Synchronous and asynchronous modes on dynamic control

Ricardo A. Téllez, Diego Pardo and Cecilio Angulo

Technical University of Catalonia, Spain

grup de recerca en enginyeria del coneixement

Outline

- Description of the task
- Implementation in real Aibo using libUrbi
  - Synchronous implementation
  - Asynchronous implementation
- Future work and conclusions

Description of the task

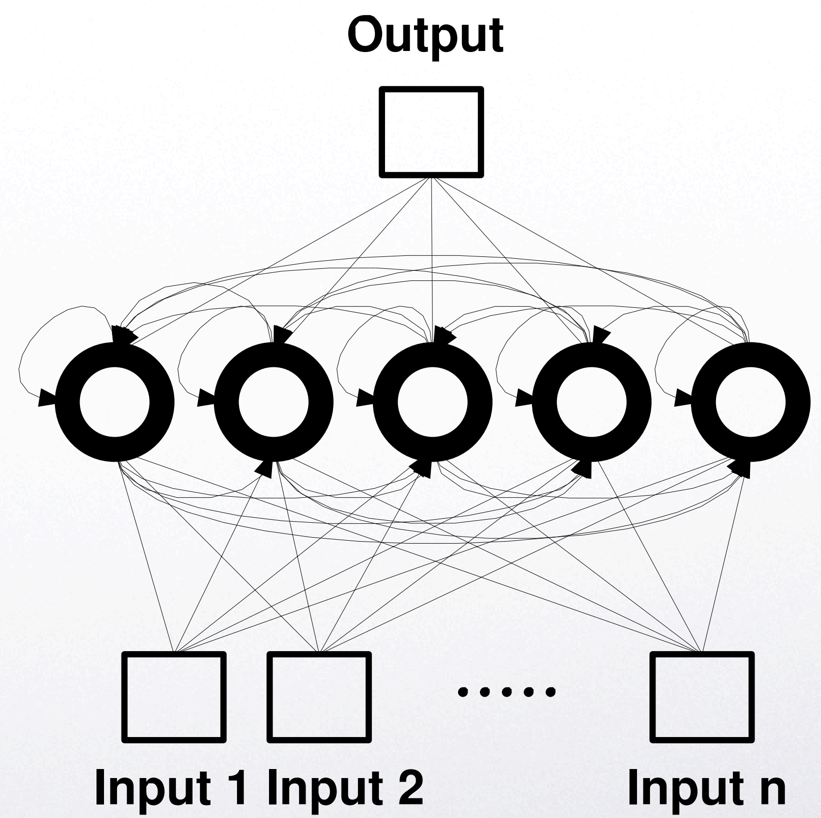
Main Goal

- Make real Aibo walk using distributed neural nets
  - First step: evolve the nets using a simulator

  - Second step: transfer simulator results to real robot using libUrbi

Neural control

- Continual Time Recurrent Neural Nets used
- One net per sensor and actuator (24)
- Actuators' net output encodes joint velocity at any time-step



