**MyExoMy project 2021**



Rene Bakx, May 2021

ontents

[1. Introduction 3](#_Toc71186148)

[2. Way of working 3](#_Toc71186149)

[2.1. For development 3](#_Toc71186150)

[2.2. For normal operation 3](#_Toc71186151)

[2.3. About Docker 3](#_Toc71186152)

[3. From Exomy to MyExomy 4](#_Toc71186153)

[3.1. Additions / changes in hardware 4](#_Toc71186154)

[3.2. Additions / changes in software 4](#_Toc71186155)

[4. Topics 5](#_Toc71186156)

[4.1. Low Power ATmega328 5](#_Toc71186157)

[4.2. Start and view RTSP stream 5](#_Toc71186158)

[4.3. Power Board 5](#_Toc71186159)

[4.4. EasyEda 6](#_Toc71186160)

[4.5. EasyEda PCB settings: 6](#_Toc71186161)

[4.6. Low Power 6](#_Toc71186162)

[4.6.1. Default EEPROM bootloader configuration settings, low power settings in red. 6](#_Toc71186163)

[5. Measurements 7](#_Toc71186164)

[5.1. Power consumption, measured with battery voltage = 6.0V 7](#_Toc71186165)

[6. Useful links 7](#_Toc71186166)

# Introduction

All credits for this document go to Maximilian Ehrhardt and Miro Voellmy who created a 3D printable robot ExoMy, a small version of the ExoMars Europe’s Rosalind Franklin ExoMars rover which is scheduled to be launched to Mars in 2022.  
They made a very nice open source hardware and software design. This document describes how I created my own version of the ExoMy, the MyExoMy.

This document can be seen as a notebook with practical information mainly for personal use.

# Way of working

## For development

* 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’.

## For normal operation

* Use a single sh ~/ExoMy\_Software/docker/run\_exomy.sh -a and ExoMy will become operational, also after a reboot.
* The ExoMy can be operated through its web page at port 8000.

## About Docker

Docker is used on the ExoMy to provide a reliable and reproducable runtime 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 writable 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.

# From Exomy to MyExoMy

The standard open source hard and software design is a very good basis and suitable for all kinds of additions. Below the additions and changes are listed.

## Additions / changes in hardware

* A 20W solar panel is added to the design to charge the batteries. This way the ExoMy can be 24/7 operational without the need of a separate charging station. Not that ‘operational’ means that for most of the time the ExoMy will be sleeping and / or charging. Even in winter with dark and short days one should be able to drive a few minutes every day.  
  In summertime this is about 20 minutes every day. This could be higher but the charging current is limited to 60 mA to keep things simple and to be sure the batteries are not overcharged, although charging will stop when the batteries reach a threshold voltage.
* To accommodate the charging with the solar panel a Power Board is designed. This board provides safe charging for the batteries and power to all components like the Raspberry Pi, servo board and the lights. It can also put the ExoMy in deep sleep.
* Instead of the LiPo accupack 11.1 V 3000 mAh, 5x baby C batteries HR14 NiMH 1.2 V 5500 mAh are used. This because of the following reasons:
  + The charging of NiMH batteries with a solar panel is more straightforward and safer.
  + No voltage converters (buck converters) are needed. Although these can reach a efficiency of > 90%, in practical use efficiency can be as low as 60%. Instead the 5x 1.2V batteries can directly provide 6V for the servo’s. A low drop voltage regulator is used to provide 5.0V for the Raspberry Pi.
  + Using a branch at the fourth battery provides 4.8V. This is needed to power an ATmega328P processor which can put the ExoMy in deep sleep using as little as 100nA. This cannot be accomplished using a LiPo and an voltage converter.

There is also a disadvantage of the NiMH batteries. Although the energy content is about the same as for the LiPo battery, the weight is about 120 grams more.  
  


Figure 1: Power board, created with EasyEDA



Figure 2: Power board PCB, manufactured at JLPCB

* Two headlights are added so ExoMy can also see at night.
* Rubber 4 mm diameter O-rings are placed around the wheels. This to give ExoMy a smooth ride on hard surfaces. On soft surfaces like sand the O-rings sink into the ground and the teeth of the wheels will provide the grip.

## Additions / changes in software

* The gui/index.html and gui/style.css were adapted to have a bigger video image and to display some statuses like battery, solar panel and wifi status.
* In docker/Dockerfile some additional installations are done:
  + SMBus needed for I2C to communicate with the ATmega328P processor. This script needed smbus to be installed.
  + OpenSSH for communication between the Docker container and the Raspberry Pi host.
* In docker/entrypoint.sh some lines are added to copy the SSH key pair to /root for authentication so SSH can be used without password by a Python script.
* On the Raspberry Pi host the public half of the SSH key pair is installed manually on the Raspberry Pi host. See <https://upcloud.com/community/tutorials/use-ssh-keys-authentication> for a description.
* In src/robot\_node.py additional code was added to publish some statuses like battery status, solar panel status and Wifi status to the ‘/battery\_status’, '/solarpanel\_status' and wifi\_status topic respectively every second.
* In src/robot\_node.py additional code is added to handle the messages published by the Web page (in index.html) 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. The Docker container uses SSH to send the shutdown command to the Raspberry Pi host. 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 sends the shutdown command with SSH to the Raspberry Pi host.
* 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.

# Topics

Below some specific topics are described, just to document things which took some effort to find out.

## Wifi configuration in /etc/wpa\_supplicant/wpa\_supplicant.conf

On the Raspberry Pi host use sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf to add the wifi SSID and password of the network to which ExoMy can connect.

## Accessing the ExoMy from outside the LAN

For accessing the ExoMy from outside the LAN it is convenient to give the Raspberry Pi a fixed IP address so port forwarding can be used. At the same time we want DHCP to be used on the ExoMy so it can also be used in other networks like in a phone’s tethering network. This can be done by letting the router assign a static DHCP IP address (192.168.1.42) to the Exomy, using its MAC address.  
For the Exomy three ports must be forwarded:

* Exomy web server port: 8000
* Exomy raw video port: 8080
* Exomy websocket port: 9090

## Communication between the Docker container and the Raspberry host

For some purposes communication between the Docker container and the Raspberry Pi host is needed. For example to shut down the Raspberry Pi host or to get the Wifi status. This is accomplished by using SSH. To enable Python to use SSH, SSH keys are installed in the Docker container (private and public) and on the Raspberry Pi host (public). See <https://upcloud.com/community/tutorials/use-ssh-keys-authentication> for a description.  
Installation in the container is added to the docker/entrypoint.sh script.  
The .ssh folder containing the public and private key is added to the .gitignore file.

## 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 on Raspberry Pi:  
  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.
* Measures the battery voltage and solar panel voltage for reporting on the web page.
* Has a trigger input to wake up from deep sleep mode.

## 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.

## Ultimaker Cura settings

The following printer settings were changed from the default for printing the ExoMy robot with the Creality 3D CR-20 Pro 3D printer.

* Printing Temperature set to 210°.
* Build Plate Temperature set to 60°.
* Layer Height set to 0.15 mm.
* If support is needed, Support Structure is set to Tree for easier removal.

## 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. This will not be used. Instead, low power will be reached using the ATmega328P processor which will completely switch off the power supply to the Raspberry Pi and the servo board.

### 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, measured 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://www.esa.int/Enabling_Support/Space_Engineering_Technology/3D_print_your_own_Mars_rover_with_ExoMy>

<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>

<https://upcloud.com/community/tutorials/use-ssh-keys-authentication>