MyExoMy project 2021



# Introduction

# Topics

## Way of working

* Connect to the RPi using SSH
* Use sh ~/ExoMy\_Software/docker/run\_exomy.sh -d to start the ExoMy docker container for development. This will map the docker container folder /root/exomy\_ws/src/exomy which contains the ExoMy scripts to /home/pi/ExoMy\_Software. This is done by the run\_exomy.sh script by using the ‘-v’ (volume) option with the docker run command.
* On the host computer open FileZilla and connect to the RPi using port 22 for SFTP (SSH FTP). This way files can be transferred between the host computer and /home/pi/ExoMy\_Software. Because this folder is mapped to the ExoMy docker folder /root/exomy\_ws/src/exomy any change will have immediate effect on the running container. This way the changes will be permanent and not lost when the container is stopped.
* When changing something in the GUI Web Interface, like the index.html or style.css, it can be necessary to delete the history of the browser at the host computer before the changes can be seen!
* After logging in to the RPi one can run the docker exomy\_devel container for development with:  
  sh ~/ExoMy\_Software/docker/run\_exomy.sh -d : starts the exomy\_devel container  
  And then in sequence:  
  source /opt/ros/melodic/setup.bash : does some catkin setup   
  cd /root/exomy\_ws  
  catkin\_make : builds all ROS packages  
  http-server src/exomy/gui -p 8000 & : starts the web server  
  source devel/setup.bash : does some catkin setup  
  roslaunch exomy exomy.launch : starts the ROS nodes as specified in exomy.launch
* When a python script is changed it is required to quit ROS and to issue roslaunch again.
* In docker/run\_exomy.sh one can see that all docker containers are based on the same image named ‘exomy’.

## About Docker

Docker is used on the ExoMy to provide a reliable and reproducable running environment. Docker uses a stable read-only image (ros:melodic) and additional installation commands specified in docker/Dockerfile are used to build a container where the software runs. Compare this with using an image and manually adding installations to it and letting it grow and grow. This is very hard to maintain or to roll back or repair if somethings goes wrong.

## Changes to ExoMy source code

* The gui/index.html and gui/style.css were adapted to have a bigger video image and to display the published battery voltage and solarpanel voltage.
* For I2C the I2C.py script was added. This was needed to communicate with the ATmega328P processor. This script needed smbus to be installed. Therefore in docker/Dockerfile the installation command apt-get install python-smbus was added.
* In src/robot\_node.py additional code was added to publish the battery voltage and solarpanel voltage to the ‘/battery\_voltage’ and '/solarpanel\_voltage' topic respectively every second.
* In src/robot\_node.py additional code is added to handle the messages published by the ATmega328P on the ‘/own\_button’ topic. These messages are:
  + lights\_on. Through I2C the ATmega328P is instructed to switch on the lights.
  + lights\_off. Through I2C the ATmega328P is instructed to switch on the lights.
  + goto\_sleep. Through I2C the ATmega328P is instructed to go to sleep. This will be acknowledged by the Arduino. Only after the acknowledge the Raspberry Pi will be shut down. Because this must happen from within a docker container, an empty file is used to communicate to the host that it must issue a shutdoen command. The sequence of events:
    - Sleep button is pressed on the web page.
    - In index.html the message ‘goto\_sleep” is published on topic ‘/sleep\_status’.
    - In robot\_node.py the message callback function sends through I2C the instruction to the ATmega328P to go to sleep.
    - The ATmega328P acknowledges the goto\_sleep and goes to sleep after a delay. This delay is needed to enable the Raspberry Pi to shut down in a proper way.
    - robot\_node.py checks for the acknowledge and then writes an empty file 'exomy\_shutdown.txt' in the '/root/exomy\_ws/src/exomy/' folder of the container. This folder is mapped to the host folder '/home/pi/ExoMy\_Software/'.
    - On the host the ‘/home/pi/ExoMy\_Software/check\_exomy\_shutdown.py’ script checks regularly for the presence of the exomy\_shutdown.txt file. If it is present, it deletes the file and uses the shutdown command to shutdown the Raspberry Pi.
* The ‘check\_exomy\_shutdown.py’ script is added on the Raspberry Pi host to ‘/etc/profile’.
* In src/motors.py the drive motors are switched off when the speed is very low. This to eleminate the need to calibrate the drive motors to zero speed using the potentiometer every time.

## Low Power ATmega328

For the MyExomy project the ATmega328P-PU is used with an alternative bootloader which sets the clock to the 8 MHz internal clock. The new board definition (named 'ATmega328 on a breadboard (8 MHz internal clock)') and bootloader is available in the breadboard-1-6-x folder.