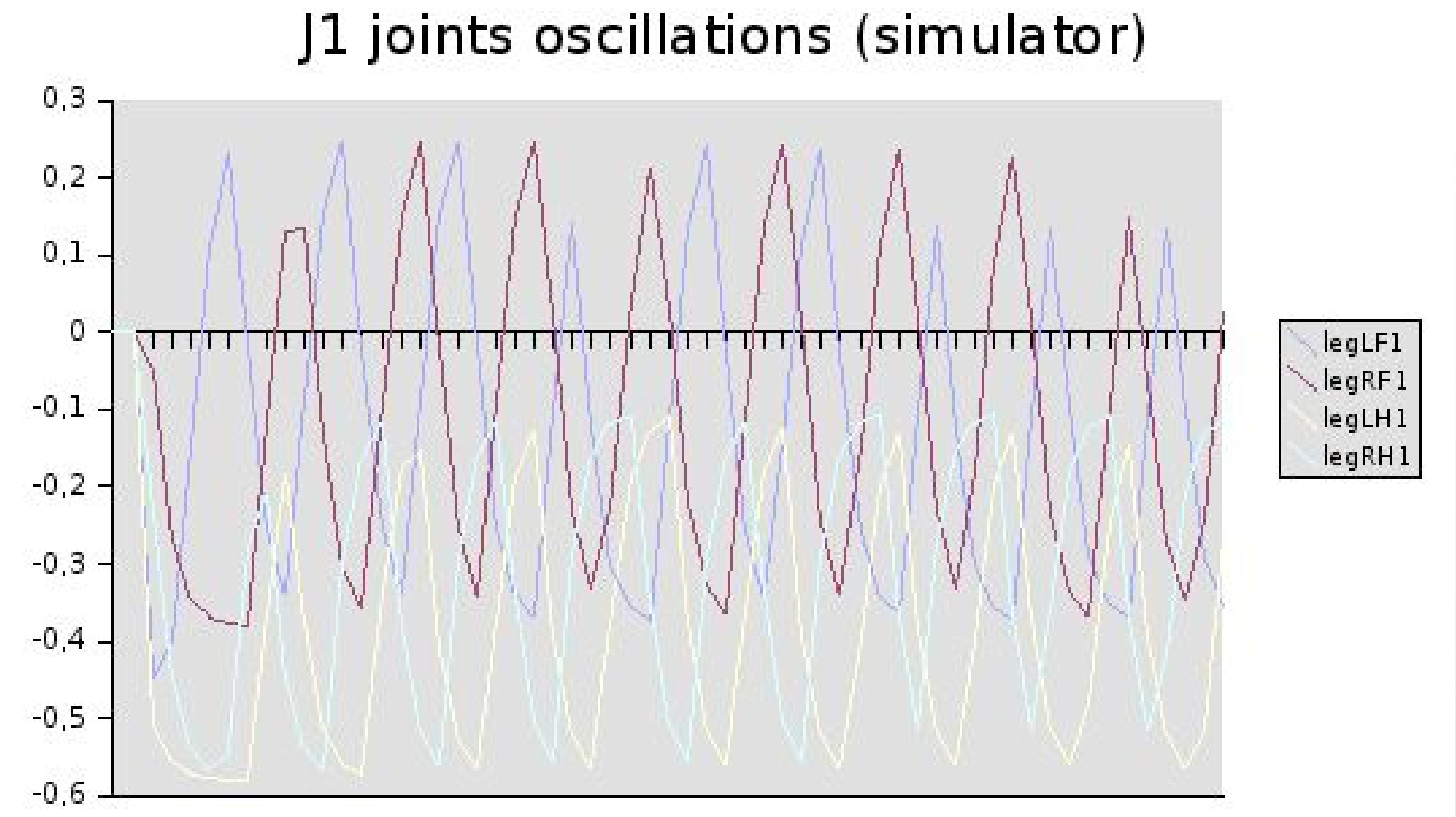
Robot control loop

- The main control loop:
  - read sensors
  - process neural nets (generate outputs)
  - send velocity commands to motors
  - wait 96 ms and repeat loop

Simulation results

- Neural net connections generated using evolutionary algorithm
- Evolutionary process made on incremental stages

