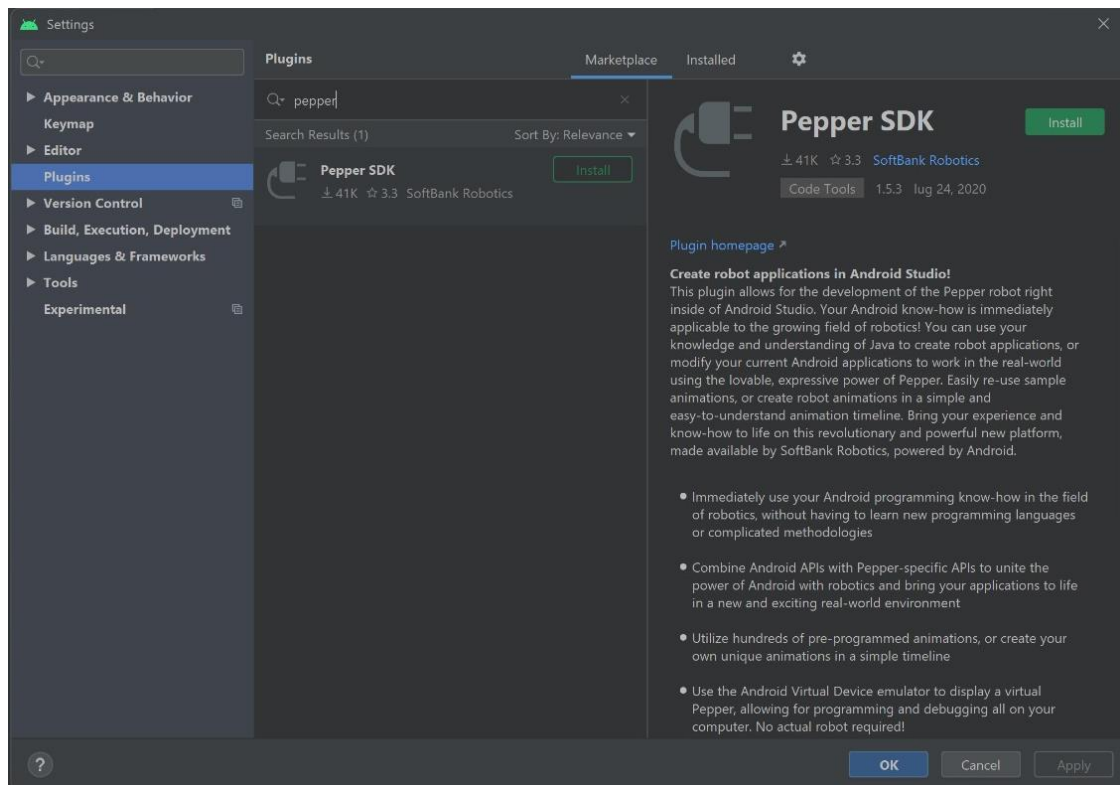


FIGURE 17. Plugin tab

When the installation is complete, restart Android Studio. Now, we can create a new project and learn how to use the most common tools.

On the Android Studio homepage shown in Figure 14, click on "Start a new Android Studio project" to create a new project. After selecting the project template, Android Studio will display a tab where you can configure the project. You can choose to program in either Kotlin or Java.

The minimum required version to work with the Pepper SDK is **API 23: Android 6.0 (Marshmallow)**. Then click **“Finish”**.

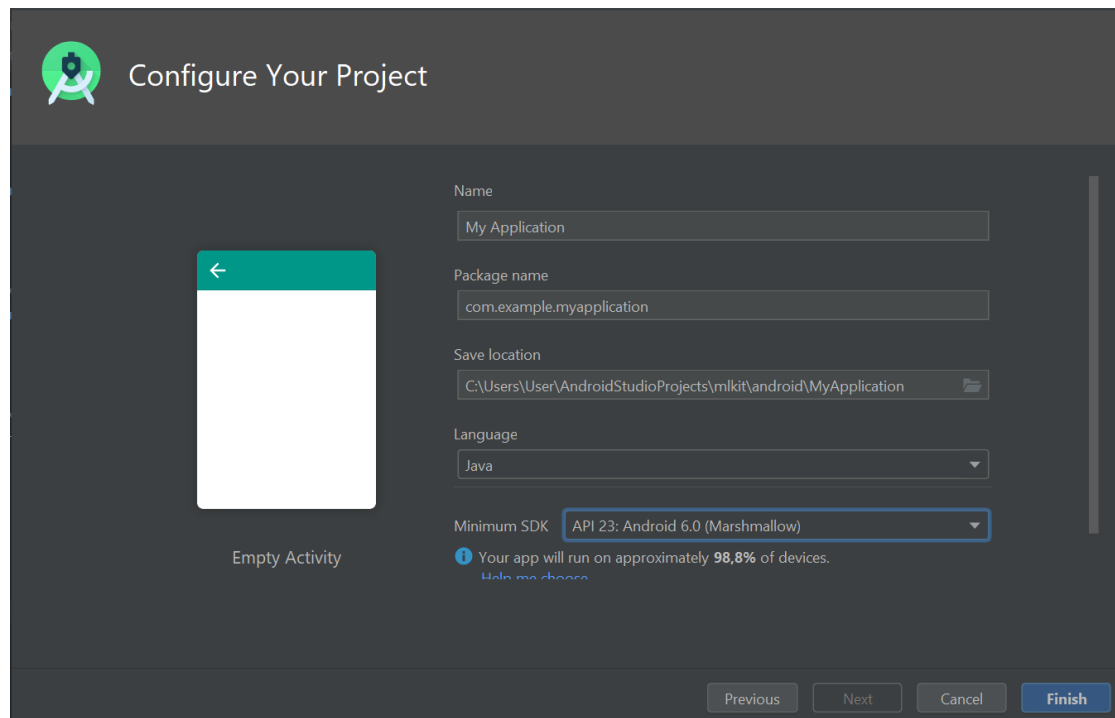


FIGURE 18. “Configure Your Project” tab

At this point, the project opens. Now go to **File > Project Structure > Modules** and ensure that both Source Compatibility and Target Compatibility are set to 1.8 (Java 8).

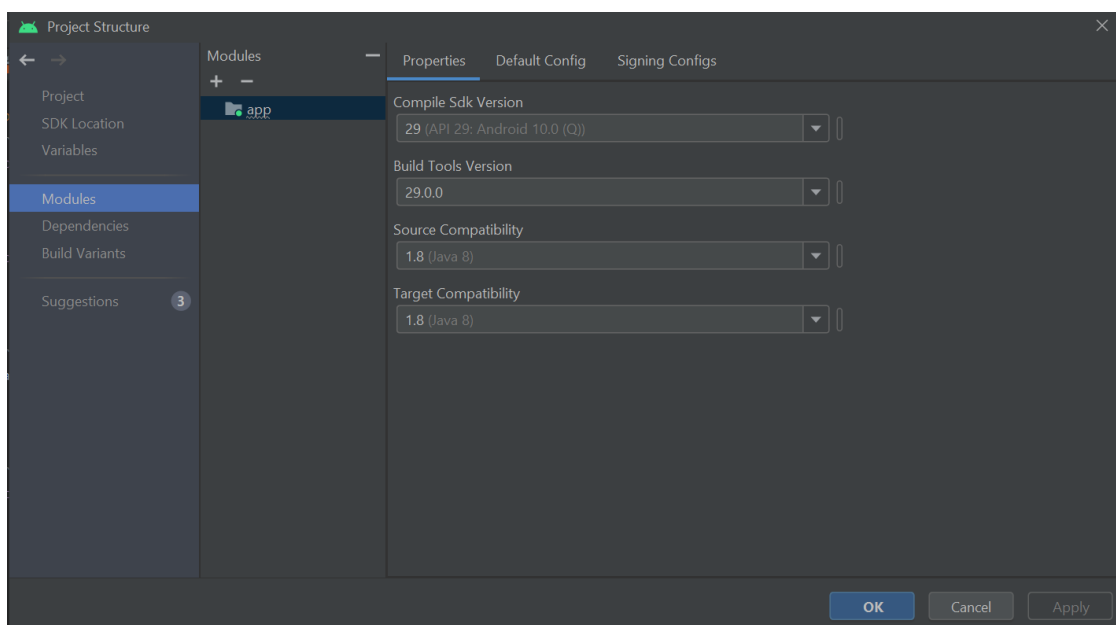


FIGURE 19. Additional project settings

Now that you have created the Android project, convert it into a robot application by selecting **File > New > Robot Application**.

