The Great Guide to MORF

The ~~legacy~~ shame ☺ of Mathias Thor

Mathias@mmmi.sdu.dk

Content

Introduction

All of the papers presenting MORF and its control can be found here. Specifically, the master thesis on MORF (available here) can act as a great way to understand the system better.

If you experience any problems with the following or has additional questions, feel free to contact me at mathias@mmmi.sdu.dk

Turning on/off MORF

Turn on

1. Plugin a charged LiPo battery
   a. Plugin the flat connector first and then the **yellow** power connector
   b. The robot will turn off automatically if the LiPo battery is not charged enough.
2. Press the **red power** button and wait for the Intel NUC to turn on (shown by a green LED)
3. Press the **black button** and wait for the Intel NUC to boot

You can now:

- Connect to the robot with the XBOX joystick
- Connect to the wireless network called `morf-redpc` using the password: `morf1324`
  - Afterward, you may use `ssh morf-one@192.168.0.1` to connect to the NUC PC

Turn on

1. Press and hold the **black button** until the Intel NUC turns off
2. Press and hold the **red power** button until the green led turns off
3. Remove the battery and put it into a safe fire bag (silver bag)

MORF Color codes

The led array on the head of MORF can show different colors. Below are what the colors mean:

Yellow = motor driver is turned off and the motors are not stiff
Blinking yellow = start motor drivers before the controller (see XBOX Joystick shortcuts)
Green = motor driver is turned on and the motors are stiff
Red = motor driver is turned on and controller is running

XBOX Joystick shortcuts

↧+ X: Pressing down and X will turn on the motor driver
↧+ Y: Pressing down and Y will turn off the motor driver (note the motors will turn off)
↧+ B: Pressing down and B will turn off the controller
↧+ A: Pressing down and A will turn on the controller

Services

The Intel Nuc PC on MORF runs several services that start on boot. An example is the blink stick service (LEDs on the head of MORF) or the dynamixel servos.

All services are implemented in systemd (`/etc/systemd/system`). To implement a new service do:

1. write a .service file in /etc/systemd/system
2. write a bash script to be called form the .service file in /opt/...
3. if the service is to be called from a script using sudo; include the command in `/etc/sudoers.d/morf-one`
4. enable the service if it should start at boot (e.g., `sudo systemctl enable ros-T265.service`)

You can find all the services used currently here: https://github.com/MathiasThor/MORF

Updating software on MORF

To update the software on MORF (e.g., controller) we use ansible.

Install ansible

Install Ansible

- `sudo apt install ansible`

Ansible commands

Ansible works by using playbooks that specify what should be updated on the NUC pc.
For example, the following can be used to start and stop the locomotion controller placed on MORF:

- `ansible-playbook -i inventory morf_controller_start.yml`
- `ansible-playbook -i inventory morf_controller_stop.yml`

Note if you get an **error** about missing permissions when running the commands in the next section, then run the following: `ssh-copy-id morf-one@192.168.0.1`
The MORF playbook is placed in ~/workspace/gorobots/utils/morfscripts/ansible/Playbook.

Update Controller

Use the following playbook to update the controller of MORF:

- `ansible-playbook -i inventory morf_transfer_controller.yml`

This will transfers a compiled controller (binary file) from your PC:
`~/gorobots/projects/morf/demo/real/catkin_ws/src/bin/morf_controller_real`

to the NUC host PC (on MORF) at:
`~/gorobots-mthor/projects/morf/real/catkin_ws/src/morf_controller/bin`

Use the following playbook to upload the demo controller to MORF (see The Demo Controller):

- `ansible-playbook -i inventory morf_transfer_demo_controller.yml`

This will transfers a compiled controller (binary file) from your PC:
`~/gorobots/projects/morf/demo/real/catkin_ws/src/bin/morf_controller_real`

to the NUC host PC (on MORF) at:
`~/gorobots/projects/morf/demo/real/catkin_ws/src/morf_controller/bin/morf_controller_real` *(REMEMBER TO COMPILE THIS CONTROLLER FIRST)*

Hardware interface

Start hardware interfaces:

- `ansible-playbook -i inventory morf_driver_start.yml`

Starts the ROS nodes for the hardware interfaces to the IMU, Dynamixel, and LED array.

