M2O2P

Multi Modal Online & Offline Programming Solutions

User Manual

Samu Rautiainen

4.4.2022

Contents

INTRODUCTION

The purpose of this manual is to make the installing and using of **M2O2P** easier, and more achievable, even with less information of related systems. Following steps will provide information about installation, volumes that are used for configuration of images, the Web UI, and ways to test the system.

Requirements:

- Windows PC
- Downloaded all the needed M2O2P docker images
- CaptoGlove
- Installed Capto Suite (optional, but recommended for the glove)

For testing section (optional):

- Installed Postman

Following of this manual will be easier if the CaptoGlove is already set up for the system using CaptoGlove manual. There will be short explanation how to make the CaptoGlove work, but information about the glove itself will not be provided.

DOWNLOADS AND INSTALLATION

Before following the later instructions, clone this GitHub repository (same repository where the manual is located). All the needed application files except the Docker images can be achieved from here.

```
git clone https://github.com/TAU-FASTLab/m2o2p.git
```

First the CaptoGlove firmware needs to be updated. For more information about this, follow the CaptoGlove manual. Newest firmware can be achieved from CaptoGlove website, and it is installed via Capto Suite when the glove is connected to Windows PC with USB-cable.

After the installation of firmware, the CaptoGlove needs to be paired with the with the Windows PC via Bluetooth. When the Bluetooth connection is established, check what is the name of the CaptoGlove from paired Bluetooth devices. The name can be for example *CaptoGlove3170*. This needs to be written to CaptoGlove SDK configuration file (configuration.txt) which located is in SDK folder. Be sure to write the glove number to right row, depending on if it is left or right glove. Now to test this, go to “captoglovesdk” folder and execute CaptoGloveSDK.exe. If everything is correct, after the 10 seconds delay, SDK command window should show “can’t connect to server” at last. If it is, everything is working till this point.

It is also recommended to use Capto Suite to calibrate every finger if there is new glove and/or the firmware was just updated. When CaptoGlove is connected to the Capto Suite (the state of connection is green), under Setup section click the Fingers button, which directs you to Calibration section for each finger. Using this, the calibration provided within the M2O2P application needs to be used only for finetuning.

CONFIGURATION

For next steps you need to pull the docker images from the RAMP docker registry. Docker images can be launched at the same time with the provided docker-compose.yml -file.

Docker-compose file

Before using the docker compose, the docker-compose.yml needs to be modified. To configure the docker images, local volume files are used. Volumes are written to docker-compose file under volume tag. When using Windows, the path can be for example:

- /c/Users/User/Documents/ros2_ros1_command.yaml:/home/is-workspace/src/bridge/src/ros2_ros1_command.yaml

The path of the volume of the host machine (the path before colon) needs to be modified to point to right folder, to the folder where the before mentioned GitHub repository was cloned.

The M2O2P offers natively ROS2 (Foxy) interface on top of normal behavior using FIWARE interface, and with integration-service same information is available in ROS1 (Noetic) topics too. If such interface is not needed, the integration-service service can be deleted from docker-compose file.

For initial testing it is easier to leave mongo-db, orion and postgres images to the docker-compose file, but in the case any of these are deployed somewhere else, corresponding changes needs to be done in configuration.json. The configuration file is explained in the next section.

Volumes

Volumes are files that are mounted to the docker image and provide the ability to configure the docker images without rebuilding them.

configuration.json

It has needed configuration for the system to work. It has following sections:

config_ac, which is the Application Controller configuration

- *gloves_connected*: the number of gloves that are connected (1-2). System works now well with 1 glove
- *gesture_list*: the list of gestures that can be done. Each gesture holds 3 fields, bending sensor values, pressure sensor values and gesture name. For the feasibility, pressure sensors are not used
- *command_map*: maps command indices to gestures. The gesture needs to have corresponding gesture name in the *gesture_list*
- *postgres*: holds information about the postgres database which holds the Task definitions, it should match the used postgres. self_deploy variable should be in test case 1, but when the postgres database is populated by some other application, then it should be 0
- *query*: here you can define what rows should be retrieved and from which table. The WHERE should match column where the task code is queried. These variables are meant to be configured so the column names of Task definition table can be determined by user. Example of self-deployed table in ***1*.**