Simulation results



Implementation on real Aibo using libUrbi

Synchronous approach

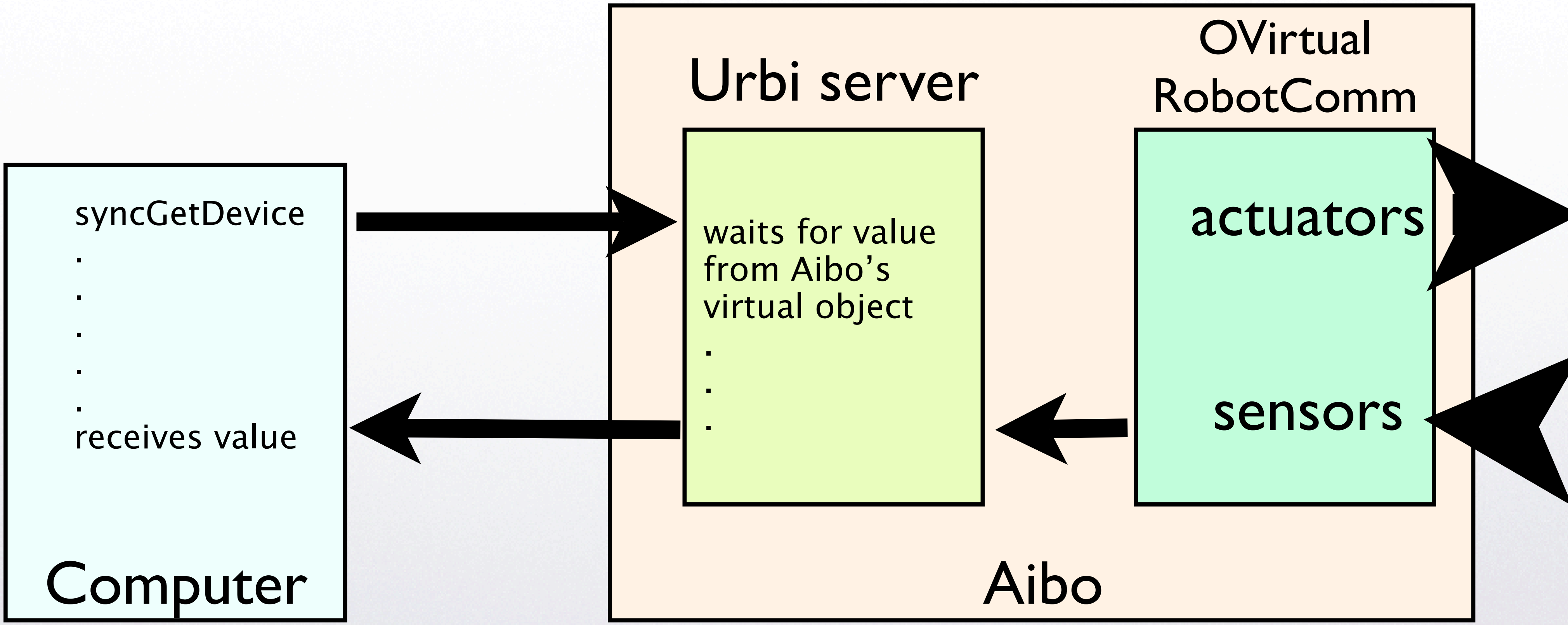
- Each time a sensor value is required, a call for the sensor value has to be performed.

```
Travolta->syncGetDevice(JOINT_MOTORS[i],sensorValue);
```

- The value returned is (in theory) the present value of the sensor
- Very easy to use and understand

Problems of this approach

- The mechanism for retrieving a value is slow and unstable (measured times of reception between 0.5 and 100 milliseconds)