Note: This command is automatically executed at boot.
Note: This will make the joints stiff, so remember to put the legs in a sensible position before running this command

Stop hardware interfaces:

- `ansible-playbook -i inventory morf_driver_stop.yml`

Stops the ROS nodes for the hardware interfaces to the IMU, Dynamixel, and LED array.

Note: This will loosen the joint, so make sure that the robot will not get damaged if this command is executed.

Update hardware interfaces on MORF:

- `ansible-playbook -i inventory  morf_transfer_drivers.yml`

Transfers drivers/hardware interfaces (i.e., ROS nodes) for the IMU, Dynamixel, and LED array from ~/catkin_ws/src/* on the host pc to ~/catkin_ws/src/* on MORF

Note: You need to compile the nodes afterward - see Compiling the driver workspace on MORF below.

Compiling the hardware interfaces workspace on MORF

- `./compile_drivers.sh` (should be executed on MORF's NUC pc via SSH)

Compiles the catkin workspace in `~/catkin_ws/` on MORF

ROS nodes:

At boot, the following hardware interfacing ROS nodes will be started:

dynamixel_ros_driver ROS node

Publishing

- joint_Positions (positions of all joints in rad (float32 array))
- joint_Velocities (angular velocity of all joints in rad/s (float32 array))
- joint_Torques (torque of all joints in Nm (float32 array))
- joint_ErrorStates (Error states (int))

Subscribes to

- multi_joint_commands (desired position in rad (float32 array))

Description:
Use multi_joint_commands to control the servos. The array should be arranged in the following format:

- `Float array: [ID1, ID1_DESIRED_POS_RAD, ID2, ID2_DESIRED_POS_RAD, ... , IDn, IDn_DESIRED_POSITION_RAD]`

Note that the desired positions are in radians and that the sign of the position tells the motor which way to rotate.

blinkstick_square_driver ROS node

Subscribes to

- set_all_led (Set color as RGB (ColorRGBA msgs))
- set_single_led (Set color of led A as RGB (ColorRGBA msgs))

Description:
In the set_all_led all LEDs are set accordingly to the **R**, **G**, and **B** values (0-255). **A** is not used.
In the set_single_led the LED specified by **A** (0-7) is set accordingly to the **R**, **G**, and **B** values (0-255)

sharp_distance sensor

Publishes

- Distance (in centimeters)

Description:
Publishes the distance measured by the sharp distance sensor on the head of MORF

Intel tracking camera

Publishes

- Video feed
- IMU
- Odometry
- and many more (you can get relative position and orientation)

Data/files to and from MORF

Usually, when I record data on morf (e.g., joint torques values), I edit the controller code to include functions that create and write the desired data to a file. Once the experiment is over and the file has been generated, you can download it from MORF using:

-
  ```
  scp morf-one@192.168.0.1:/path/to/file /path/to/destination
  ```

To upload a file use:

-
  ```
  scp /path/to/file morf-one@192.168.0.1:/path/to/destination
  ```

This is, for example, used when uploading new behavior weights for the CPG-RBFN controller.

Code repository

All of the code for MORF and its controller is placed in the gorobots GitLab repository:
https://gitlab.com/ens_sdu/gorobots

Motor pattern adaptation using the CPG-RBFN framework:

All of the code for the motor pattern adaptation mechanism can be found in the following directory:

- `gorobots/projects/C-CPGRBFN/`

Each of the project directories in this directory is self-contained, meaning that they do not rely on code outside their respective directories. The following explains each of the CPGRBFN project folders:

- CPGRBFN_compact
  - This was the first version of the CPGRBFN network with no modules. It only contains the open-loop controller as presented in this paper: https://mathiasthor.github.io/assets/pdf/generic_neural_locomotion_control_framework.pdf
- CPGRBFN_compact_v2
  - This improves and cleans the code from CPGRBFN_compact (above)
- CPGRBFN_feedback_v3
  - In this version, we introduce the closed-loop modules as presented in this paper: https://mathiasthor.github.io/assets/pdf/CPG_RBF_FB.pdf
- CPGRBFN_feedback_nature
  - A cleaned version for the Nature Machine Intelligence paper
- CPGRBFN_dil_v4
  - In this version, we combine the closed-loop modules (CPGRBFN_feedback_v3) with the DIL for frequency adaptation as presented in this paper: https://www.frontiersin.org/articles/10.3389/fncir.2021.743888/full
- CPGRBFN_BBO_v5
  - Compared different learning algorithms
    - `CMA-ES`
    - `PI^BB`
    - `PI^BB with covariance adaptation`
- CPGRBFN_continuous_v6

    - Displays continuous learning for the Autonomous lifelong learning project. Ideally, this may be transferred directly to a real robot. However, before this is possible, the learning loop needs to be set up on MORF. I.e., the machine learning code (python files) and the directories for the `.json files`.

Information on how the CPG-RBFN framework can be used is found in The CPG-RBFN framework.

Frequency adaptation using DIL:

The newest code for the DIL, as used in https://www.frontiersin.org/articles/10.3389/fncir.2021.743888/full, can be found in:

- `gorobots/projects/C-CPGRBFN/CPGRBFN_dil_v4`

The older code (where it calculates the error from the amplitude and not the shape as in the above) as used in https://mathiasthor.github.io/assets/pdf/DIL.pdf and https://mathiasthor.github.io/assets/pdf/RAL_DIL.pdf can be found here:

- `gorobots/controllers/neutron`

The CPG-RBFN framework

A comprehensive guide for using the CPG-RBF controller can be found here: https://github.com/MathiasThor/CPG-RBFN-framework

The following explain in short how the framework functions (*assuming you are working in CPGRBFN_feedback_v3*):

In the `machine_learning` directory, all the code for the learning algorithm is placed (e.g., pi^bb). This also includes the communication to the simulation program (CoppeliaSim) as well as the code for running several simulation instances in parallel to speed up learning. The main file in the `machine_learning` directory is `RL_master.py`. Here you may set up things like the learning rate, the behavior to be learned (e.g., pipe climbing, base behavior, obstacle reflex controller, etc.), type of robot, learning algorithm, and so on.