config_bridge, which is the configuration for ros2-fiware-bridge

- *self_deploy_device*: when 1, device entity is deployed by the bridge. This is the normal behavior
- *orion_url*: if docker compose is used to deploy the Orion Context Broker, use here <container_name>:1026. If OCB container is deployed somewhere else (on different host computer), use the IP of the host machine of this other PC
- *device_entity*: the Device entity that is self-deployed. Here the state of the device and the command id will be updated by Application Controller
- *update_status*: manifest for updating the Task entity status
- *update_device_state*: manifest for updating the Device entity state
- *update_command_id*: manifest for updating the Device entity command id
- *subscription_entities*: all the needed subscriptions for the system to work. First one will subscribe the Task entities for new Tasks that has the Device id as involvement. Now the system only supports systems where there is only one involvement per Task. The second one subscribes status change if some other application updates the status of the Task that was given to M2O2P

1

Table 1. Example how the start of TaskDef table looks like

task_id	task_process_id	task_name	task_code	command_id
1	2	Reach the tray COMMAND	RTTC	1
2	2	Freedrive COMMAND	FC	2
3	2	Grasp component COMMAND	GCC	3

EXAMPLE CASE OF CONFIGURATION:

Orion Context Broker (OCB) is running on a different host machine, let's say with IP 192.168.7.21. The M2O2P communicates with FIWARE through the ROS2-FIWARE bridge. Therefore, all the needed changes should happen there. This IP needs to be changed in every place where there is "orion" as IP, meaning to the *orion_url* parameter. Additionally, the OCB is not familiar with ip "ros2-fiware-bridge", so the *subscription_entities* needs be changed so that the IP matches the host machine where the docker-compose is launched.

More examples are added if/when they are discovered through testing.

ros2_ros1_command.yaml

This is needed only for the use of integration-service. The yaml-file is used for defining the connected systems, routes for them and topics. There are no changes needed here, since from Application Controller the ROS2 (Foxy) topic is "/command_id" as string and so it will be on ROS1 (Noetic) side.

original_limits.txt

This file contains limits for each bending sensor (upper and lower limits → 3 states per finger) and for pressure sensors. For now, only bending sensors are used. These limits can be modified via Web UI of AC, so this file can be left untouched. In free version of the glove application the limits cannot be modified from Web UI. In this case either:

- Get the full version

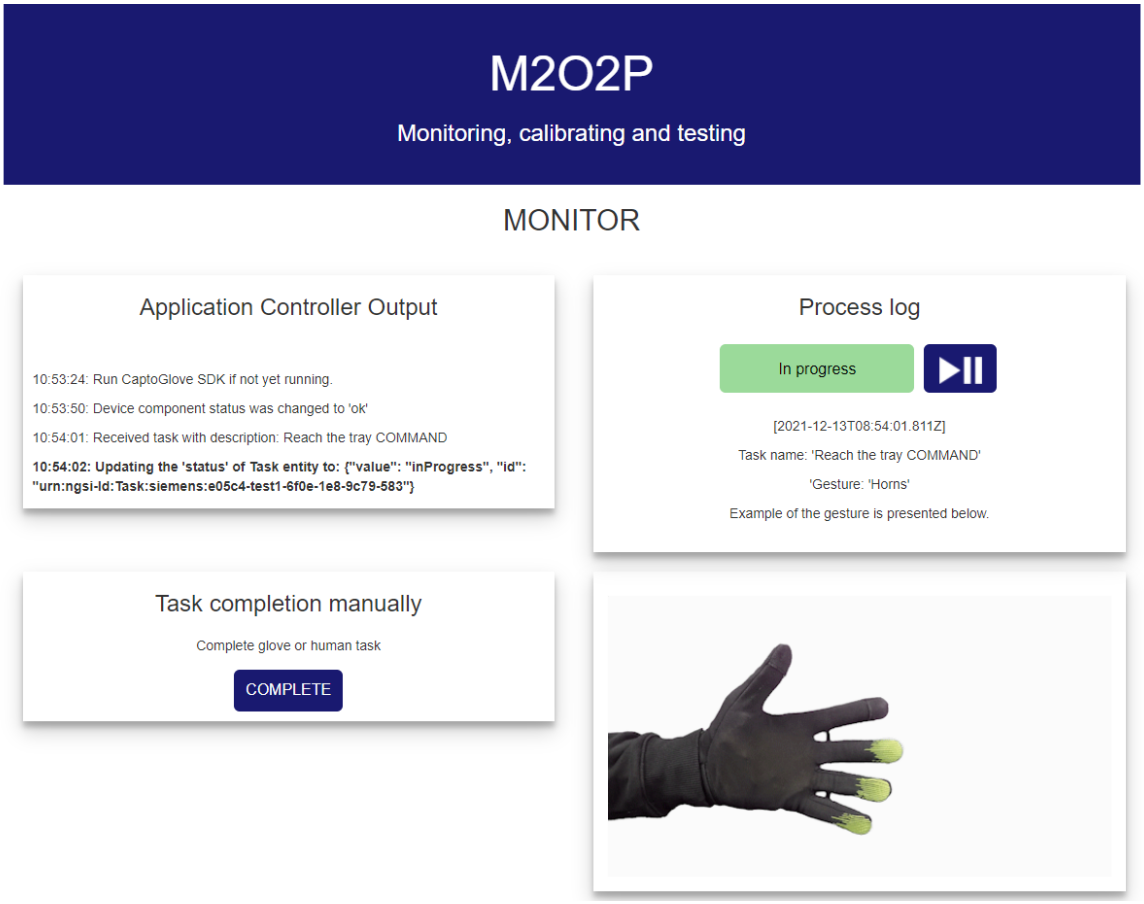
- Straighten all the fingers and check the SDK what the values are (snipping tool etc.) and bent all fingers and do the same. Then put the higher threshold lower than the value of each finger when straightened, and lower threshold similarly to higher value than each finger when bent. This way there will be three states, bent, something in between and straightened.