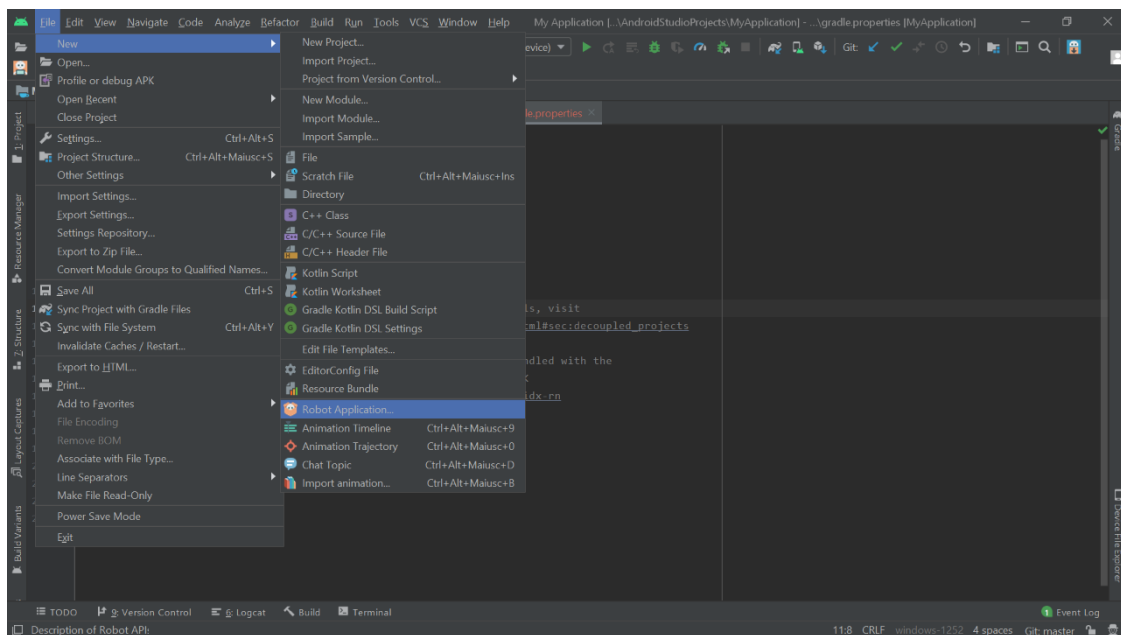


FIGURE 20. New robot application

You will be asked to select the API version you want to use. After making your choice, click the “OK” button.

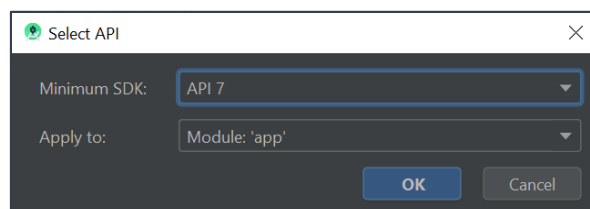


FIGURE 21. Alert that appears after the creation of a new robot application

As a result, the `robotsdk.xml` file is added to the `assets/robot` directory of the project. This file specifies the minimum Robot SDK version required for your project (the one you have selected).

Additionally, the `QiSDK` dependency has been added to your project’s `build.gradle` file, providing all the necessary functionalities to interact with Pepper. A `uses-feature` tag has also been added to your `AndroidManifest.xml`, indicating that your application uses Pepper and will be available only on the robot tablet.

When a project is created, an `activity_main.xml` file is also generated. By clicking on this file, you can access various tools provided by Android Studio to design your application's interface, including the Layout Editor. This tool allows developers to preview how their layouts will appear on different devices. To configure the preview for Pepper's tablet, set the Device for Preview to **Pepper 1.9 (1280 x 800, tvdpi)**.

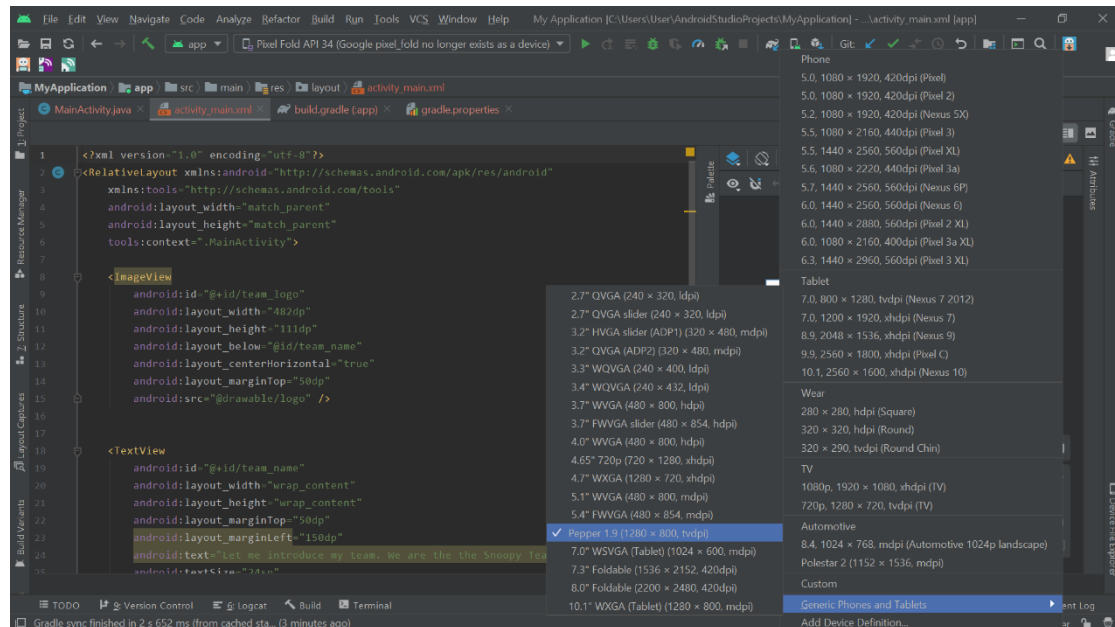



FIGURE 22. Layout Editor configuration

Our final step in setting up the environment is to download the tools for developing robot applications. To do this, choose **Tools > Pepper SDK > Robot SDK Manager**. There is also an option to enable a toolbar where you can directly access the SDK tools. To enable it, go to **View > Appearance > Toolbar**. The Robot SDK Manager corresponds to the  icon.

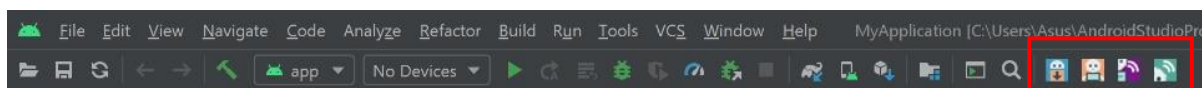


FIGURE 23. Toolbar with the Pepper SDK tools

Select the target API version for developing your robot application and click the **“Apply”** button. Clicking next to the API name (API 7, API 6, API 5, API 4 or API 3) will automatically select all the tools associated with that API version.

Note: On Windows 11, the best working version of the API is 6, as it prevents the Android Virtual Device from crashing.

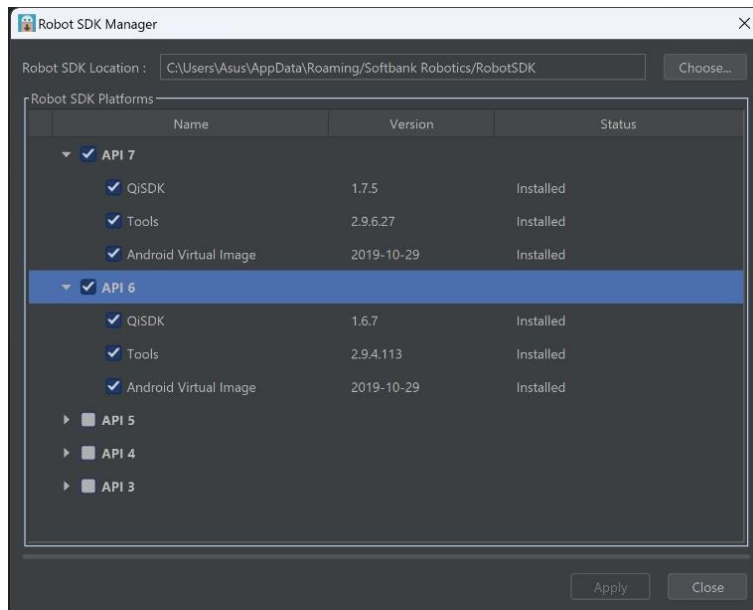




FIGURE 24. Robot SDK Manager

Now everything is ready to write our first Pepper application. But first, we need to check if the emulator works correctly. To start the emulator, click on the icon  or go to **Tools > Pepper SDK > Emulator**. Note: the emulator works correctly if both the Android Virtual Device and the Robot Viewer open.

To run an application on the emulator, click the  button.