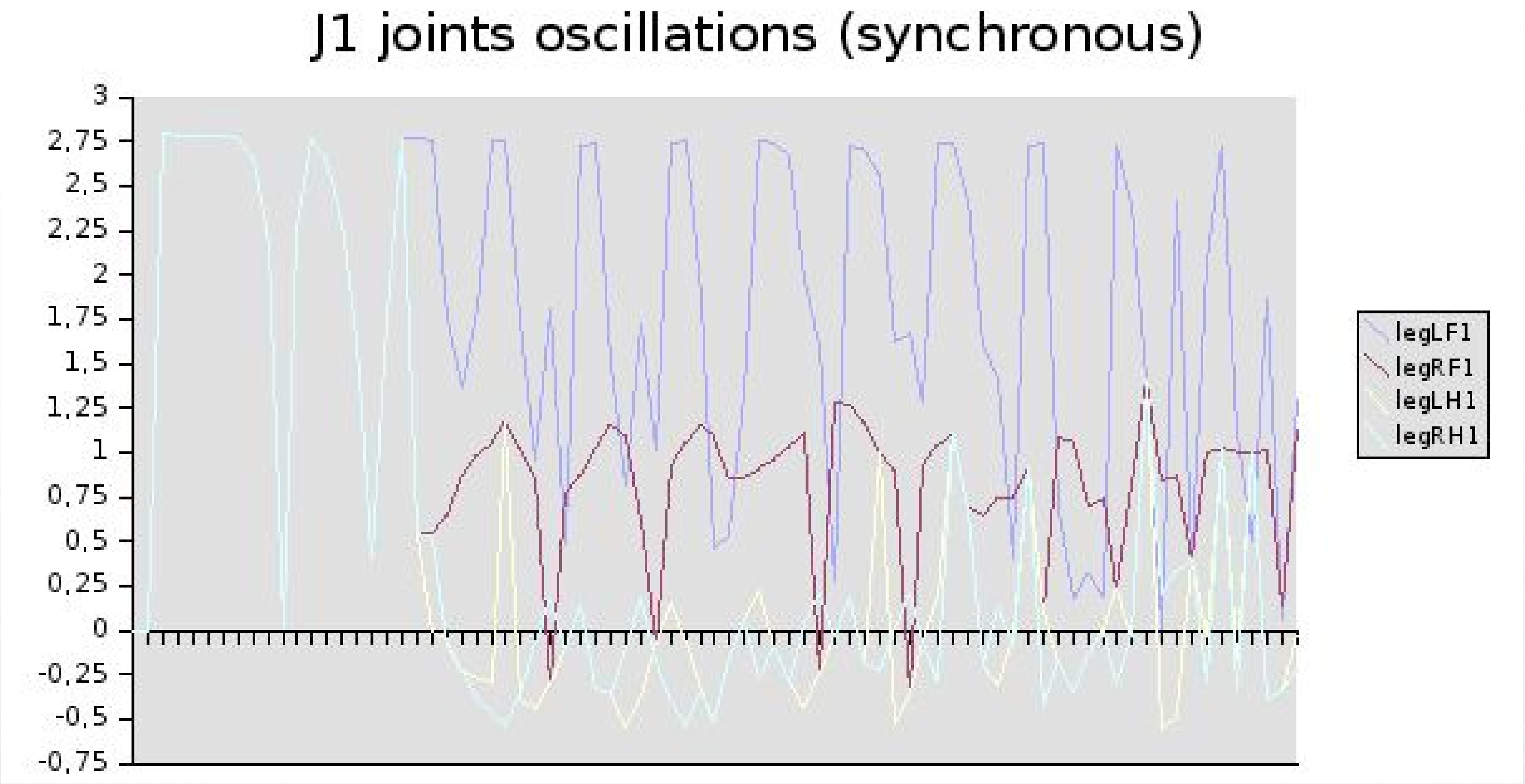


Problems of this approach

- The syncGetDevice is not optimized (yet!)
- A message has to be created for each value (in our case, 12 messages required)
- Some time required between consecutive messages for correct reception of value

```
for (int i=0; i<NUM_SENSORS; i++)
    {
        Travolta->syncGetDevice(JOINT_MOTORS[i],sensorValue);
        sensors[i] = sensorValue;
        usleep (7000);
    }
```

Synchronous results



Synchronous results

- No coordination achieved

Asynchronous approach

- Use of callback functions

```
neuronal.Travolta->setCallback(onJointValue,JOINT_MOTORS[i]);
```

- At every time that the Urbi server has a sensor value, it sends the value to the client, activating the callback
- A message received every few miliseconds (measured)

Asynchronous approach

- The callback stores locally the values received from the server

```
UCallbackAction onJointValue(const UMessage &msg)
{
    for (int i=0;i<NUM_JOINTS;i++)
     {
        if (!strcmp(msg.tag,JOINT_MOTORS[i]))
          {
              JointLastValue[i]= msg.doubleValue;
               return URBI_CONTINUE;
        }
    }
    cout << "error: no device " << msg.tag << endl;

    return URBI_CONTINUE;
}
```