USER INTERFACE

When the containers are running, Web UI can be reached from the host computer by navigating *localhost:54400* or in providing the IP of the host when using another computer. Short explanation of the UI:

First, we have **MONITOR** section. This is the only section that is provided within the free version of the component. Here we have 4 blocks;

- *Application Controller Output*: needed outputs from the AC is provided here, acts as outlet for AC and provides useful information to user on what is going on at the background
- *Process log*: In the picture, green square will have different colors depending on the state of the task, or if there is one. Play-pause button provides functionality for the user to pause the task. This will update the Task entity in OCB to "paused" state. Under these there are Task name, the gesture name and GIF of the gesture.
- *Task completion manually*: The task can be completed manually with this button.

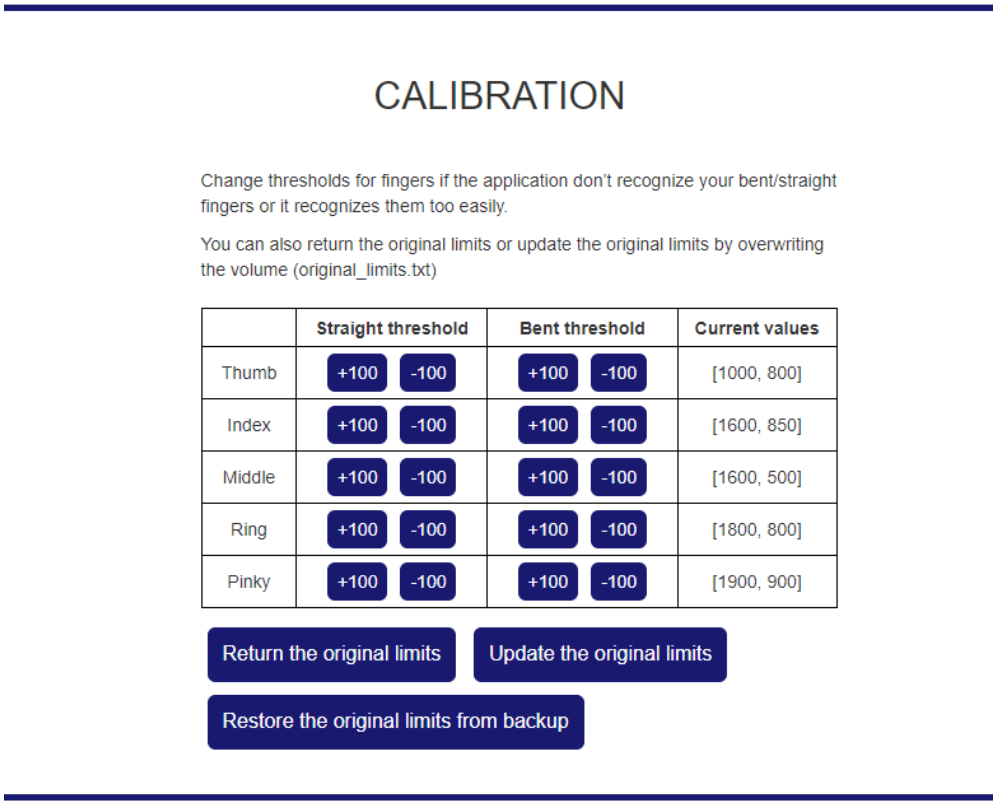


Next, we have **CALIBRATION** section

Here the user can change the limits of the thresholds. For each finger, Straight threshold and Bent threshold can be changed. If for example index Straight threshold +100 is clicked, the upper threshold will get +100, and for the system this change will happen immediately.

Under the table, there are three buttons, *return the original limits* which will return the limits that were used before the user clicked the +100 for index finger, *update the original limits* which will update the limits to the original_limits.txt file and now the *return of the original limits* does nothing,

and *restore the original limits from backup*, which will retrieve the limits that were given to the system when it was launched.



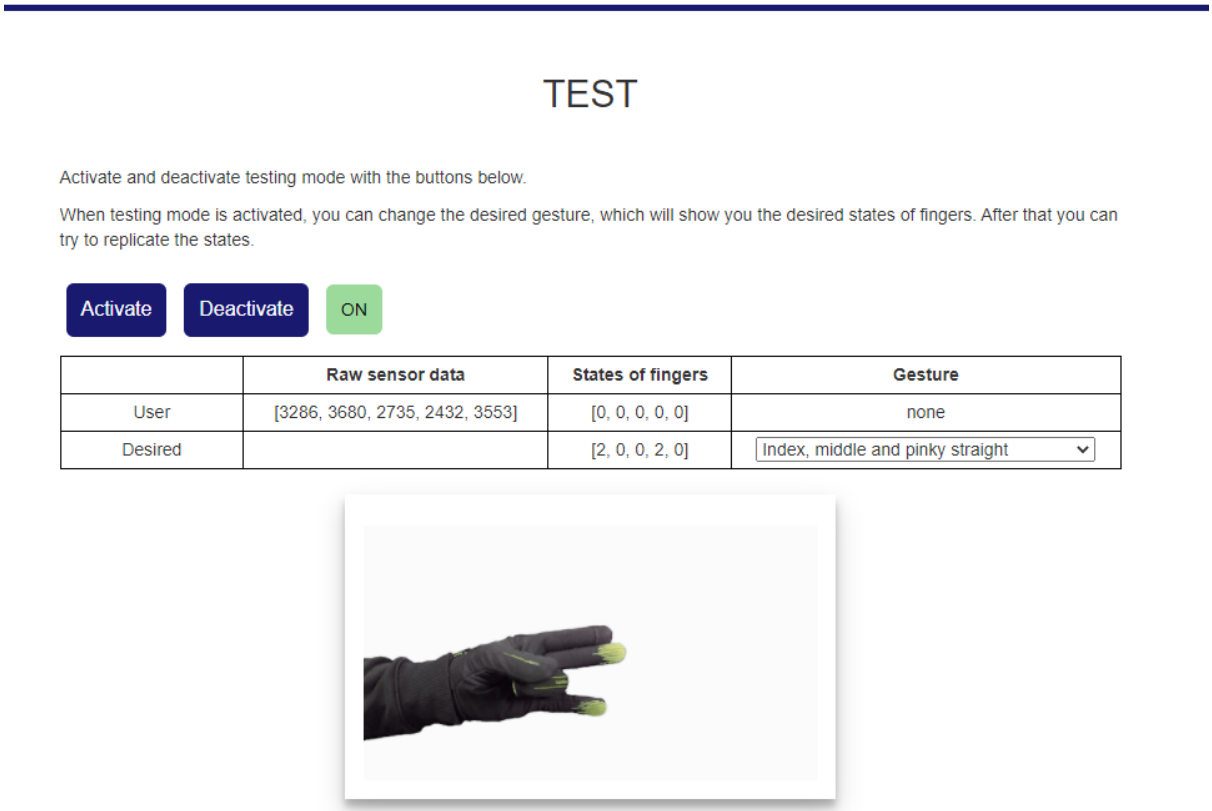
Next we have **TEST** section.