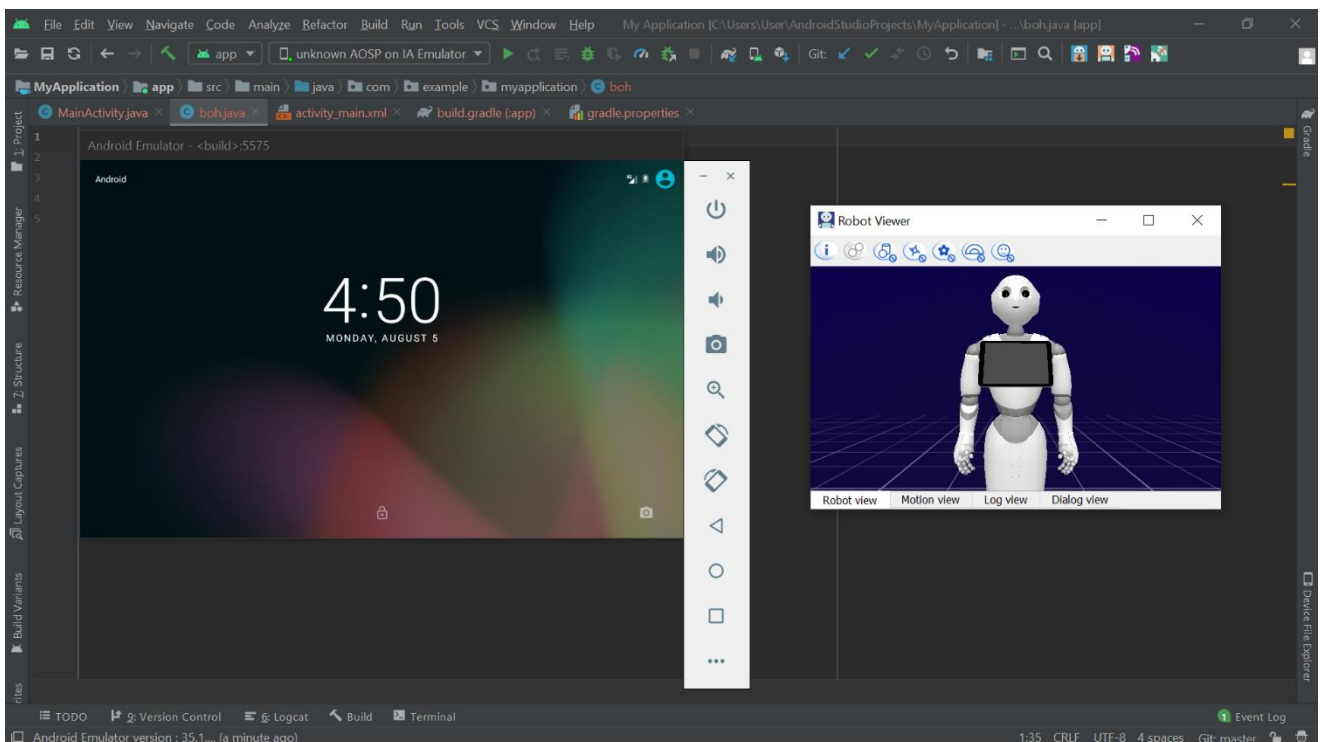




FIGURE 25. The emulator when it works correctly

As we can see, there are two other tools in the toolbar shown in Figure 19 (represented by the icons  and ) that are useful for connecting and running an application on a real robot. These tools can also be accessed by going to **Tools > Pepper SDK > Connect** and **Tools > Pepper SDK > Open Robots Browser**, respectively.

Before setting up the connection, we first need to set a password for the robot. To do this, follow these steps:

1. On Pepper's tablet, tap the six-dot icon to open the Android menu.
2. Tap the Settings icon to access Android Settings.
3. Select the "Advanced Settings" option.
4. Tap "Robot Password" to set a new password for the robot. The default password is "nao".



FIGURE 26. Android menu on the Pepper's tablet homepage

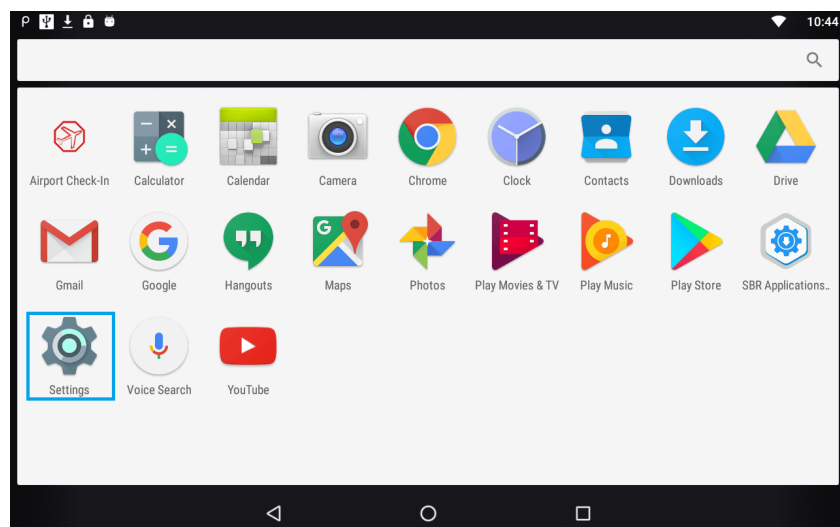



FIGURE 27. Android Settings

The robot needs to be connected to Wi-Fi to be detected by the Robots Browser. To access the Wi-Fi settings, you must go through the Android settings on Pepper's tablet. The steps are similar to those used to set the robot's password:

1. On Pepper's tablet, tap the six-dot icon to open the Android menu.
2. Tap the Settings icon to open Android Settings.
3. Tap the Wi-Fi icon  and configure the Wi-Fi from there.

At this point, open the Robots Browser, which displays all the detected robots. To connect to a robot, select it from the list. If the robot does not appear, adjust the port and IP address settings in the menu on the right.

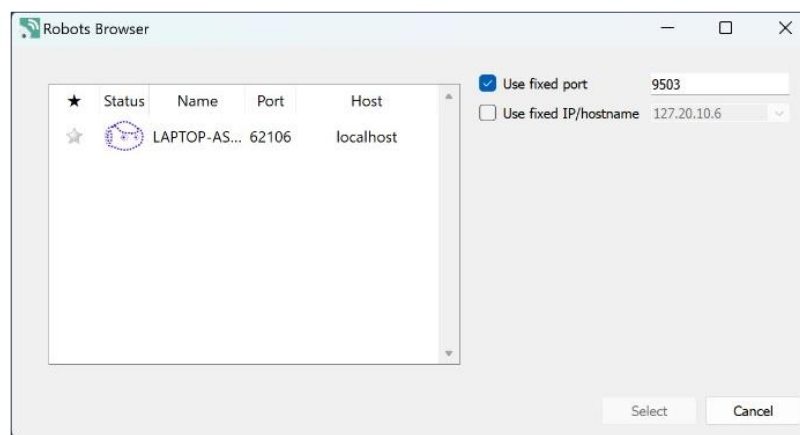


FIGURE 28. Robots Browser

You can obtain Pepper's IP address by pressing the robot's chest button once. Make sure Pepper is turned ON and connected to the network.

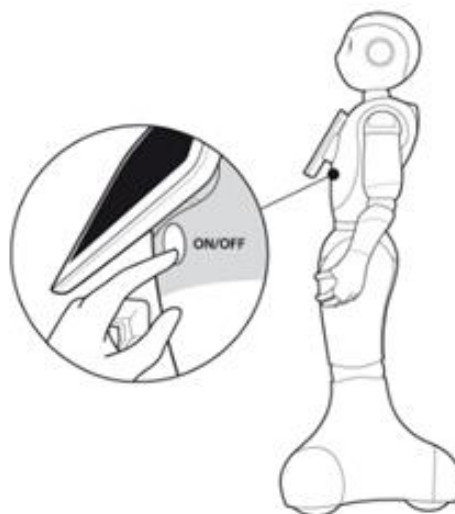




FIGURE 29. Pepper's chest button

When a physical robot is selected in the Robots Browser, a security alert appears, prompting you to enter the robot's password to connect. Enter the password and click the “OK” button.

Now we need to connect to the robot's tablet. Click on the icon  in the toolbar or navigate to **Tools > Pepper SDK > Connect**. Enter the IP address of the robot's tablet. **Note that the robot's IP address and the tablet's IP address are different.** To find the tablet's IP address, swipe down from the top of the robot's tablet screen to display the notifications, and look for the following logo: 

You can also choose whether you want the Robot Viewer to open automatically after the tablet connection is established.

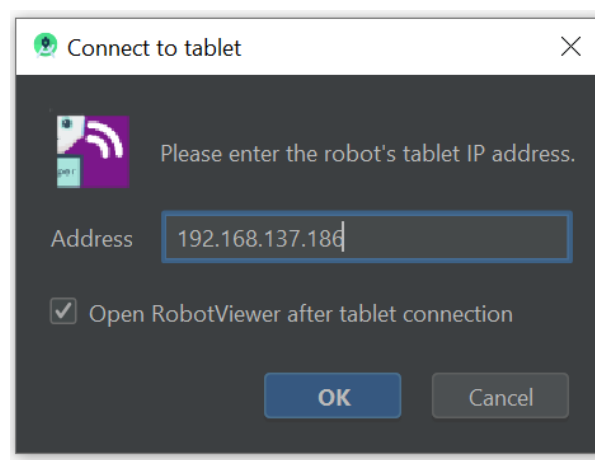


FIGURE 30. Connect to tablet

The recommended version of the Android Gradle Plugin is 3.6.1, and the suggested Gradle version is 5.6.4. You can update these versions by going to **File > Project Structure > Project**.

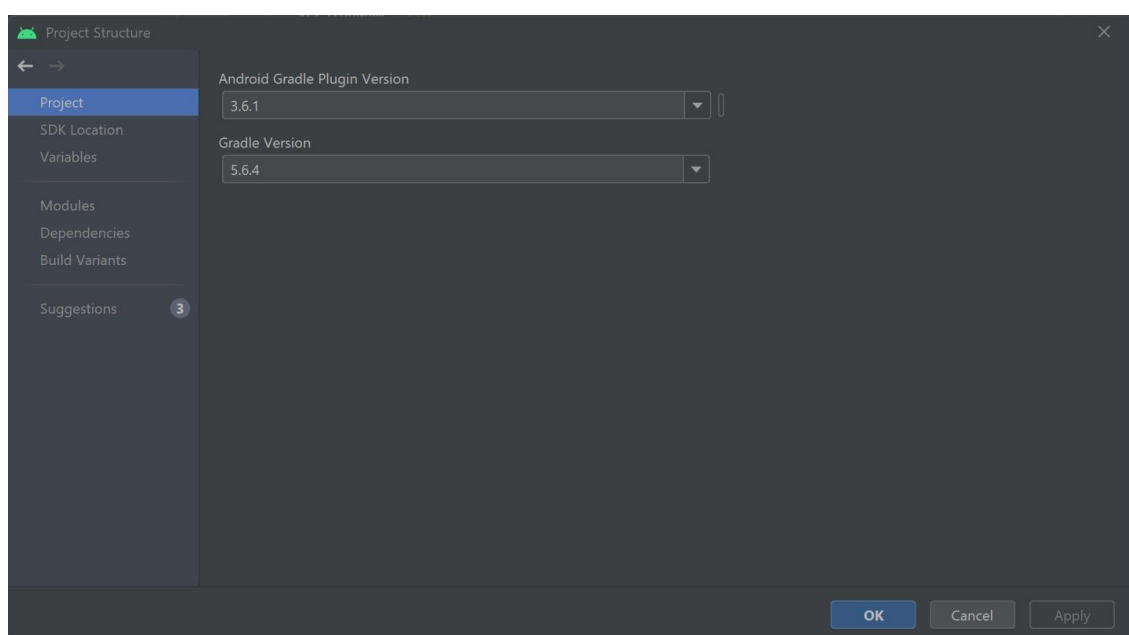


FIGURE 31. Recommended Gradle Versions

Troubleshooting and Windows 11 workaround

There is a common problem that causes the emulator to crash, which requires downgrading the Android Studio emulator to version 29.0.11. You can download it here:

<https://dl.google.com/android/repository/emulator-windows-5598178.zip>

Replace the emulator folder (located at `C:\Users<your_username>\AppData\Local\Android\Sdk`) with the emulator from the zip archive.