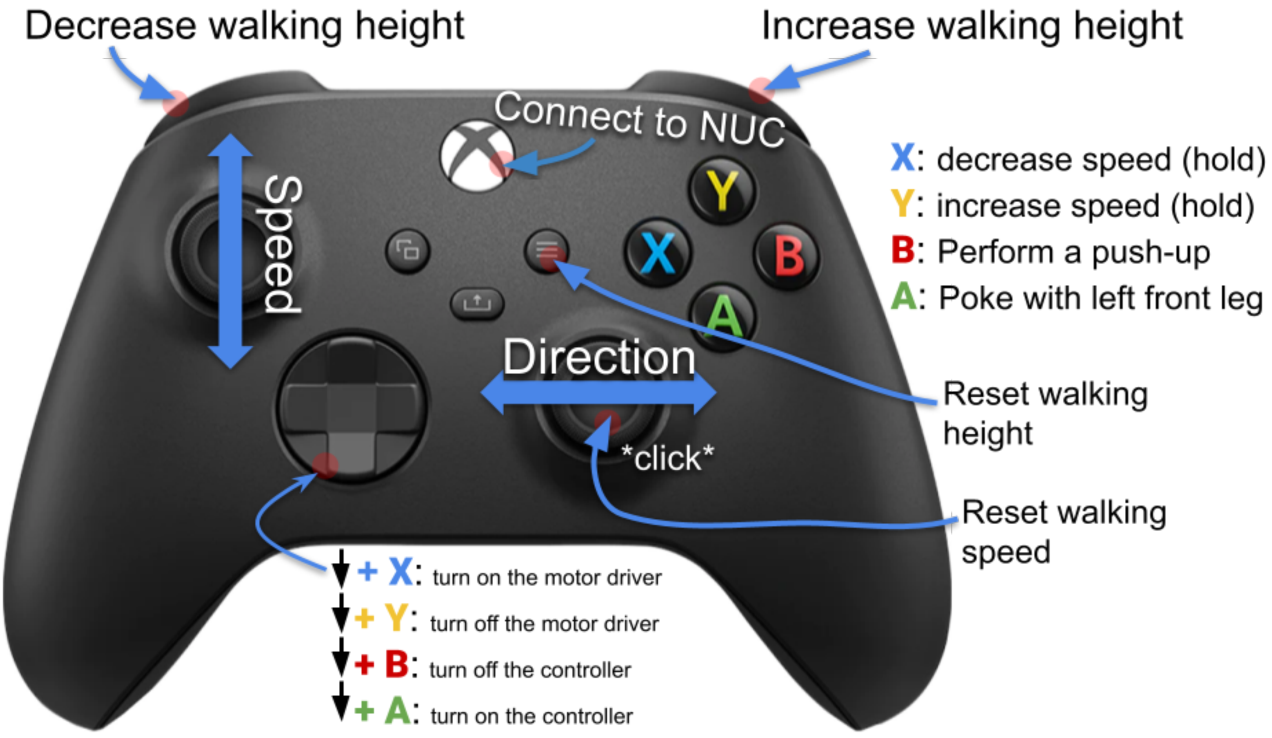
In `interfaces/morf/sim/morf_controller_script.lua` is the simulation script that is responsible for mimicking the real-life MORF robot and setting up the simulation environment. The script, therefore, creates ROS nodes, configures the simulation, and collects data that is sent to the controller or machine learning script (e.g., for calculating the reward).

In `neural_controllers/morf/sim/` is all the code for the controller running on MORF. This includes the CPG and RBF networks as well as DIL and any configuration of the robot (e.g., behavior mode, sensory feedback, and joint limits).

Note that when you want to learn a new behavior is has to be implemented in both the machine learning code and the controller code. This includes its reward function, behavior index, etc. A good approach is to look at the implementation of an existing behavior (e.g., direction) and use it as a template. Also, remember to suppress the other behaviors in `neural_controllers/morf/sim/neutronController.cpp` (by fixing the sensory input to the behaviors to 0 → e.g.: `postProcessedSensoryFeedback[behaviour_index][j] = 0;`) such that the other behaviors are not triggered during learning. Once a new behavior is learned, you can take the .json file from either `data/jobs/` or `data/storage/` (if you did not stop the learning before time) and place it in the `data` directory (probably you want to rename it at this point). Assuming you have implemented the new behavior correctly, you need to specify the directory of the new behavior in the readParameterSet function in `neutronController.cpp`. You may then implement the new behavior in `machine_learning/run_sim.sh` and run the script to test the behavior. It is not an easy process, but you will get the idea after some time with the code.

The Demo Controller

To show off the real MORF hexapod robot you may use the demo controller. It is simply using a CPG without any adaptation modules. Instead is has hardcoded behaviors that can be triggered with the XBOX controller. The commands are specified below:



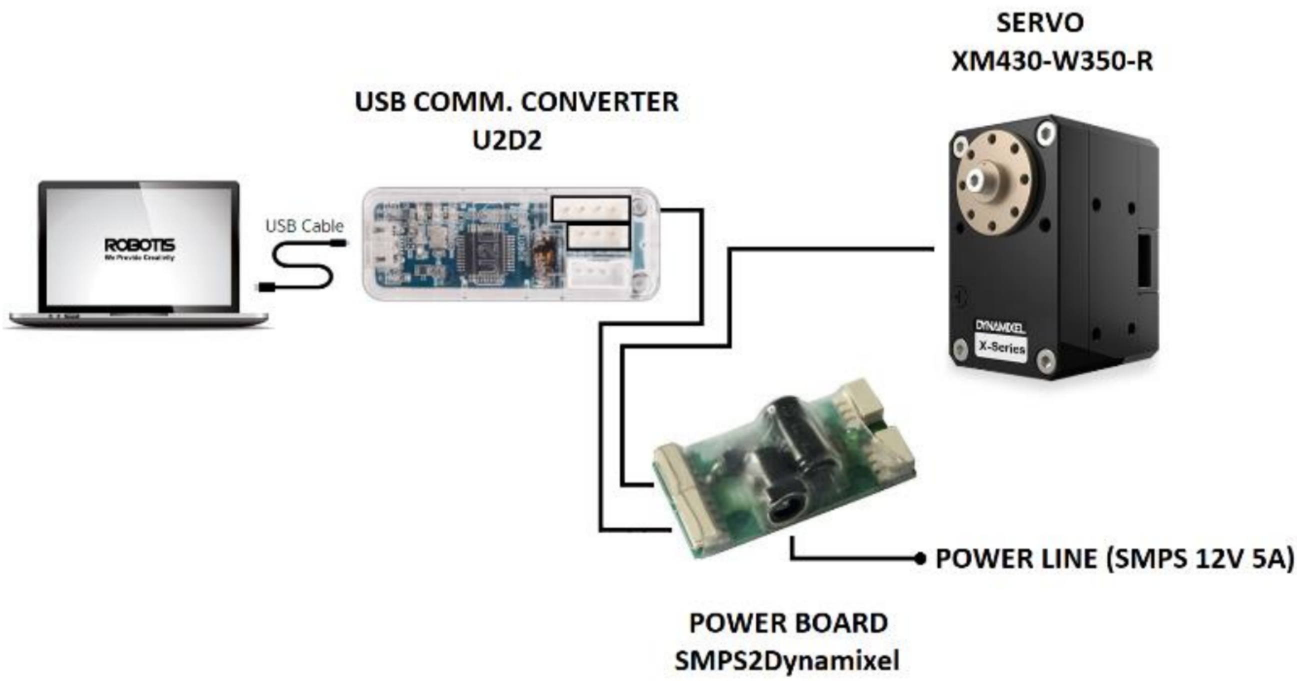
X: decrease speed (hold)
Y: increase speed (hold)
B: Perform a push-up
A: Poke with left front leg

↓ + **X**: turn on the motor driver
↓ + **Y**: turn off the motor driver
↓ + **B**: turn off the controller
↓ + **A**: turn on the controller

To upload the demo controller to MORF see Update Controller.

Dynamixel servo setup

The Dynamixel servos are all connected to a U2D2 (see image below) that connects to the NUC pc via USB. For testing a single servo it can also be connected to the U2D2 (and a power connector - see also the guide by Cao) that is then connected to your personal PC.



Installing Dynamixel drivers

Set USB latency

Start by setting the USB latency on the NUC or your personal PC for fast communication. This can be done by running the following command (assuming U2D2 is connected to USB0):

- `sudo usermod -aG dialout $USER && echo 1 | sudo tee /sys/bus/usb-serial/devices/ttyUSB0/latency_timer`

You can check it by running the following:

- `cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer`

Install dependencies

The following dependencies need to be installed on the nuc or your personal PC.

1. Install ROS
2. Dynamixel SDK and ROS Controller
    a. Run the following commands to install the remaining dependencies:
        i. `sudo apt-get install -y git cmake python-tempita python-catkin-tools python-lxml xsltproc qt4-qmake libqt4-dev libqscintilla2-dev`
    b. Run the following to install the Dynamixel Workbench used for communicating with the servos through ROS:
        i. `sudo apt-get install ros-melodic-dynamixel-sdk`
        ii. `mkdir ~/catkin_ws && cd ~/catkin_ws`
        iii. `mkdir src && cd src`
        iv. `git clone https://github.com/MathiasThor/my_dynamixel_workbench.git`
        v. `git clone https://github.com/MathiasThor/dynamixel-workbench.git`