See <https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoISP> for how to burn a bootloader on one Arduino board using another Arduino board as ISP.

The 'ATmega328 on a breadboard (8 MHz internal clock)' board is made available to the Arduino IDE by copying the breadboard folder to C:\Program Files (x86)\Arduino\hardware.

See <https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoToBreadboard> for a description of how to connect an ATmega328P on a breadboard and use it.

To make the new board available in Visual Studio Code an Atmega328\_on\_breadboard\_8MHz.json file is added to C:\Users\reneb\.platformio\platforms\atmelavr\boards. This json file is created by copying the uno.json file (which is for the Arduino Uno) and adapted the content with info from boards.txt in breadboard-1-6-x.

## Start and view RTSP stream

* Start RTSP server:  
  raspivid -o - -t 0 -hf -w 1920 -h 1080 -fps 30 | cvlc -vvv stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/x}' :demux=h264
* View in VLC:  
  GUI -> Media -> Open Network Stream -> rtsp://192.168.1.170:8554/x

Note: it is needed to give the stream a name, here ‘x’.

## Power Board

The power board has the following features:

* Provides 5V for the RPi and 6V for the servos and the headlights.
* Can turn the MyExoMy into deep sleep mode, consuming only 100 nA.
* Can switch on the MyExomy using an external trigger.
* Has a light sensor to turn on the lights when it is dark.
* Has a charge connection for the batteries.
* Has a trickle charge circuit with voltage protection for the solar panel.

### EasyEda

* To work with EasyEda for the MyExoMy projet first change the Data Directory to the corresponding EasyEda folder with EasyEda -> Setting -> Desktop Edition Setting -> Data Directory. After that with EasyEda -> File -> Open Project the MyExoMy EasyEda project can be opened.
* Switching to another project can be done by setting the Data Directory to a different folder.
* If desired the other projects can be removed from the ‘Opened Projects’ list after EasyEda -> Login and then using the right mouse button -> Refresh List.

### EasyEda PCB settings:

* The track width is chosen to be 0.8 mm with a clearance of 0.4 mm.
* The copper thickness is chosen to be 2 oz = 2x 1.4 mil = 2x 35 μm = 70 μm. Normally it is 1 oz. 2 oz is chosen to allow more current and for robustness. It cannot be set in EasyEda but it can be selected when ordering at JLCPCB.
* According to <https://www.7pcb.com/trace-width-calculator.php> with a track width of 0.8 mm and a thickness of 2 oz the current can be appr. 3A.
* Only the battery tracks have a width of 1.27 mm and with 2 oz thickness can carry appr. 5A.

## Low Power

The Raspberry Pi can be put in low power by issuing a ‘sudo halt’. The power of the Raspberrt Pi 4 model B board will go from appr. 700 mA to appr. 16 mA in low power. For this to work the EEPROM bootloader configuration has to be adapted.

### Default EEPROM bootloader configuration settings, low power settings in red

See also <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711_bootloader_config.md>

BOOT\_UART=0

WAKE\_ON\_GPIO=1 -> 0

POWER\_OFF\_ON\_HALT=0 -> 1

DHCP\_TIMEOUT=45000

DHCP\_REQ\_TIMEOUT=4000

TFTP\_FILE\_TIMEOUT=30000

ENABLE\_SELF\_UPDATE=1

DISABLE\_HDMI=0 (can be set to 1 if HDMI is not used, makes no difference for low power though)

BOOT\_ORDER=0xf41

To view: rpi-eeprom-config  
To edit: sudo -E rpi-eeprom-config –edit

After editing, whether you changed anything or not, always issue a sudo reboot, otherwise the setting does not seem to have any effect.

# Measurements

## Power consumption, measered with battery voltage = 6.0V

* Fully operational, standing still: appr. 0.90 A
* Fully operational, standing still + lights on: appr. 1.35 A
* Fully operational, driving and steering: appr. 1.5 .. 2.0 A
* Fully operational, driving and steering, lights on: appr. 2.0 .. 2.5 A

# Useful links

<https://github.com/esa-prl/ExoMy/wiki>

<https://github.com/esa-prl/ExoMy_Software>

<https://msadowski.github.io/ros-web-tutorial-pt1/>

<https://www.clearpathrobotics.com/assets/guides/kinetic/ros/Practical%20Example.html>

<https://learn.adafruit.com/16-channel-pwm-servo-driver>