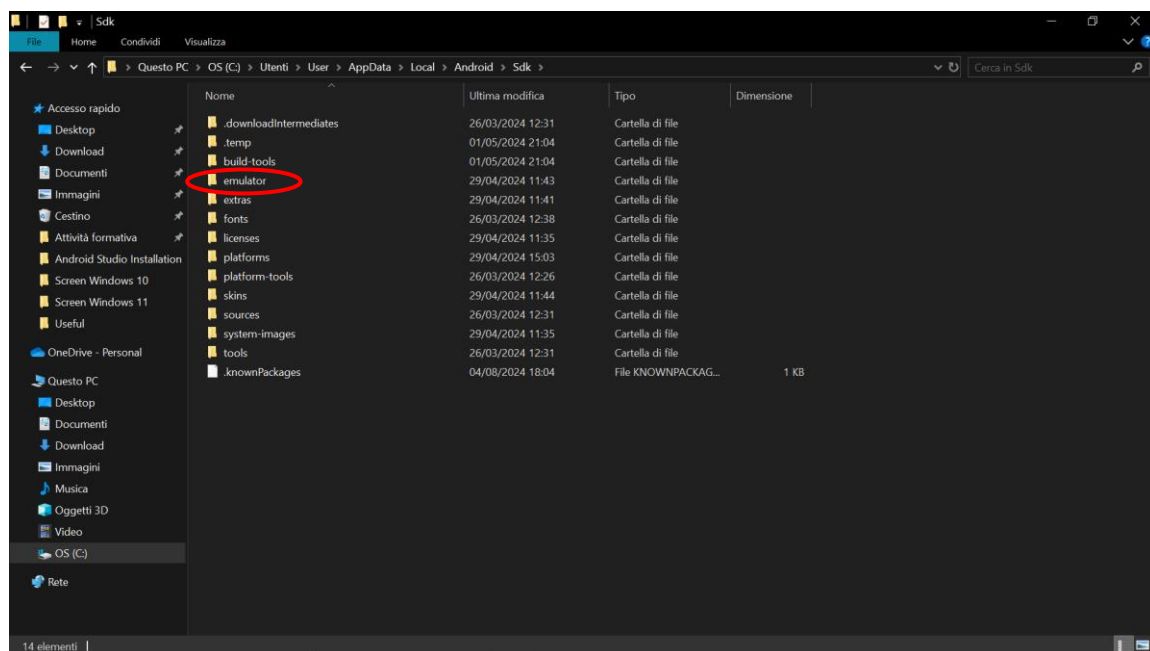


FIGURE 32. Folder that should be replaced

As mentioned earlier, Android Studio can experience unexpected crashes when trying to open common tools such as Animation Browser, Robot Viewer and Robots Browser on Windows 11. To address this issue, you can open these tools directly from the terminal or by clicking on their .exe files, which can be found at the following path on your computer:

`C:\Users\<your_username>\AppData\Roaming\SoftbankRobotics\RobotSDK\<API_version_you_installed>\tools\bin`

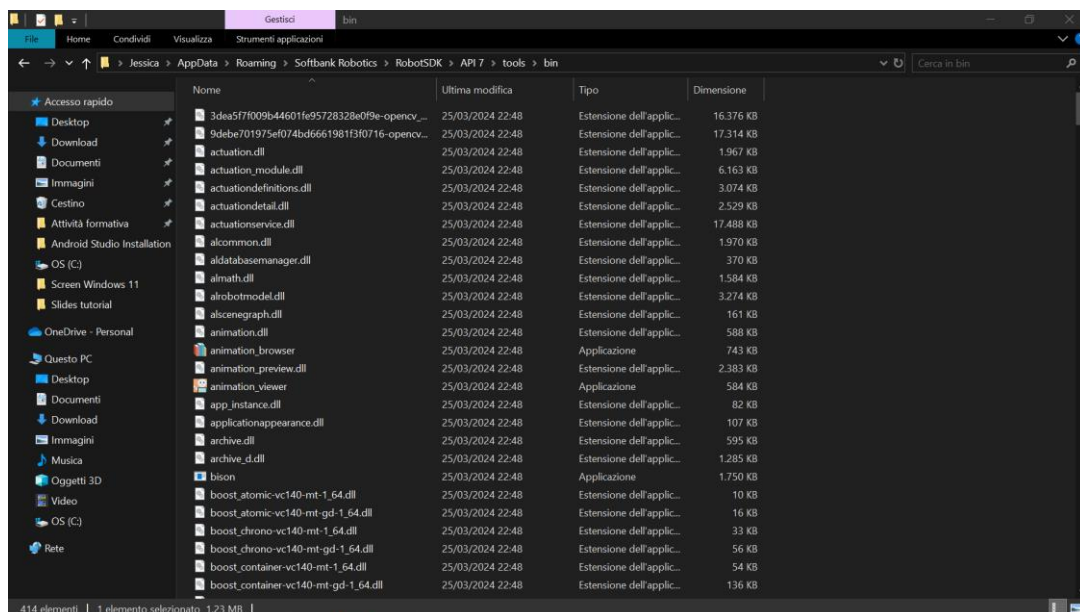



FIGURE 33. Folder containing the .exe files for the main plugin tools

To start the tools from the terminal, first open it and use the `cd` command to navigate to the appropriate directory. Once you're in the correct directory, simply type the name of the tool you want to open, such as “`animation_browser.exe`” or “`robots_browser.exe`”.

The Robot Viewer tool requires the Android Virtual Device (AVD) to be running. You can start it by clicking on the icon  or by going to **Tools > Pepper SDK > Emulator**. If Android Studio crashes after opening the AVD, simply reopen it.

After navigating to the appropriate path, enter the following command in the terminal: “`robot_viewer.exe localhost`”.

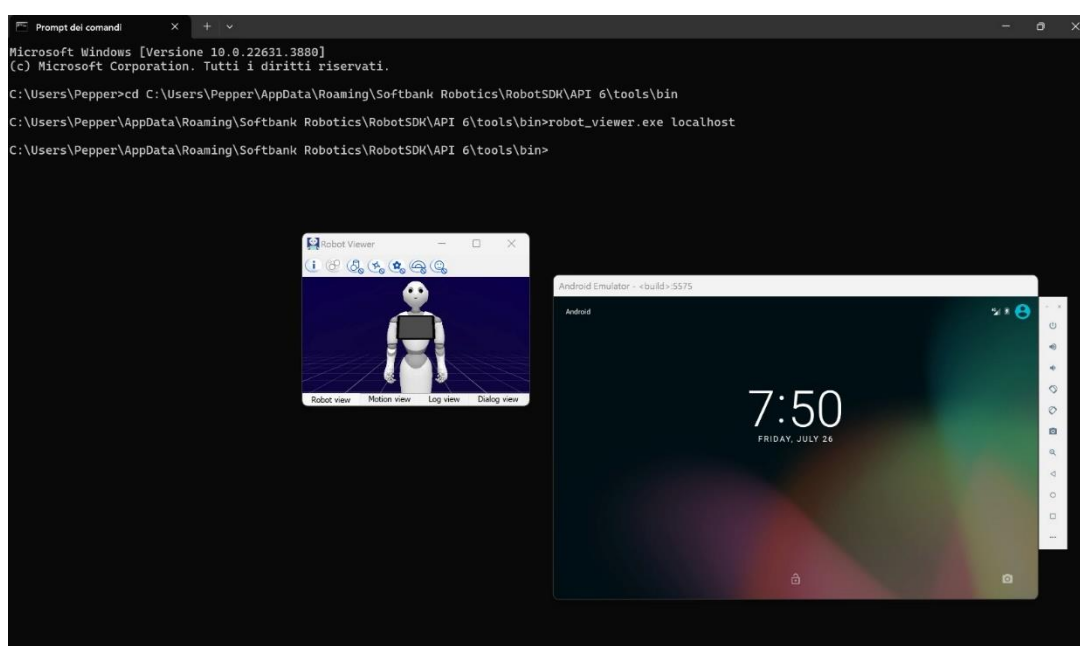


FIGURE 34. How to open Robot Viewer from the terminal

Creating an example application

Let's create our first Pepper application. Begin by creating a new project and robot application, following the steps outlined in the “SDK installation and configuration” chapter. Choose an empty activity to start your project and select the Java programming language. Two important files will be created: MainActivity.java and activity_main.xml.

The MainActivity.java file will be used to set the logic of our application, while the activity_main.xml file will be used to design the appearance of our application on Pepper's tablet.

