MIDAS Projects

Wednesday 07.10.2015 at 13.00 - 15.30

Andreas Henelius and **Jari Torniainen**

Brain●Work Research Centre
Finnish Institute of Occupational Health
http://www.ttl.fi/bwrc
https://github.com/bwrc/midas

Introduction

The goal of this hands-on part of the tutorial is to implement fully functioning systems using the MIDAS framework.

Below are presented three different projects utilising MIDAS. Projects 1 and 2 have step-by-step implementation instructions whereas Project 3 allows you to freely create any MIDAS-based system you can think of. Model solutions for the steps in Projects 1 and 2 are provided near the end of the tutorial session. All source code for the implementations is available from the MIDAS tutorial repository.

You are of course also free to just use the project descriptions as suggestions and inspiration when creating some system of your own.

We are here to help you in the implementation so please do not hesitate to ask for help!

Before starting, clone the exercise repository to your machine by typing

```
git clone https://github.com/bwrc/midas-symbiotic2015
```

Project 1: ECG Processing and Quantified Self with MIDAS

In this project you will construct a simple MIDAS network consisting of two nodes and a dispatcher. Your first task is to write two separate MIDAS clients for interacting with the nodes. In the next step the output of the two nodes are combined into a single client. The final step in this project is to replace the simulated data streams with an actual physiological data stream and then test the system in real-time.

The two nodes that make up the system are an electrocardiography (ECG) node and a quantified self (QS) node. The ECG node reads an ECG stream and can calculate various heart-rate variability metrics from the data. The QS node on the other hand monitors the current computer and can report features such as the current active application, the idle time and some network statistics. You are given the source code for both the ECG and QS nodes and they should work out of the box.

1. Create a simple ECG MIDAS client for calculating the current heart rate

All source code for Project 1 can be found in the tutorial repository under a directory called ‘project_1’.

```
repository_root/project_1
```

The code for the ECG node is functional and does not need to be modified in order to run. Please feel free to look briefly at the node code if you wish. The ECG reads a pre-recorded ECG stream that we are broadcasting.

Hint: *You can visualize 10 seconds of this stream by using the lslgraph command:*
lslgraph sim_ecg 2500
(10 seconds x 250 samples / seconds).

Currently the node is capable of calculating the average heart rate in the time window specified by the API call. The node implementation in ecg_node.py contains a wrapper function and the actual functionality can be found in the file ecg_utilities.py.

Before you can start the ECG node you must first change some configuration parameters. A configuration file template is provided in the project directory

```
repository_root/project_1/config.ini
```

Open the file in your preferred editor and locate a configuration section named [ecg]. Change the value of the variable *nodename* to something unique (your own name is a good choice for instance). In this way we prevent the confusing situation of having 15 identically named nodes in the same network. Next you can start up your node in the terminal by typing:

```
python3 ecg_node.py config.ini ecg
```

In order to communicate with the node you now need a **dispatcher**. The dispatcher has already been installed and can be started using the command

```
midas-dispatcher config.ini dispatcher
```

The dispatcher is configured to locate all the nodes on the network. You can verify that the dispatcher started correctly by visiting the URL

http://127.0.0.1:8080

You can also use the following URLs to get more information about the network.

http://127.0.0.1:8080/status/nodes
http://127.0.0.1:8080/status/metrics

You can verify the status of your ECG node from

http://127.0.0.1:8080/your_node_name/status

Don't forget to substitute 'your_node_name' with the nodename you chose earlier!

Next check the API on how to request the average heart rate.

TASK After figuring out the request create a simple MIDAS client for the ECG node that outputs the average heart rate (HR) of the past 15 seconds every 5 seconds. Print out the result on the screen.

Hint: *One of the easiest ways of performing HTTP-GET requests in Python is the* ***requests*** *module. This module has been installed and you only need to import it.*

2. Extending the ECG node

Now extend the ECG MIDAS node by adding a new function for calculating the RMSSD metric (Root Mean Squared of Successive Differences). RMSSD is a commonly used time-domain

heart rate variability metric. The function can be found in *ecg_utilities.py* and just needs to be added to the list of metric functions in the ECG node.

Modify your ECG client code to also print out the RMSSD.