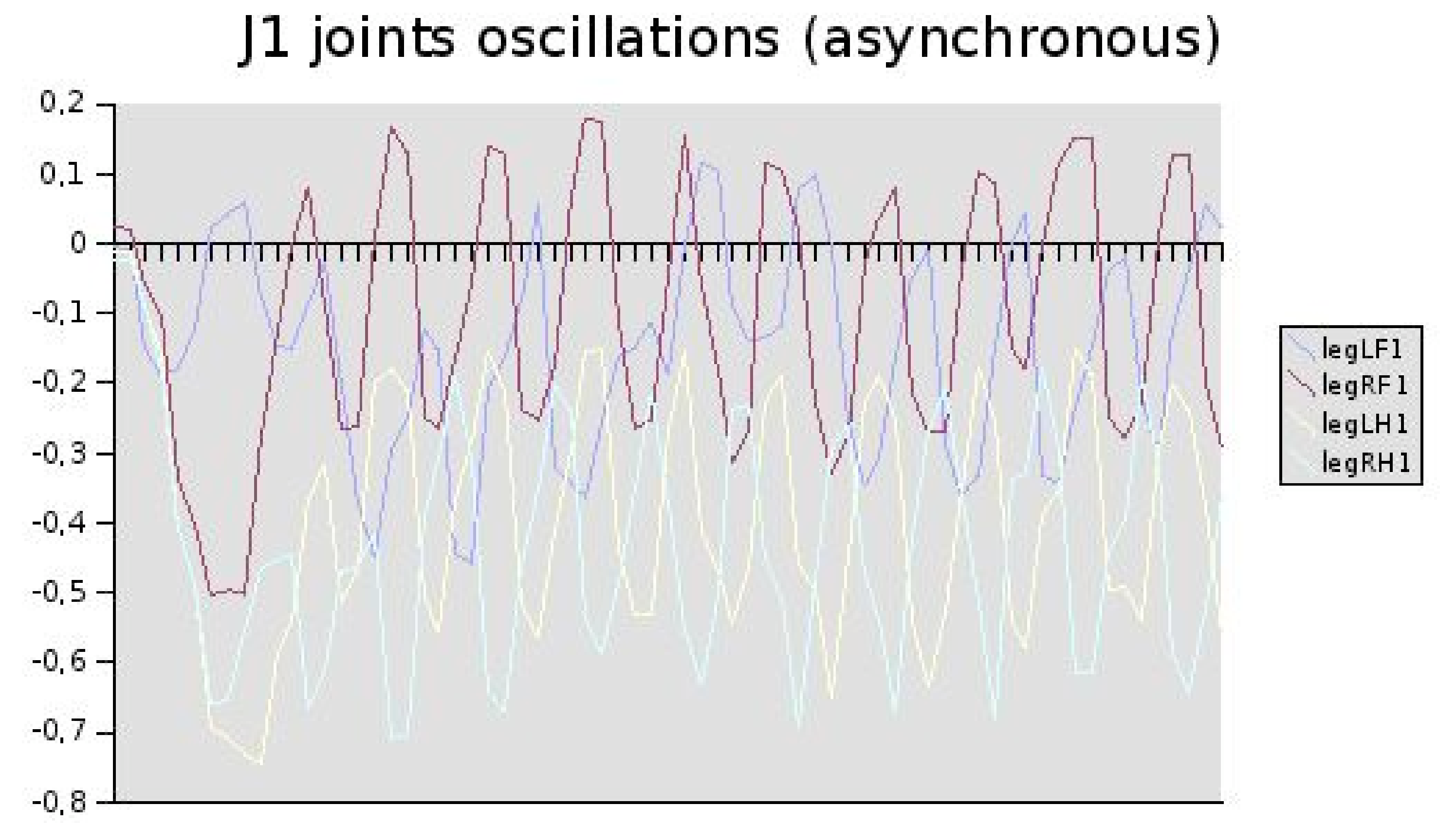
Asynchronous approach

- Each time the neural controller needs a sensor values, just takes the last value stored
- No waiting times for sensor values!
- Now the important delay is the one in sending commands from the client to the joint (but a lot smaller than the sensor delay)