The code in Figure 35 manages the robot's lifecycle events and serves as the starting point for every Pepper application.



```
public class MainActivity extends RobotActivity implements RobotLifecycleCallbacks {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Register the RobotLifecycleCallbacks to this Activity.
        QiSDK.register(this, this);
    }

    @Override
    protected void onDestroy() {
        // Unregister the RobotLifecycleCallbacks for this Activity.
        QiSDK.unregister(this, this);
        super.onDestroy();
    }

    @Override
    public void onRobotFocusGained(QiContext qiContext) {
        // The robot focus is gained.
    }

    @Override
    public void onRobotFocusLost() {
        // The robot focus is lost.
    }

    @Override
    public void onRobotFocusRefused(String reason) {
        // The robot focus is refused.
    }

}
```

FIGURE 35. Robot lifecycle events

Let's analyze the purpose of each method:

- **onCreate:** this lifecycle method is called when the activity is first created.
The line *super.onCreate(savedInstanceState);* invokes the parent class's onCreate method, ensuring that the activity is initialized according to the behavior defined in its base class.
The line *QiSDK.register(this, this);* registers the current activity (this) as a listener for robot lifecycle events with the QiSDK. As a result, MainActivity will start receiving callbacks related to the robot's lifecycle, as defined by the RobotLifecycleCallbacks interface.
- **onDestroy:** this lifecycle method is called when the activity is being destroyed.
The line *QiSDK.unregister(this, this);* unregisters the activity from receiving robot lifecycle events. This step is crucial for preventing memory leaks or unnecessary processing after the activity is destroyed.
The line *super.onDestroy();* calls the parent class's onDestroy method to ensure proper cleanup according to the base class's implementation.
- **onRobotFocusGained, onRobotFocusLost, onRobotFocusRefused:** these methods are part of the RobotLifecycleCallbacks interface and must be implemented because MainActivity implements this interface.
 - **onRobotFocusGained(QiContext qiContext)** is invoked when the robot gains focus, meaning it is now active and capable of interacting with MainActivity. The QiContext parameter provides the context needed for interaction with the robot.
 - **onRobotFocusLost()** is called when the robot loses focus, which might occur if another activity or application takes over.
 - **onRobotFocusRefused(String reason)** is triggered if the robot refuses to grant focus, potentially due to a reason specified in the reason parameter.

From this schema, we will begin developing our first simple application, which will display basic information on Pepper's tablet screen, such as your team's name, a logo, and an "Introduce the team" button. When the app starts, Pepper will greet the users. Once someone presses the " Introduce the team " button, Pepper will introduce the team with a spoken message.

Let's start by defining the interface for the application in the activity_main.xml file.

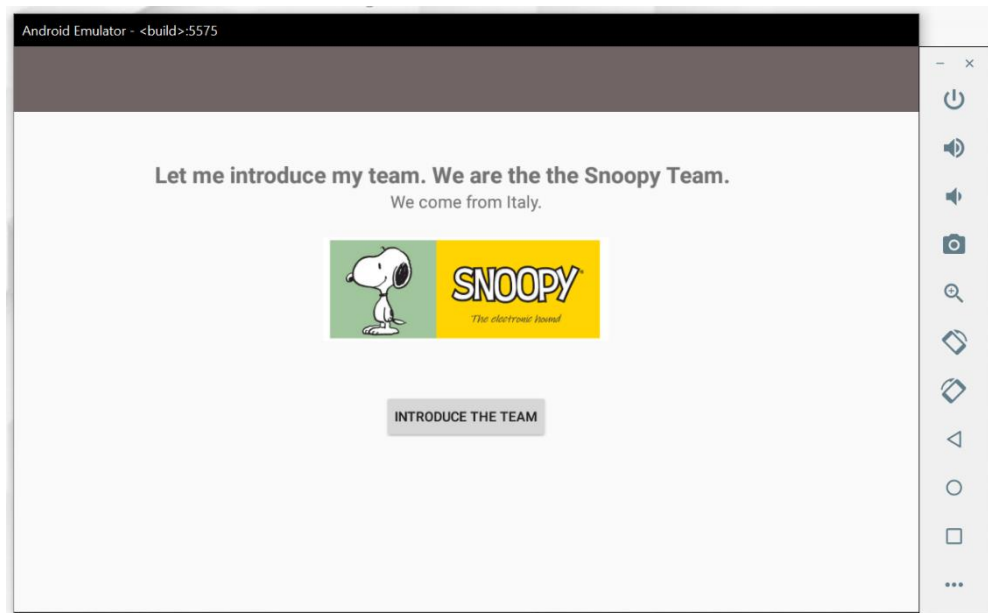


FIGURE 36. Our application goal interface

First, we define a `RelativeLayout`, which is a type of layout that arranges its children relative to each other or to the parent container. Setting both `layout_width` and `layout_height` to `match_parent` ensures that the layout will match the width and height of its parent.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

FIGURE 37. First activity_main.xml code snippet

To display an image, you need to upload it to the `res/drawable` folder in your Android Studio project. Then, you can visualize it using an `ImageView`. When creating the `ImageView`, specify the source for the image. Additionally, assign an ID to the `ImageView` so it can be used in the `MainActivity.java` file, and set the image's dimensions and position on the display.

```
<ImageView
    android:id="@+id/team_logo"
    android:layout_width="482dp"
    android:layout_height="111dp"
    android:layout_below="@id/team_name"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:src="@drawable/logo" />
```

FIGURE 38. Second activity_main.xml code snippet

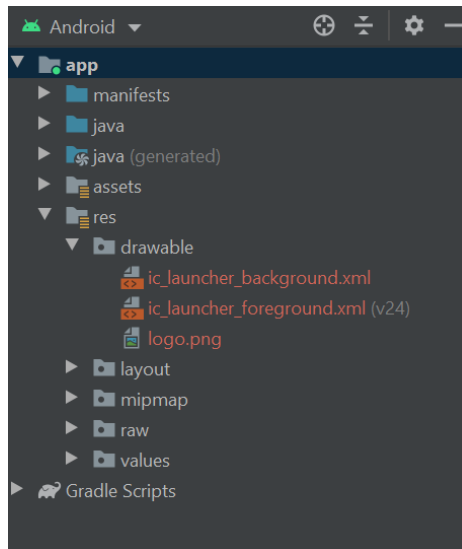


FIGURE 39. The path res/drawable

You can also add text to provide more information about yourself and your team using TextView components. You will need to set an ID for each TextView, and configure its position, dimensions, font and content accordingly.

```
<TextView
    android:id="@+id/team_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:layout_marginLeft="150dp"
    android:text="Let me introduce my team. We are the the Snoopy Team."
    android:textSize="24sp"
    android:textStyle="bold" />

<TextView
    android:id="@+id/team_affiliation"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_below="@id/team_name"
    android:textSize="18sp"
    android:text="We come from Italy." />
```

FIGURE 40. Third activity_main.xml code snippet

Finally, add a button labeled “Introduce the Team” to the interface. Be sure to set an ID for the button, as well as configure its dimensions, position and text.

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="300dp"
    android:text="Introduce the team" />
```

FIGURE 41. Final activity_main.xml code snippet

Now, we will focus on the application logic written in MainActivity.java. Starting with the robot event lifecycle, we will add the necessary components. Some fields are initialized and used to interact with both the robot and the UI elements of the activity. Specifically, the fields created are:

- *private Button button*; this field holds a reference to the button component from the layout created previously in the activity_main.xml file.
- *private QiContext qiContext = null*; qiContext is an object provided by the QiSDK that gives you access to the robot's capabilities and environment.
- *private Say sayAction1, sayAction2*; these fields hold instances of the Say action, which represents the robot speaking specific text.
- *private Animate saluteAction*; this field holds an instance of the Animate action, which represents an animation that the robot will perform.

```
// GUI button
private Button button;
// The QiContext provided by the QiSDK.
private QiContext qiContext = null;

private Say sayAction1, sayAction2;
private Animate saluteAction;
```

FIGURE 42. First MainActivity.java code snippet

The onCreate method is called when the activity is first created. It sets the content view to the layout defined in activity_main.xml. Additionally, it defines a click listener for the button. When the button is clicked, it triggers sayAction2 and saluteAction.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    button = (Button)findViewById(R.id.start_button);
    // Set the button onClick listener.
    button.setOnClickListener(v -> {
        sayAction2.async().run();
        saluteAction.async().run();
    });

    // Register the RobotLifecycleCallbacks to this Activity.
    QiSDK.register( activity: this, callbacks: this);
}
```

FIGURE 43. Second MainActivity.java code snippet

The onDestroy method remains unchanged from the original code that manages the robot's lifecycle events shown in figure 35.

The `onRobotFocusGained` method is called when the activity gains control over the robot. After storing the provided `QiContext` instance, we call the `initActions()` method. This method is not part of the robot's lifecycle events, its role is to initialize the actions used in the activity, such as speaking and animating. The line `sayAction1.async().run()` executes `sayAction1` asynchronously, making the robot say "Hello."

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
    // Store the provided QiContext.
    this.qiContext = qiContext;
    initActions();
    sayAction1.async().run();
}
```

FIGURE 44. Third MainActivity.java code snippet

The `onRobotFocusLost` method is called when the robot loses focus. When this happens, we clear the `qiContext` reference.