        vi. `git clone https://github.com/ROBOTIS-GIT/dynamixel-workbench-msgs.git`
        vii. `git clone https://github.com/stonier/qt_ros`
        viii. `cd dynamixel-workbench-msgs && git checkout f91ae7dbd5d368a3121ca5bb901771b2e6471c01`
        ix. `source /opt/ros/melodic/setup.bash`
        x. `source /home/$USER/catkin_ws/devel/setup.sh`
    c. In order to add the above command to your .bashrc use the following command:
        i. `gedit ~/.bashrc`
    d. and add the following in the end of the file:
        i. `source /opt/ros/melodic/setup.bash`
        ii. `source /home/$USER/catkin_ws/devel/setup.sh`
    e. Finally, compile the dynamixel ros controller:
        i. `cd ../.. && catkin_make`
3. Setup automatic startup (not required)
    a. see Services

Now you can connect to a single or multiple servos using the following commands respectively:

- `roslaunch my_dynamixel_workbench_tutorial single_manager_4mil.launch`
- `roslaunch my_dynamixel_workbench_tutorial multiple_motor_test.launch`

Note the above is for bautrate 4000000. If standard bautrate use:

- `roslaunch my_dynamixel_workbench_tutorial single_manager.launch`

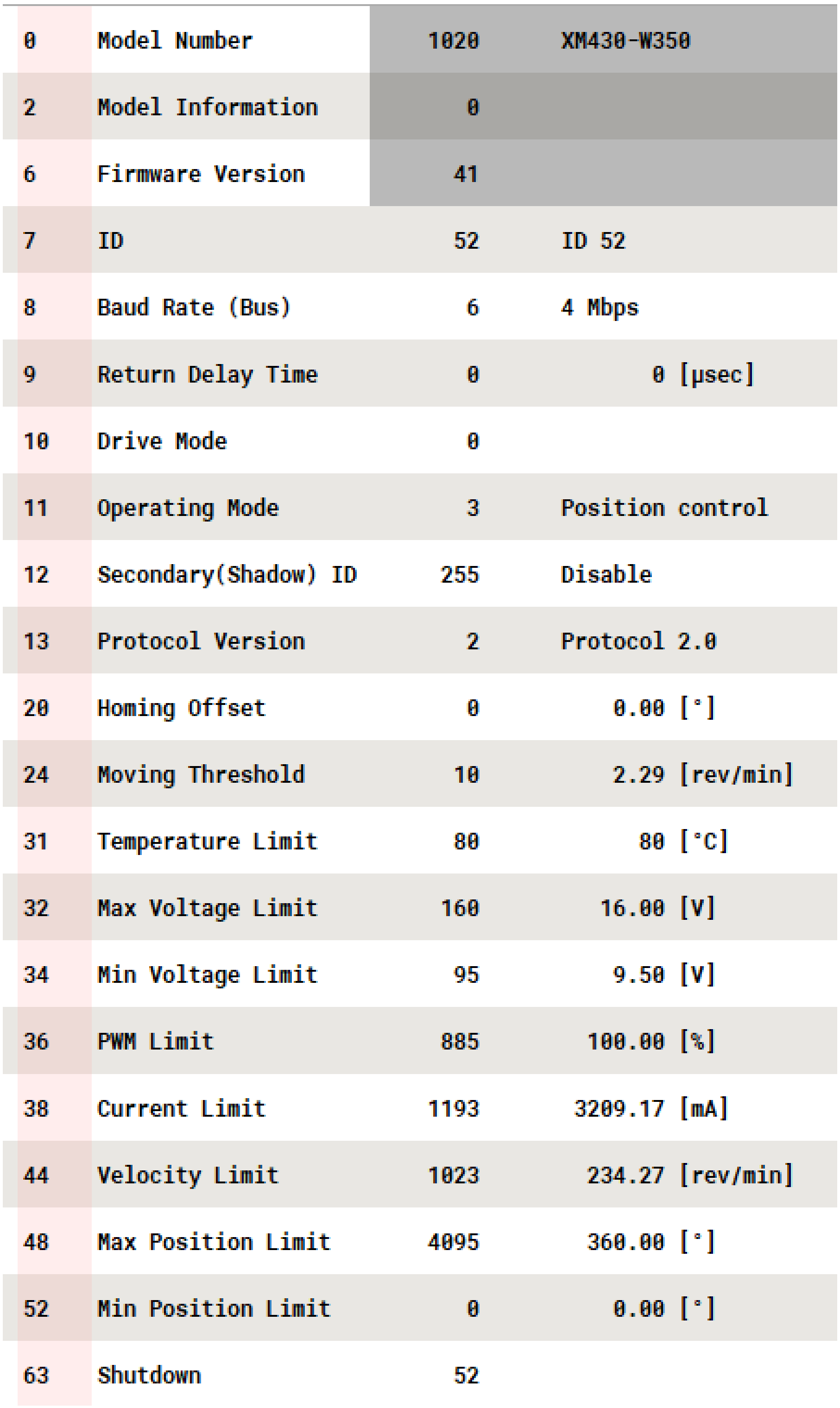
With the single_manager you can specify the values of the motor. After you see “`init success!`” hit enter. Now you can set all the parameters of the servo (see them all here: https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/). Alternatively, you may use the dynamixel_wizard app for Windows (https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/)

