Gazebosc

Github page: https://github.com/hku-ect/gazebosc

- What is Gazebo?
- Python Actors
- Mediapipe Python actors

What is Gazebo?

> Gazebosc is the Swiss armyknife for OSC data and OSC capable programs.

Gazebosc is a high level actor model programming environment. You can visually create actors and author them using a nodal UI. Actors are running concurrently and can be programmed sequentially using C, C++ or Python. Actors communicate using the OSC serialisation format.

Gazebo is a node based programming environment used to connect different hardware and software using the OSC protocol. It is an easy way to receive data from different sources on one central computer and then forward it to as many other OSC devices you want.

Main Site:

https://github.com/hku-ect/gazebosc

Prebuilt binaries

https://pong.hku.nl/~buildbot/gazebosc/

Features:

- You can use Gazebo to receive Optitrack NATNET motion capture data and convert it to OSC data that you can send to any OSC capable device/program on the network to any OSC capable device/program on the network (windows only)
- You can use Gazebo to receive data from HTC VIVE Trackers using the vive system and send this dat as OSC data
- You can use Gazebo to receive MIDI data and convert it to OSC data that you can send to any OSC capable device/program on the network
- You can control DMX light from OSC
- You can play modfiles (audio) that send OSC events based on the music.
- You can create your own Gazebo actors in Python to do different things with data

Python Actors

Adding new nodes

This documentation is based on the Gazebosc github: https://github.com/hku-ect/gazebosc

Easiest method of adding a new node is using Python. You'll need to have a Gazebosc build with Python.

- In Gazebosc create a new Python actor: (right mouse click - Python)
- Click on the edit icon, a text editor will appear
- Paste the following text in the texteditor and click save

```
class MyActor(object):
    def handleSocket(self, addr, msg, type, name, uuid, *args, **kwargs):
    print("received OSC message {} {}".format(addr, msg))
        return ("/MyActorMsg", [ "hello", "world", 42] )
```

This is just the most basic actor which responds to incoming messages. A template you can use for a full feature actor is as follows:

```
class actor(object):
    def __init__(self, *args, **kwargs):
        self.timeout = 1000         # Use this timeout value for when you need recurring handle
                                    # Set to -1 to wait infinite (default)

    def handleApi(self, command, *args, **kwargs):
        print("The API command is {} and its arguments is {}".format(command, args))
        return None

    def handleSocket(self, address, data, *args, **kwargs):
        print("The osc address is {} and its data is {}".format(address, data))
        return ("/myreturnaddress", ["hello", 3, 2, 1])

    def handleTimer(self, *args, **kwargs):
        # This is a timed event, use it as you need
        print("My timed event with type: {}, name: {}, uuid: {}".format(args[0], args[1], args[2
        return ("/mytimedreturn", ["hello", 1, 2, 3])

    def handleCustomSocket(self, *args, **kwargs):
        # We'll explain this in the future
        return ("/myreturnaddress", ["hello", "world"])

    def handleStop(self, *args, **kwargs):
```

```
        # We are shutting down
        print("Bye bye from {}".format(args[1]))
```

Save this file as actor.py as the filename needs to equal the class name!

Node Lifetime

Once a node has been created, it goes through the following steps:

- **Construction**
  - if performed from loading a file, also passes and Deserializes data
- **CreateActor** (this is called after instantiation to preserve polymorphic response)
- **Threaded Actor events**
  - Init: actor has been created, and can be used to do threaded initializations (see OSCListener example)
  - Message: actor has received a message
  - Callback: actor has received a timeout (timed event, probably scheduled by calling the SetRate function)
  - Stop: actor has been stopped and threaded resources can be cleaned up (see OSCListener example)
- **Destruction**

Construction & Destructions

During these phases, you can prepare and clean up resources used by the class. Examples include UI char buffers for text or values (see PulseNode).

CreateActor

This GNode function can be overriden to perform main-thread operations once the actor has been created. Primary use-case at this time is calling the *SetRate* function (an API-call, which must be called from the main thread) to tell the node to send timeout events at a set rate (x times per second).

Threaded Actor events

Throughout the lifetime of the actor, the GNode class will receive events, and pass these along to virtual functions. Override these functions to perform custom behaviours (see above description for which events there are). Important to note is that this code runs on the thread, and you should not access or chance main-thread data (such as UI variables). For such cases, we are still designing *report functionality* (copied thread data that you can then use to update UI, for instance).

Destruction

When deleting nodes or clearing sketches, the node instance will be destroyed and its actor stopped.

Mediapipe Python actors

Mediapipe Python actors

We are currently in the progress of developing several Gazebo Python actors based on the Mediapipe framework. These are based on the 0.10.9 version of the Mediapip library

- The base repository can be found here: https://github.com/hku-ect/PoseTrackActor
- Development is being done here:
  https://github.com/ikbenmacje/PoseTrackActor/tree/mediapipe0.10.9

PoseTrackActor

The PoseTrackActor is a python actor for gazebo using the Mediapipe framework to do body pose recognition. In order for the actor to work you need to install mediapip 0.10.9 and requirements in de folder of the python actor. You also need to save the stage to that directory so that the directory with the python modules becomes the working directory form which the actor is loaded.

See for more instructions here:
https://github.com/ikbenmacje/PoseTrackActor/tree/mediapipe0.10.9

FaceDetectionActor

The face detection actor is a Python actor based on the Mediapipe framework that track the face.

Installing mediapipe 0.10.9 with python3.9 on OSX

Before you start creating your virtual environment and installing mediapip do this first:

```
export SYSTEM_VERSION_COMPAT=0
```

To see why look here:
https://github.com/AnyLifeZLB/FaceVerificationSDK/blob/main/install_newest_mediapipe_on_macos.md