```
@Override
public void onRobotFocusLost() {
    // Remove the QiContext.
    this.qiContext = null;
}
```

FIGURE 45. Fourth MainActivity.java code snippet

Similar to the `onDestroy` method mentioned earlier, the `onRobotFocusRefused` method remains unchanged from the original code that manages the robot's lifecycle events shown in figure 35.

Now we can finally explore the last method of our Java class, the previously mentioned `initActions` method. In this method, `sayAction1` and `sayAction2` are created using `SayBuilder` to specify the text the robot should speak.

To use an animation, you first need to select it from the list shown in the Animation Browser. This tool can be accessed via **File > New > Import Animation**. If using this feature causes Android Studio to crash, refer to the workaround proposed in the previous chapter.

The Animation Browser categorizes animation files based on what they express, with classes such as "Action," "Basic Movements," "Emotion," and "Expressions." When you click on a specific animation file, the corresponding animation is shown on the virtual robot displayed on the right.

If you click the **"Select"** button, the animation is saved into the `res/raw` folder in your project.

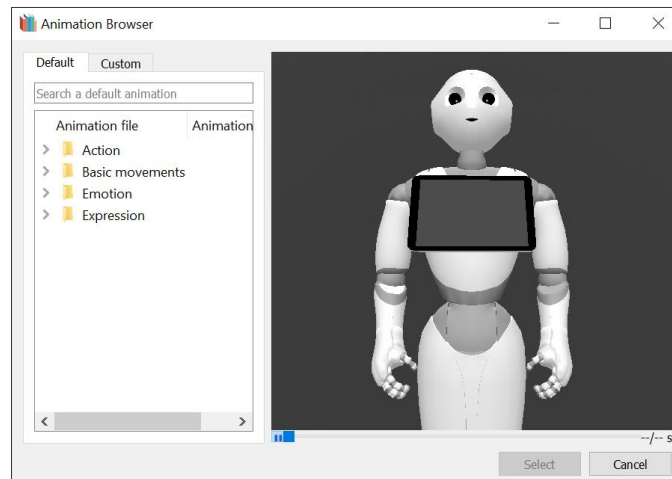


FIGURE 46. Animation Browser

You can then call an animation in a Java class using `AnimationBuilder` by accessing the resource with `R.raw.[name_of_the_animation]`, where `R` is a generated class in Android that references all the resources in your `res` folder, and `raw` is a subclass of `R` that contains references to resources in the `res/raw` directory of your project.

```
public void initActions() {
    // Create new say actions.
    sayAction1 = SayBuilder.with(qiContext) // Create the builder with the context.
        .withText("Hello") // Set the text to say.
        .build(); // Build the say action.

    sayAction2 = SayBuilder.with(qiContext) // Create the builder with the context.
        .withText("Let me introduce my team. We are the the Snoopy Team. We come from Italy.") // Set the text to say.
        .build(); // Build the say action.

    // Create an animation.
    Animation animation = AnimationBuilder.with(qiContext) // Create the builder with the context.
        .withResources(R.raw.salute_left_b001) // Set the animation resource.
        .build(); // Build the animation.