Here we can test the glove. Activating the testing mode with *Activate* button, we get values to the table below. When in testing mode, we are not able to send commands or complete tasks.

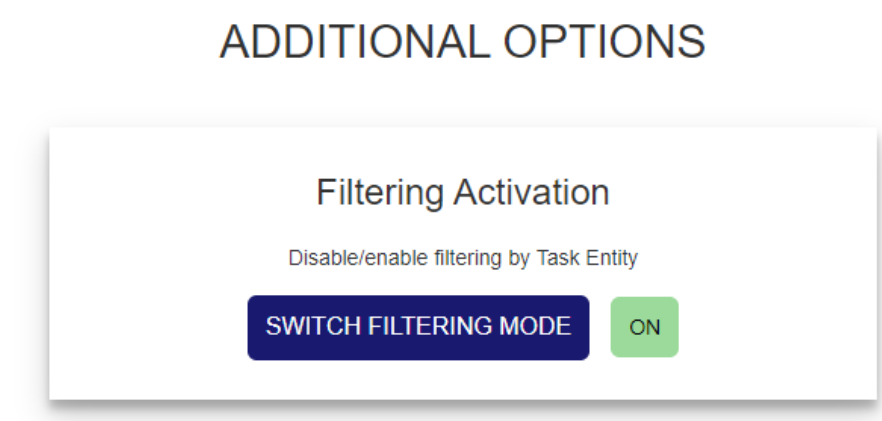
First in the table we have the raw sensor data, then we have states of fingers, and then the gesture. Desired states of fingers are the states that the user needs to achieve for the chosen gesture to be done. In gesture column the current made gesture is shown if there is one. In desired gesture cell the user can choose what is the gesture to be tested. This section can be cross referenced with calibration section to get the glove calibrated for the user.

Below this table there is the GIF of the gesture shown. The GIFs can be seen even if the testing mode is not activated. This offers a way to user to check all the available gestures.

The idea here would be to go through the list of gestures and do them, as a training procedure. All the gestures that are used in configuration command map should be achievable.



Lastly there is **ADDITIONAL OPTIONS** section, where filtering mode can be set OFF and ON. If Filtering mode is set OFF, when the user makes a gesture and holds it 1,5s at any given moment, command will be sent to Device entity. This mode can be used if one wants to test the commands that are sent from the glove without updating the Task. For normal behavior, it is recommended to have the filtering mode on. It cuts out many unwanted gestures. In practice, filtering mode provides functionality, where when M2O2P has a task, there is only one gesture that can be done to make the process go forward. When there is no task, no gesture will have an effect.



TESTING

Now that all the systems are installed, we can try out the M2O2P. First go to the directory where the docker-compose.yml is, use *docker compose up* -command. After the docker containers are running, execute the CaptoGloveSDK.exe. If everything is going well, now the command window for the docker compose up will have line “<left or right> glove connected” and the CaptoGloveSDK.exe window will have prints of sent data. Now one can navigate to *localhost:54400* on browser to see the UI. With Postman, GET all device entities shows the newly made Device entity.

Use the TEST section of UI to figure out if the limits are okay, and the gestures are achievable if needed. Now for the testing of the Tasks, use the Postman template provided in the GitHub and create Task. The id has to be every time new, so update it if multiple tests are done in a row. The Task entity will have “pending” status when created. The template has “RTTC” as task code so after POSTing the Web UI should show Task “Reach the tray command” in “inProgress” state. If it is, M2O2P successfully got the task. This can be verified with Postman GET method for Task entities. After this, the user can do the “Horns” gesture as indicated in the Web UI. When the “Horns” is held 1,5 seconds, the user will see from Application Controller output that the Task entity “status” is updated to “completed”. This also can be seen from Postman when using GET to retrieve all Task entities. The Device entity command id is also updated with the same command id that is mapped to “Horns” gesture in the command_map in configuration file.