3. Create a Quantified Self (QS) MIDAS client

You can find the source code for the QS node in the project_1 directory. Again you need to first change the nodename property of the QS node in the config.ini file. You can find this under the **[qs]** section. Running the node can be done in a similar fashion as in section 1.

```
python3 qs_node.py config.ini qs
```

The QS node is run on a computer and tracks the application currently in use and how long (in seconds) that the user has been inactive (not touched any inputs). All these metrics are imported from the qs_utils.py file. These metrics require no additional arguments (such as channel names or time windows) when queried.

Write a MIDAS client that request the currently active application and idle time and then prints these to the screen. You can use the API request from the ECG node as a starting point for QS client.

4. Combine the ECG and QS clients

Now combine the ECG and QS clients to create a new client that prints out the average heart rate per application, e.g., average HR of the last 30 seconds when the application was used or using some other time interval.

5. Replace the simulated ECG with a real-time ECG from a Faros180 device

The final step of the project is to replace the simulated ECG stream with a real live data stream. Begin by consulting the instructions on how to wear the Faros device. Once the device has been attached and turned on you can test if the Bluetooth device is found:

```
faros --scan
```

The above command lists the serial number and MAC address of all available Faros devices. You can check the serial number of your device from the backplate of the device. Documentation for the faros program can be accessed by typing

```
faros --help
```

If your device is on the list you can then check if it can be connected to by typing

```
faros --mac MAC_ADDRESS --blink
```

This command should cause all the four LEDs on the front face of the device to light up for few seconds. If this step was successful you can proceed. If the LEDs do not light up there might be a problem with the Bluetooth drivers or with the Faros device.

Once you have verified that you can connect to your Faros you can start streaming by typing

```
faros --mac --stream --stream_prefix YOUR_NAME
```

This command starts multiple LSL streams from your device. The stream_prefix option makes it possible for you to identify your own stream from the other ones. You can list all visible LSL streams by typing

```
lslgraph
```

If you can find your name in the list you can also visualize it with lslgraph

```
lslgraph YOUR_NAME_ecg 1000
```

Again, the numeric argument is the length of the data in samples.

Now modify config.ini file to accept this stream as an input for the ECG node. In the configuration file you'll find a section named **[ecg_rt]**. You need to change the *lsl_stream_name* and *sampling_rate* parameters to match the stream you just created. Also change the nodename to whatever you used in the previous section. Shutdown the existing ECG node and restart it using this new configuration block.

```
python3 ecg_node.py config.ini ecg_rt
```

You might also need to restart your client application. You should now see how your heart rate changes between different applications.

Project 2: EOG Processing and IoT with MIDAS

In this project you will construct two separate MIDAS clients and eventually combine them into one larger system.

You will first create a MIDAS client that communicates with an IoT platform based on the Intel Edison. You are provided with an Internet of Things (IoT) node which tracks the ambient light level intensity in the room. First step of the exercise is to request and print the current light level from the node. The second MIDAS node uses an EOG signal to detect blinks and the rate at which they occur. The final step is to produce a combined client that monitors the blink rate as a function of the current light level.

1. Create an Internet of Things (IoT) MIDAS client

All source code for Project 2 can be found in the tutorial repository under a directory called ‘project_2’.

```
repository_root/project_2
```

The IoT node runs on an Intel Edison embedded system that has some sensors connected to it and tracks the ambient light level in the room.

Hint: *You can visualize 5 seconds of this stream by using the lslgraph command:*

lslgraph iot 50
(5 seconds x 10 samples / seconds).

Before you can start the IoT node you must first change some configuration parameters. A configuration file template is provided in the project directory

```
repository_root/project_2/config.ini
```

Open the file in your preferred editor and locate a configuration section named [iot]. Change the value of the variable *nodename* to something unique (your own name is a good choice for instance). In this way we prevent the confusing situation of having 15 identically named nodes in the same network. Next you can start up your node by moving to the project 2 directory and typing (in the terminal)

```
python3 iot_node.py config.ini iot
```

In order to communicate with the node you now need a **dispatcher**. The dispatcher has already been installed and can be started using the command

```
midas-dispatcher config.ini dispatcher
```

The dispatcher is configured to locate all the nodes on the network. You can verify that the dispatcher started correctly by visiting the URL

http://127.0.0.1:8080

You can also use the following URLs to get more information about the network.

http://127.0.0.1:8080/status/nodes
http://127.0.0.1:8080/status/metrics

You can verify the status of your IoT node from

http://127.0.0.1:8080/your_node_name/status

Don't forget to substitute 'your_node_name' with the nodename you chose earlier!