    // Create an animate action.
    saluteAction = AnimateBuilder.with(qiContext) // Create the builder with the context.
        .withAnimation(animation) // Set the animation.
        .build(); // Build the animate action.
}
```

FIGURE 47. Final MainActivity.java code snippet

Your first Pepper application is now complete!

If you encounter issues with the gradle compiling your code in Android Studio, a common problem is an incorrect version of `AppCompat` specified in your `build.gradle` file. To resolve this, add the following line to your dependencies:

implementation 'androidx.appcompat:appcompat:1.3.0'

Python SDK

NAOqi is the name of the main software that runs on the robot and controls it. [8]

The NAOqi framework is a programming framework used to program Pepper through the Python SDK. It answers to common robotics needs including: parallelism, resources, synchronization, events. This framework allows homogeneous communication between different modules (motion, audio, video), homogeneous programming and homogeneous information sharing.

It is possible to develop with NAOqi framework on Windows, Linux or MacOS.

A very important concept in the NAOqi framework is modules. Modules in the Pepper SDK are essentially pre-defined collections of functions and classes that allow you to interact with specific features or capabilities of the Pepper robot. Each module typically corresponds to a particular aspect of the robot's functionality, such as movement, speech, vision, or interaction with the environment. For example, you might have a module for controlling the robot's arms, another for managing its speech output and another for handling sensor data.

The APIs provided by the Python SDK allow us to interact with the robot's features through these modules.

The main modules provided are:

- **Core Modules:** these are a set of essential modules that are always available. The most important ones include `ALMemory`, which handles memory storage and retrieval, allowing you to store and access data such as sensor readings and event triggers, and `ALBehaviorManager`, which manages behaviors and allows you to start, stop, or check the status of various behaviors.
- **Motion Modules:** these modules allow control of the robot's movements, including walking, head movements, and joint angles.
- **Audio Modules:** the key modules include `ALTextToSpeech`, which generates speech from text, enabling Pepper to speak; `ALSpeechRecognition`, which processes spoken commands and can be used to recognize and interpret human speech; and `ALAudioPlayer`, which plays audio files for purposes like background music or sound effects.
- **People Perception Modules:** these modules analyze the people around the robot. The most important is `ALFaceDetection`, which handles face recognition and detection, enabling Pepper to recognize and interact with people.

- **Vision Modules:** the most important of these is ALVideoDevice, which provides access to the robot's cameras and enables the capture of images.
- **Sensor Modules:** these modules enable the robot to perceive and interact with its environment by accessing data from various sensors, including touch, motion, and environmental sensors. Key sensor modules include ALTouch, which manages Pepper's touch sensors located on its head and hands, allowing detection of when and where Pepper is being touched. ALSonar handles the sonar sensors, which detect objects at a distance by measuring sound waves. Pepper's torso houses these sonar sensors, helping it detect obstacles and gauge distances.

Use ...	To ...
ALAutonomousLife	Maintain the robot life cycle and manage the launching of activities
ALBehaviorManager	Start and stop behaviors
ALConnectionManager	Manage connection to a network and its configuration
ALMemory	Get and insert data for every other module to use
ALModule	Create your own modules
ALNotificationManager	Manage notifications
ALPreferenceManager	Read and save robot settings
ALResourceManager	Handle resources
ALStore	Retrieve applications from the <i>Apps 2.1 Store</i>
ALSystem	Manage the robot system
ALUserSession	Manages the state of active users, and the bindings to their data
ALTabletService	Load web application, play videos, and manage the tablet itself
ALWorldRepresentation	Store long term data about detected objects in a spatially structured database
PackageManager	Manage packages: installation, uninstallation

FIGURE 48. Core modules

SDK installation

The installation process discussed in this chapter has been tested only on Windows 10 and Windows 11. We do not guarantee it will work on other operating systems. In our tests, we did not observe significant differences between the two installations.

Unfortunately, the most recent version of Python that supports the Pepper Python SDK is 2.7. We recommend uninstalling any other versions of Python currently installed on your machine. If you want to continue using a newer Python version without creating conflicts with the 2.7 version required for this guide, consider installing Anaconda, which typically comes with the latest Python version.

We tested version 2.7.18, which is available for download at this link:

<https://www.python.org/downloads/release/python-2718/>

We chose the download file named [Windows x86-64 MSI installer](#).

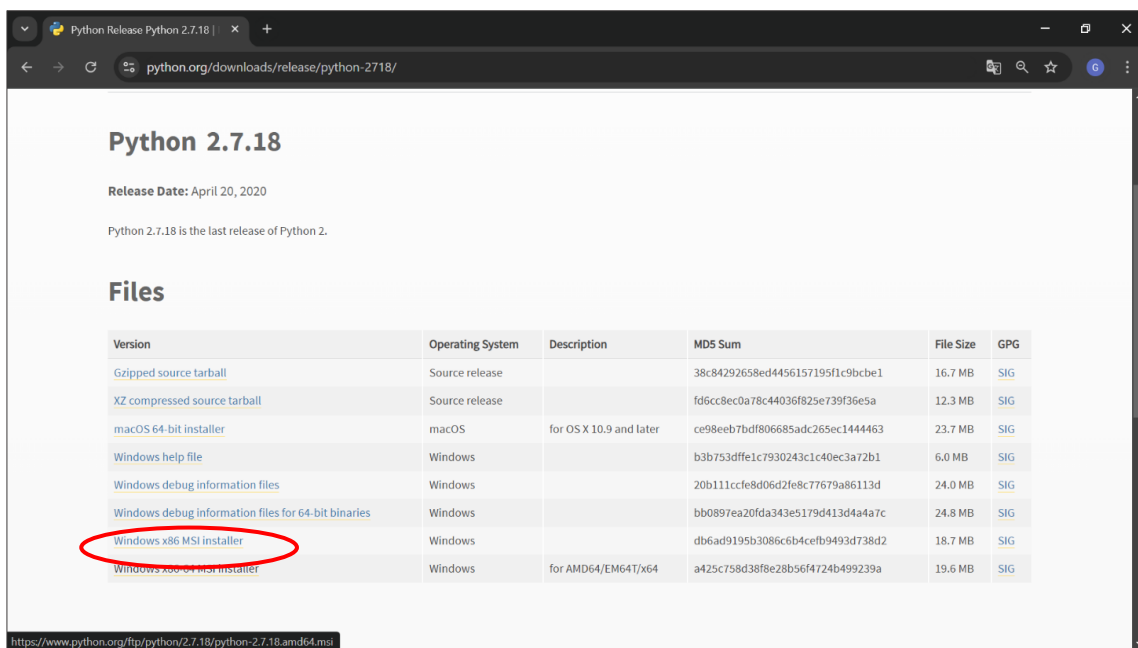


FIGURE 49. Python 2.7.18 download page

Open the setup file to start the installation process. We kept all the default options. At one point, you'll be asked if you want to add Python to your PATH environment variable. You should select this option to make Python 2.7.18 accessible directly from your terminal.

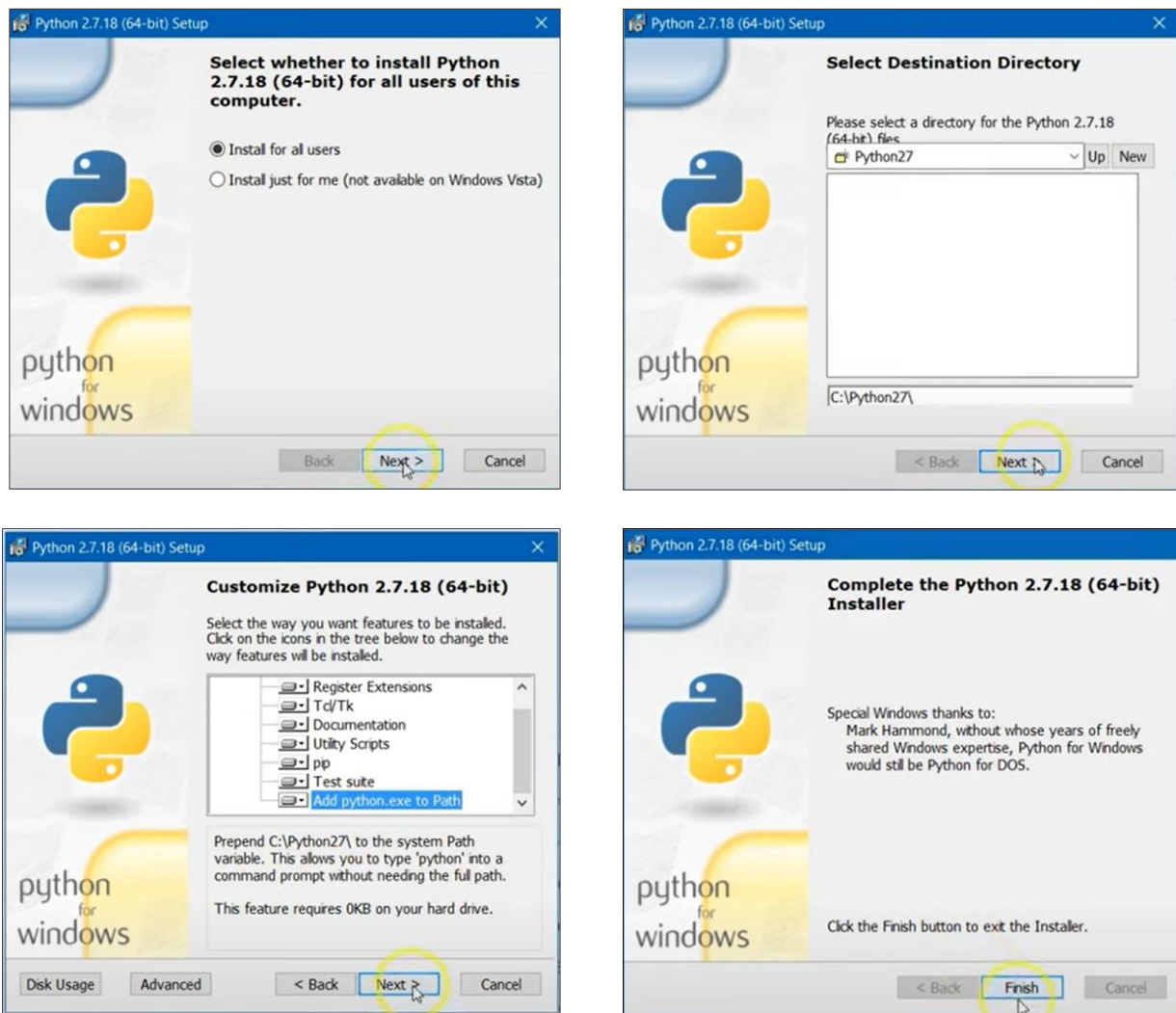


FIGURE 50. Python 2.7.18 installation process

In the examples we provide, we will use the IDLE code editor, which comes bundled with Python, making it easily accessible. To verify that the installation was successful, search for IDLE on your computer and open it. Then, type a simple arithmetic operation, such as $2 + 2$, to ensure that everything is working correctly.

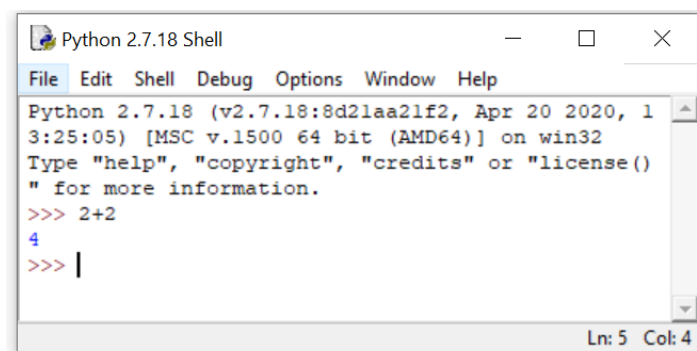


FIGURE 51. What you see if Python is correctly installed

Now that Python is correctly installed, we need to install the Pepper SDK. You can download it from the following link:

<https://support.aldebaran.com/support/solutions/articles/80001018812-nao-6-downloads>

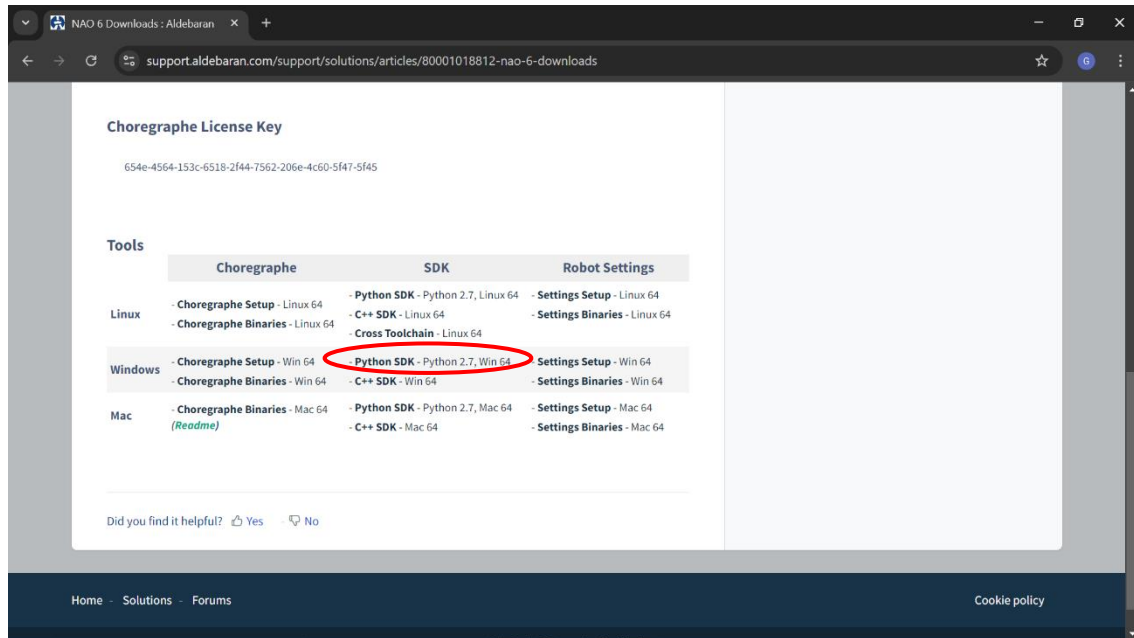


FIGURE 52. Python SDK download file

Unzip the downloaded file and copy its contents to the directory C:\Python27\Lib\site-packages on your computer.

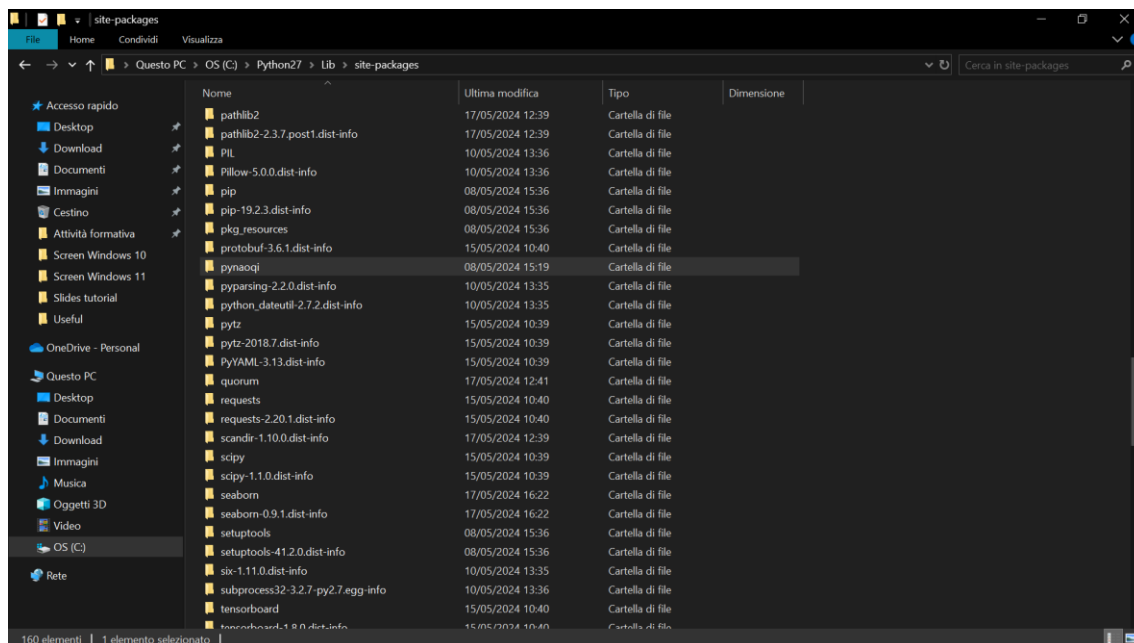


FIGURE 53. Pynaoqi copied in the sit-packages folder

Now you need to set an environment variable to enable Python to correctly locate the library. The variable should be named PYTHONPATH and should point to C:\Python27\Lib\site-packages\pynaoqi\lib.

To access and modify environment variables on a Windows computer, follow these steps:

1. Open the Start Menu by clicking the Start button or pressing the Windows key on your keyboard.
2. Type "Environment Variables" or "Edit the system environment variables" into the search bar.
3. Click on the "Edit the system environment variables" result that appears.
4. In the System Properties window that opens, click on the "Environment Variables..." button near the bottom.
5. Under "User variables", click **"New"** to create a new variable.
6. Enter the name of the variable as PYTHONPATH and set its value to C:\Python27\Lib\site-packages\pynaoqi\lib.
7. Click **"OK"** in all windows to save the changes and close them.

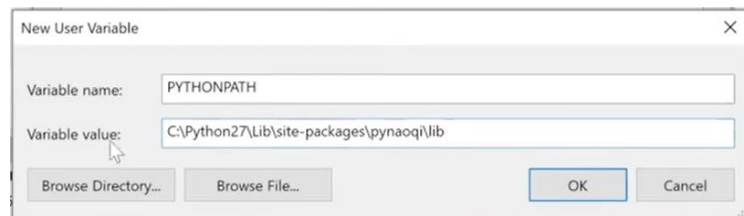


FIGURE 54. PYTHONPATH user variable

Now, we need to verify that the library is correctly installed. To do this, open IDLE and try typing "import naoqi". If you do not receive any errors, your installation is successful, and you can start using the Pepper Python SDK.

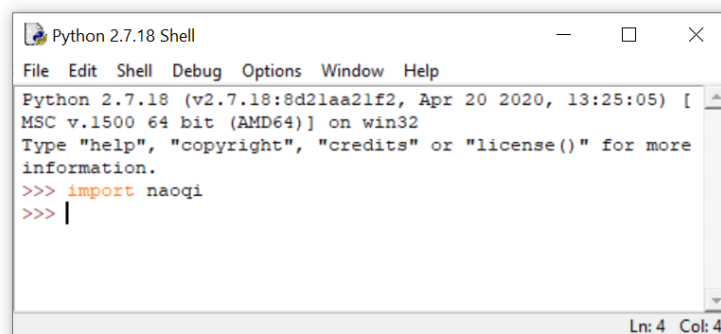


FIGURE 55. What you see if the SDK is correctly installed

Communication with a real robot and Robot Viewer

If the operating system running on Pepper is NAOqi 2.5, connecting to it with the Python SDK is very straightforward. Ensure that your robot is ready to use. Start your preferred editor, in our case IDLE, and enter the following code:

```
from naoqi import ALProxy
tts = ALProxy("ALTextToSpeech", "<IP of your robot>", 9559)
```

The first line imports the ALProxy class from the naoqi module. ALProxy is a class that allows you to create a proxy object to communicate with different services on the robot. In this case, the service specified is ALTextToSpeech, which converts text into speech and enables Pepper to speak.

Replace "<IP of your robot>" with the actual IP address of your robot to direct ALProxy to the correct location on the network. The port number 9559 is the default for NAOqi services, so it is typically used unless you have configured your robot to use a different port.

If you don't know its IP address, press the robot chest button. Save the file as a Python file and run it.

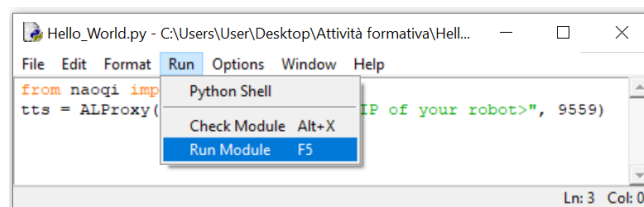


FIGURE 56. How to run a Python file with IDLE

If you have already installed QiSDK on your machine, you can also connect the Python SDK to the Robot Viewer to test your code before running it on a real robot. First, start the robot emulator from Android Studio, as described in the chapter “SDK installation and configuration”.

To communicate with the Robot Viewer using the code we discussed earlier, replace "<IP of your robot>" with “localhost”.

You can connect to a robot running the NAOqi 2.9 operating system, even if it is Android-based. To do this, use a private API gateway available at “tcps://<robot_ip>:9503”. This gateway exposes a subset of officially supported services that can be used by clients. These services are part of the QiSDK, typically accessed through Android applications. However, they are also available from Python if you connect directly to this gateway.

To access the gateway, you need to authenticate. As discussed in the “SDK installation and configuration” chapter, connecting to a real robot with QiSDK requires authentication using the robot's password. To authenticate, add these lines to your Python code each time you connect to a real robot:

```
class Authenticator:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def initialAuthData(self):
        return {'user': self.username, 'token': self.password}

class AuthenticatorFactory:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def newAuthenticator(self):
        return Authenticator(self.username, self.password)

session = qi.Session()
logins = ("nao", "<robot password>")
factory = AuthenticatorFactory(*logins)
session.setClientAuthenticatorFactory(factory)
session.connect("tcps://<<IP of your robot>>:9503")
```

FIGURE 57. Authentication code

The Authenticator class is responsible for storing and returning authentication data. Its *initialAuthData(self)* method returns a dictionary containing the authentication data with the keys "user" and "token."

The AuthenticatorFactory class is responsible for creating instances of the Authenticator class. Its *newAuthenticator(self)* method creates and returns a new Authenticator object using the stored username and password.

By executing the line *factory = AuthenticatorFactory(*logins)*, an instance of AuthenticatorFactory is created using the provided username and password. The **logins* syntax unpacks the tuple into two separate arguments (username and password). Finally, the session is configured to use the AuthenticatorFactory for authentication. This means that whenever authentication is required, the session will request a new Authenticator from the factory.

Example code

Starting with the code in figure 57, we will develop a program that enables our Pepper robot to capture images.

First, we need to import some essential libraries: `qi`, which is part of the NAOqi SDK; `Image` from the PIL library, used for handling and manipulating images and `vision_definitions`, another module from the NAOqi SDK that contains predefined constants related to the robot's vision system (e.g., resolutions, color spaces).

If you encounter the error “`ImportError: No module named PIL`”, it means you need to install the Pillow library. You can do this by typing `pip install pillow` in your terminal.

```
import qi
from PIL import Image
import vision_definitions
```

FIGURE 58. First code snippet

Now, copy the code for the authentication part from Figure 57 and proceed.

We need to retrieve the video service (`ALVideoDevice`), which allows interaction with the robot's cameras. We set the resolution to 720p (1280x720 pixels) and the color space to RGB, with a frame rate of 5 frames per second.

```
video_service = session.service("ALVideoDevice")
resolution = vision_definitions.k720p
color_space = vision_definitions.kRGBColorSpace
fps = 5
```

FIGURE 59. Second code snippet

Now we need to subscribe to the camera with a unique client name (`"python_client"`), camera index, resolution, color space and fps.

The camera index specifies which camera the robot uses. Index 0 corresponds to the top camera, while index 1 corresponds to the bottom camera. Both of these are identical video cameras located on the robot's forehead. Indexes 2 and 3 may have different outcomes depending on the robot model (1.8 or 1.8a) [3].

```
video_client = video_service.subscribeCamera("python_client", 0, resolution, color_space, fps)
```

FIGURE 60. Third code snippet

The line `video_service.getImageRemote(video_client)` instructs the robot to capture an image from the camera. The returned image data includes the width, height and raw pixel data. This raw pixel data is then converted into a byte array. The image is then displayed using the default image viewer.

```
for _ in range(10):
    nao_image = video_service.getImageRemote(video_client)
    if nao_image:
        image_width = nao_image[0]
        image_height = nao_image[1]
        array = nao_image[6]
        image_string = bytearray(array)
        image = Image.frombytes("RGB", (image_width, image_height), bytes(image_string))
        image.show()
    break
```

FIGURE 61. Fourth code snippet

Finally, we unsubscribe from the camera to clean up and release resources.

```
video_service.unsubscribe(video_client)
```

FIGURE 62. Final code snippet

References

[1] QiSDK documentation:

<https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/index.html>

[2] Pepper SDK documentation:

<http://doc.aldebaran.com/2-5/dev/python/index.html>

[3] Pepper video and depth sensors:

http://doc.aldebaran.com/2-5/family/pepper_technical/video_overview.html

[4] Pepper specs:

<https://www.softbank.jp/en/robot/>

[5] Comparison of Pepper's OS versions:

<https://www.aldebaran.com/developer-center/articles/OSComparison/index.html>

<https://www.aldebaran.com/developer-center/articles/assets/Datasheet.pdf>

[6] Choregraphe Suite:

<http://doc.aldebaran.com/2-4/software/choregraphe/index.html>

[7] Pepper SDK, JetBrains Marketplace:

<https://plugins.jetbrains.com/plugin/8354-pepper-sdk/versions>

[8] NAOqi Framework:

<http://doc.aldebaran.com/2-1/dev/naoqi/index.html>

[9] Python SDK connection to NAOqi 2.9:

<https://github.com/aldebaran/libqi-python/issues/22#issuecomment-1941618222>