Github project example

Scenario 1, how to achieve active interaction perception between Pepper robot and human?

Business Scenario:

One of the features of the Pepper robot is that it can actively detect and sense customers around it who are interested in interacting with the robot in real business scenarios, and can actively approach customers to interact with it.

To implement a feature similar to the above scenario, you can refer to the following project:

pepper-interaction-sample

Project Description:

The project procedures show the use of QiSDK to complete the Pepper robot interaction process of the smallest unit , can be seen as a starting point for the development of robot interaction applications using Pepper . The project within the interaction state machine to control the interaction process.

Interaction State Machine

Pepper alternates between three states:

- Idle state, when Pepper is idle (in this example, Pepper does not perform any action)
- Attract state, when Pepper sees a person from a distance (in this example, Pepper will initiate 'Hey')
- Engaged state, when the person is in close proximity to Pepper (in this example, Pepper initiates a Chat conversation).

In addition, Pepper's current status is displayed at the bottom of the tablet and can be updated as the status changes.

Extended State Machine

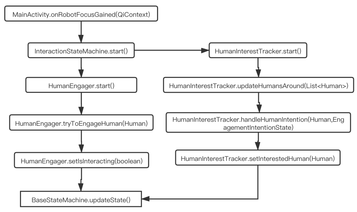
The project contains a generic state machine framework to add states, change Actions, and modify the tablet UI.

Each state can be associated with the following:

- A Behavior object that will run while the associated state is active and complete when the state completes (e.g. Pepper initiates speech and chat)
- A Fragment, which can be displayed on the Pepper tablet whenever the associated state is active.

Behavior and Fragment can be added or changed on their own, as well as new states can be added, and these can be modified in the InteractionStateMachine class, which contains the core logic of the process.

A brief flowchart of the project is shown below:



Based on the flowchart it is possible to focus on two tool classes for sensing humans (these can also be reused in other projects to achieve a clear flow of interactions)

- HumanEngager, which automatically creates an EngageHuman on activation via RecommendedHumanToEngage.
- HumanInterestTracker, which provides 'human' objects of interest via HumansAround on activation.

Scenario 2: Is there a library of interaction actions that includes Pepper's vivid and natural interactions?

Business Scenario:

Pepper, as an intelligent receptionist, can attract customers through natural anthropomorphic movements and simulate human movements during interaction with customers to maximise human-machine friendly interaction when it senses customers nearby.

If you need to realise functions similar to the above scenario, you can refer to the following projects:

Project one:

pepper-core-anims-master

Project Information:

The types of Anims included under this project are:

- Pepper active greeting
- Pepper emotion display
- Pepper resume standing position
- Pepper pause, interval action

Some of these Anims can be viewed in AndroidStudio -> New -> import animations->Default.

You can view or edit different Anim effects after importing Anim files into AndroidStudio (import method: AndroidStudio -> New -> import animations-> Custom).

Project two:

pepper-orientation-anims

Project Information:

The types of Anims included under this project are:

- Pepper points in different directions with his fingers

All of the Anim effects can be viewed in the project/img folder.

You can import Anim files in AndroidStudio (import method: AndroidStudio -> New -> import animations-> Custom), view or edit different Anim effects.

Scene 3: How to control Pepper's movement by using the joystick?

Scene example:

In a stage scene, the joystick of the gamepad is used to control Pepper's movement or rotation in different directions so that Pepper can 'walk' to the right position on the stage and start hosting or performing an entertainment programme.

If you want to implement something similar to the above scenario, you can refer to the following project: pepper-gamepad

Project description:

The project uses Bluetooth to provide communication between the gamepad and the Pepper robot. Before you start using it, make sure that the Bluetooth input device is connected to the Pepper's Bluetooth (turn on Bluetooth in the Pepper's tablet and actively connect the input device). And, close the power hatch (charging port cover) or Pepper will not move.

You can directly clone the project to open it within AndroidStudio and run it directly on Pepper.

You can also refer to the following steps to integrate this feature into your project.

Step 1:

Import the library pepper-gamepad into your project.

Step 2 (Make sure your project is already a robot project):

Disable BasicAwareness in onRobotFocusGained and instantiate RemoteRobotController

```
override fun onRobotFocusGained(qiContext: QiContext) {
    val basicAwarenessHolder = HolderBuilder.with(qiContext)
        .withAutonomousAbilities(AutonomousAbilitiesType.BASIC_AWARENESS)
        .build()

    basicAwarenessHolder.async().hold().thenConsume {
        when {
            it.isSuccess -> Log.i(TAG, "BasicAwareness held with success")
            it.hasError() -> Log.e(TAG, "holdBasicAwareness error: " + it.errorMessage)
            it.isCancelled -> Log.e(TAG, "holdBasicAwareness cancelled")
        }
    }

    remoteRobotController = RemoteRobotController(qiContext)
}
```

Step 3:

Implement a method to get the value back from the Bluetooth input device (the left lever enables Pepper to move in different directions and the right lever enables Pepper to rotate).

```
private fun getCenteredAxis(event: MotionEvent, device: InputDevice, axis: Int): Float {
    val range: InputDevice.MotionRange? = device.getMotionRange(axis, event.source)

    // A joystick at rest does not always report an absolute position of
    // (0,0). Use the getFlat() method to determine the range of values bounding the joystick axis

    range?.apply {
        val value = event.getAxisValue(axis)

        // Ignore axis values that are within the 'flat' region of the joystick axis center.
        if (Math.abs(value) > flat) {
            return value
        }
    }
    return 0f
}
```

Step 4:

Rewrite the onGenericMotionEvent method of Activity and get the value passed back when operating the joystick through different TAGs of MotionEvent and finally implement Pepper motion update through remoteRobotController.updateTarget.

```
override fun onGenericMotionEvent(event: MotionEvent): Boolean {
    // Add null protection for when the controller disconnects
    val inputDevice = event.device ?: return super.onGenericMotionEvent(event)

    // Get left joystick coordinates
    val leftJoystickX = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_X)
    val leftJoystickY = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_Y)

    // Get right joystick coordinates
    val rightJoystickX = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_Z)
    val rightJoystickY = getCenteredAxis(event, inputDevice, MotionEvent.AXIS_RZ)

    if (::remoteRobotController.isInitialized) {
        thread {
            remoteRobotController.updateTarget(leftJoystickX, leftJoystickY, rightJoystickX, rightJ
        }
    }
    return true
}
```