Next check the API on how to request the average level of ambient light.

TASK After figuring out the request create a simple MIDAS client for the IoT node that outputs the average level of ambient light of the past 10 seconds every 5 seconds. Print out the result on the screen.

Hint: *One of the easiest ways of performing HTTP-GET requests in Python is the* ***requests*** *module. This module has been installed and you only need to import it.*

2. Counting the blinks in real-time

Next step is to create an EOG node for detecting eye blinks. The source code for the MIDAS EOG node ("blink node") is also found in the directory

```
repository_root/project_2
```

The code for the blink node does not require modification. Please feel free to look briefly at the node code if you wish. The blink node reads a pre-recorded EOG stream that we are broadcasting. The node includes functionality to detect the blinks from the EOG signal using a

threshold-based algorithm. The node has a single metric function that detects and calculates the number of blinks in the specified time-window.

You can run the blink node as you did the IoT node in the previous section. Again, remember to change the *nodename* parameter in **[blink]** section of the configuration file.

Next check the API on how to request the number of blinks.

TASK After figuring out the request create a simple MIDAS client for the blink node that request the blink count of the past 15 seconds.

Print out the result on the screen.

*Hint: One of the easiest ways of performing HTTP-GET requests in Python is the **requests** module.*

3. Extending the EOG node

The raw number of blinks is not a very useful metric. The *blink rate* on the other hand is a much more descriptive metric and can be used to quantify, for instance, mental fatigue. Extend the blink node by adding a new function for calculating the blink rate. You can use the existing metric function and just divide the blink count by the length of the time window.

Also modify the blink client to print out the *blink rate*.

4. Combining the EOG and IoT clients

Now combine the EOG and IoT clients to create a new client that prints out the average blink rate and current ambient light level each five seconds.

5. Replace the simulated EOG with a real EOG signal from a Faros180 device

The final step of the project is to replace the simulated EOG stream with a real live data stream. Attach the red electrode above your eye and the yellow electrode below the eye. Once the device has been attached and turned on you can test if the Bluetooth device is found:

```
faros --scan
```

The above command lists the serial number and MAC address of all available Faros devices. You can check the serial number of your device from the backplate of the device. Documentation for the faros program can be accessed by typing

```
faros --help
```

If your device is on the list you can then check if it can be connected to by typing

```
faros --mac MAC_ADDRESS --blink
```

This command should cause all the four LEDs on the front face of the device to light up for few seconds. If this step was successful you can proceed. If the LEDs do not light up there might be a problem with the Bluetooth drivers or with the Faros device.

Once you have verified that you can connect to your Faros you can start streaming by typing

```
faros --mac --stream --stream_prefix YOUR_NAME
```

This command starts multiple LSL streams from your device. The stream_prefix option makes it possible for you to identify your own stream from the other ones. You can list all visible LSL streams by typing

```
lslgraph
```

If you can find your name in the list you can also visualize it with lslgraph

```
lslgraph YOUR_NAME_ecg 1000
```

The suffix of the signal is *ecg* although this is an EOG signal. The numeric argument is the length of the data in samples.

Now modify config.ini file to accept this stream as an input for the blink node. In the configuration file you'll find a section named **[blink_rt]**. You need to change the *lsl_stream_name* and *sampling_rate* parameters to match the stream you just created. Also

change the nodename to whatever you used in the previous section. Shut down the existing blink node and restart the node using this new configuration block:

```
python3 blink_node.py config.ini blink_rt
```

You might also need to restart your client application.

Project 3: Free Implementation with MIDAS

Choose this project if you have an idea of your own that you want to try. Feel free to mix and combine any ideas from the above described project.

You can use either the streaming pre-recorded ECG signal or then you can record an ECG signal or three-axis accelerometer signal using the Bluetooth-enabled Faros 180 devices provided.

It is a good idea to sketch a diagram of the architecture of the MIDAS system needed for your project before you start implementing.

Hint: *You can list all available lsl-streams from the command line by using the* ***lslscan*** *command.*