Raspberry Pi Emulator

Embedded and Pervasive Systems Summer Term 2020

Introduction

The program serves as a substitution for the group project in Embedded and Pervasive Systems to emulate the behaviour and sensors of a Raspberry Pi. It allows students to implement the logic and communication without having access to the real hardware and realize their projects.

It is created using python 3 and py qt for the user interface with zeroMQ serving as the communication method between multiple Raspberry Pis.

Installation and Start

To install the project simply import it into a new project in a programming environment of your choice (e.g. pycharm) and install the requirements.txt file (*pip install -r requirements.txt*) or do it manually for each library if you prefer.

After installation, run the *Main.py* file to load both Raspberry Pis, all sensors and user interfaces (see below for further details).

At startup each component is placed at screen centre, to permanently change its location call the corresponding *set_window_location* function.

Sensors

Each Sensor comes with a set() function allowing to bypass the UI and set values directly e.g. for testing automation.

The following sensors are included in the emulator:

- Accelerometer
    - Calculates the current acceleration given a duration in milliseconds, the movement speed in meters per second and the axes along the movement (x, y, z). If multiple axes are selected they have an equal acceleration. Orientation and earth gravitation are not considered.
    - Get function returns a triple of accelerations for x, y, z axes in meters per square second.
    - Set function takes three Booleans – one for each axis -, the duration and movement speed.
- Brightness
    - Returns the brightness that is set in candela.
    - Get function returns current brightness level as float
    - Set function takes a numeric value to set brightness in candela.
- Button
    - Allows a physical button press for a given duration in milliseconds.
    - Get function returns the current state of the button.
    - Set function takes the duration of the button press in milliseconds
- Camera
    - Allows to capture live video or load a local video
    - Local video source needs to be set in init of camera
    - Mode can be changed in init and ui

    - *get_image* returns the current image of the video/live stream
- Green/Red LED
    - Sets a green or red LED on or off.
    - on function activates the led
    - off function deactivates the led
- Gyroscope
    - Allows to change the orientation of the Raspberry Pi (roll, pitch, yaw)
    - Get function returns a tipple of roll, pitch yaw as integers
    - Set functions takes three integers to set the orientation
- Humidity
    - Allows to measure current humidity.
    - Get function returns the current humidity in percent.
    - Set function takes a numerical value to set the current humidity.
- Infrared
    - Emulates an infrared barrier that can be physically interrupted.
    - Get function returns the current state.
    - Interrupt function interrupts the barrier (object passing between sensors).
    - Close function closes the barrier (object left sensors).
- Servo
    - Emulates a servo motor going up or down.
    - on function starts the servo in the given direction (string “up”, “down”).
    - off function stops the servo.
    - To change direction of the running servo call on function with the opposite direction.
- Temperature
    - Allows to measure the current temperature in degrees Celsius.
    - Get function returns current temperature as float.
    - Set function takes a numerical value to set temperature.
- Display
    - Allows to display some text
- Weight
    - Returns a set weight in grams
- Distance
    - Measures the distance of some moving object in mm for a given step size in mm either towards or away from the device (from > to = moving closer; vice versa moving away; step size must always be positive)
    - Adjust time.sleep() value in movement_thread to slow down the process
    - If you set stepsize to 0 the distance will be constantly set to the from value
- Microphone
    - Samples input from a microphone; shows a green circle when audio input is detected
    - Function get_stream returns the entire audio stream
    - Function get_audio_data returns an array of all recorded audio data so far
- Audio
    - Loads an audio file and replays it
    - Function play_audio plays the pre loaded audio file
- RFID Sensor
    - Allows to add known tags and check if a present tag is known to the system
    - To add presence of a tag put the tag id in the corresponding edit field; to remove its presence set the field empty

- Gesture Sensor
    - Allows to emulate a swiping gesture that lasts 1 second
    - get_direction returns the current gesture or an empty string if no gesture is performed at that moment

Raspberry Pi

The Raspberry Pi consists of a simple user interface showing all connected sensors as well as a sender and receiver for communication between another Raspberry Pi (see below).

The a*ctiveSensors* dict holds all sensors that are currently running, leave that untouched.

The *intialSensors* dict holds all sensors that should be loaded at startup for the chosen Raspberry Pi respectively, add as many sensors as you need and give them appropriate names (name: type).

sampling_thread is run at startup and continuously samples data from sensors that are both connected and requested. As in the real Raspberry Pi, you have to subscribe to the data streams of each sensor by accessing the objects from the dictionary (*activeSensors[name]*) and call their respective *get* functions (see below).

Communication

Communication between two Raspberry Pi is established via zmq's client server principle (https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/client_server.html)

Each Raspberry Pi implements both, a client and server, for sending and receiving messages respectively. To send a message call the *sendMessage()* function with the message to be sent. Receiving is done in the *receiver_thread()*.

As the client server principle is based on requests and responses you first need to send a request from either Raspberry Pi to the other and then respond using the receiver's socket. If, however, you want to bundle some messages you need to store incoming messages and send blank responses as you cannot send further requests before receiving the corresponding response.

If you want to use another messaging protocol (like MQTT) that is fine. However, you must not call functions of the other Rasperry Pi directly, always use a broker for communication!

Main File

The main file loads a simple user interface showing both Raspberry Pi that are connected and starts them at startup with all their sensors respectively. If you want to define some sequences of sensor values you can do this best here by calling the *getSensor()* function of the Raspberry Pi you want to alter the sensor for and set the values.

Example

Some example code is provided where Raspberry Pi 1 loads a gesture sensor and a display and Raspberry Pi 2 loads a gyroscope. When the "swipe up" gesture on Raspberry Pi 1 is pressed it sends a message to Raspberry Pi 2 asking for its current orientation which responds with the current gyroscope levels. These levels are received in Raspberry Pi 1 again and set on the display.

However, as the "swipe up" gesture is performed for one second, multiple requests and responses will be sent. Here it does not matter but in your program you might have to account for that.