Note if you get an “error opening serial port!” error, run the following:

- `sudo chmod 777 /dev/ttyUSB0`

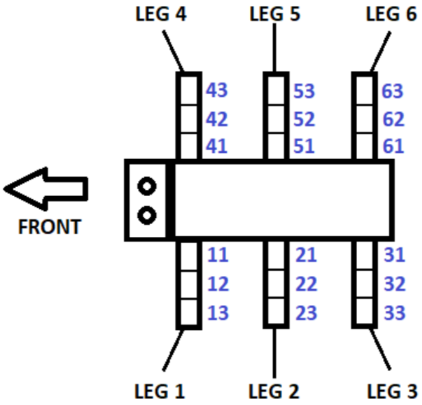
Dynamixel servos default parameters:

The dynamixel servos used on MORF are `XM430-W350-R`. Detailed information about the servos can be found here. Notice that the servo has a lot of changeable parameters. These are set as follows per default:



0	Model Number	1020	XM430-W350
2	Model Information	0	
6	Firmware Version	41	
7	ID	52	ID 52
8	Baud Rate (Bus)	6	4 Mbps
9	Return Delay Time	0	0 [μsec]
10	Drive Mode	0	
11	Operating Mode	3	Position control
12	Secondary(Shadow) ID	255	Disable
13	Protocol Version	2	Protocol 2.0
20	Homing Offset	0	0.00 [°]
24	Moving Threshold	10	2.29 [rev/min]
31	Temperature Limit	80	80 [°C]
32	Max Voltage Limit	160	16.00 [V]
34	Min Voltage Limit	95	9.50 [V]
36	PWM Limit	885	100.00 [%]
38	Current Limit	1193	3209.17 [mA]
44	Velocity Limit	1023	234.27 [rev/min]
48	Max Position Limit	4095	360.00 [°]
52	Min Position Limit	0	0.00 [°]
63	Shutdown	52	

Note that the IDs are set accordingly to the below figure:



The parameters can be changed using the ROS single manager (as described above) or by following the guide from Cao Danh Do (for windows users). Generally, the parameters will only be changed once when building the robot or replacing servos. Note: Pay attention to the small dot on the horn of the servos when replacing them!

Battery and charging

MORF requires 22.2V or a 6 cell lipo battery. Currently, it has a 3D printed battery holder for a Zippy Compact 25C Series 5800 (see image below) - however, you can print a new holder for another 6 celled battery.

MORF will automatically shut down when the lipo battery is drained. To charge the battery use a lipo charger (i used the blue Hyperion - see image below) and set C=5800mAh and charge with 5.8A.