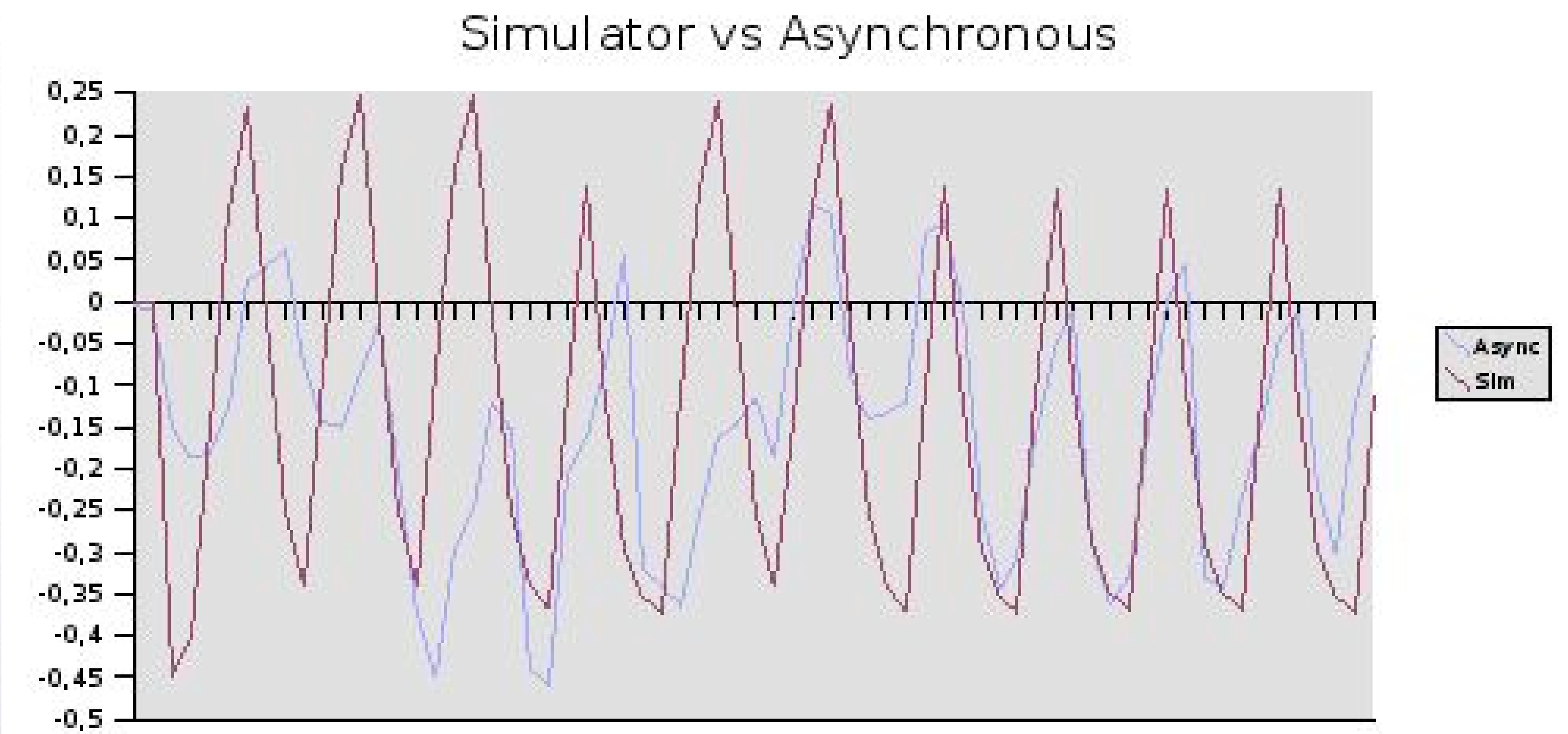
Asynchronous results

- Better coordination achieved

Asynchronous results



Asynchronous results



Future work and conclusions

Onboard implementation

- To implement the neural controller directly on the Aibo processor using libOPENR
- Better results expected, like in cross-compilation from Webots to OPENR


Conclusions

- Urbi provides two different ways of interaction with the robot sensors
- Synchronous mode is not good for highly dynamic control processes but is easier to use
- Asynchronous allows for quick sensor updates but requires the use of callbacks
- Direct implementation onboard may be even more adequate for highly reactive tasks

More information

QUESTIONS?

Urbi code of this presentation available at:

www.ouroboros.org