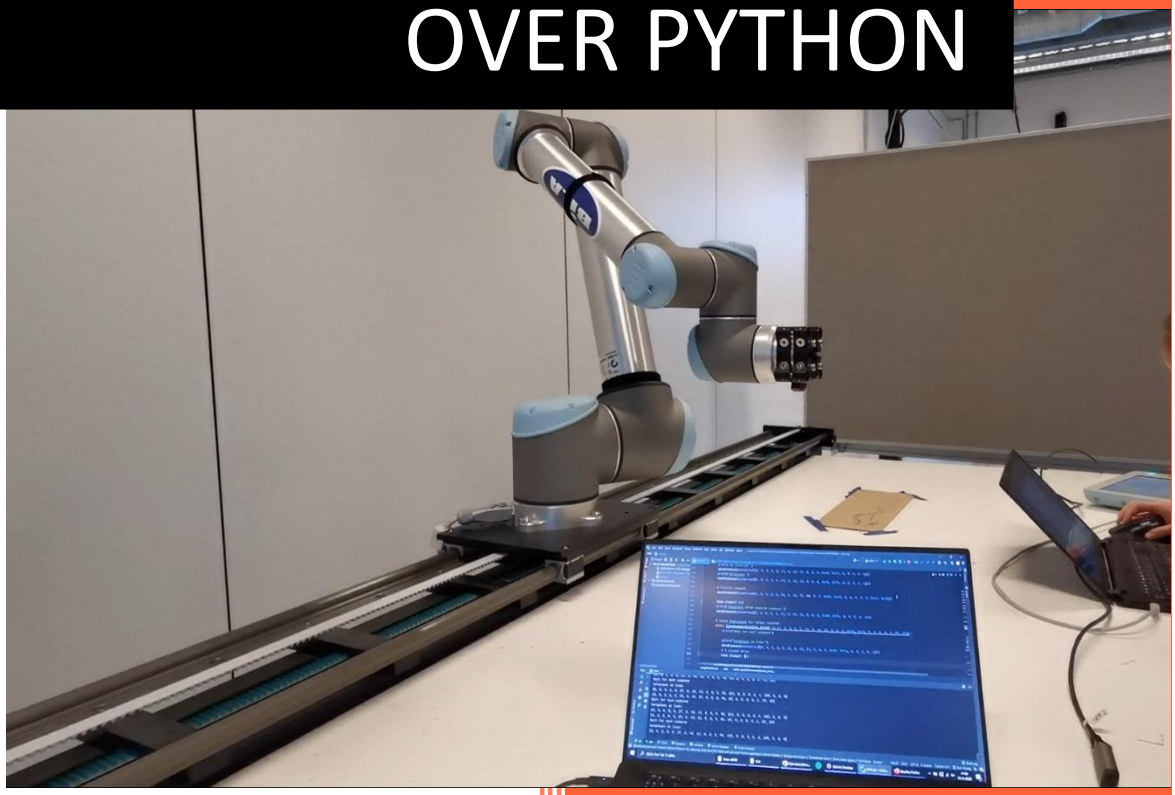
2022

HOW TO CONTROL THE DRYVE D1 OVER PYTHON



Elias Thomassen Dam, Nichlas Overgaard Laugesen

Aalborg University

24.11.2022

CONTENTS

HOW TO CONTROL THE DRYVE D1 OVER PYTHON

A GUIDE TO CONTROLLING THE RAIL – MADE BY DUMMIES FOR DUMMIES

Elias Dam and Nichlas Overgaard Laugesen

TLDR:

Our custom Python library is available on GitHub if you do not want to make it yourself. It automatically completes the start-up procedure and homing. You are then free to give position in millimetres and let it move. See the end of this document for more information.

INTRODUCTION

WHAT IS THE DRYVE D1

The Dryve D1 from Igus is a controller. In our lab it controls a single motor which is connected to a rail. The Dryve D1 is a computer, and to control the rail, you simply send commands to the Dryve. One way to send data is over Modbus TCP/IP, which is easy to implement in Python. The Dryve sits inside the UR10 controller and is a small box with orange accents.

WHAT IS MODBUS TCP/IP

Modbus TCP/IP is two standards coupled together. TCP/IP is an internet protocol, the same which makes sure a computer or mobile phone can connect to different servers in the

world. It is the transport layer, and takes whatever information you want to send, makes sure that it is formatted, and sent to the right address (IP address, which you might be familiar with). There are two computers acting on each end of the connection. One is the client, and one is the server. The difference is that the server is looking for incoming connections, and accepts them. The client tries to connect to the servers IP address. Once a connection is established, they are free to send data between them. In this case, the computer running Python is the client, and the Dryve is the server.

Modbus is a standard for how the data is structured. In our case, the data is sent as a 23-byte array. Each byte has its own meaning, depending on its position in the array.

HOW TO COMMAND THE DRYVE

Inside the Dryve there are several virtual objects. These objects are named in hex numbers, e.g. 6060h (the “h” at the end signifies a hex number). These objects contain several variables which relate to different parameters. For example, object 607Ah sets a target position.

There are two very important objects in the Dryve. These are 6040h and 6041h, named Control word and Status word, respectively. Control word is used to control essential parts of the system. This object can be written to and consists of 2 bytes → 16 bits. These bits are labelled from 0 to 15. Each of these bits control a single thing, and when writing to this object you are therefore changing things on a bit level, making it a little complex to read and understand it in plain text. The same thing goes for the Status word, but this object is read only. It tells you the status of the Dryve, e.g., if a target is reached, if it is switched on etc.

The byte array has this structure:

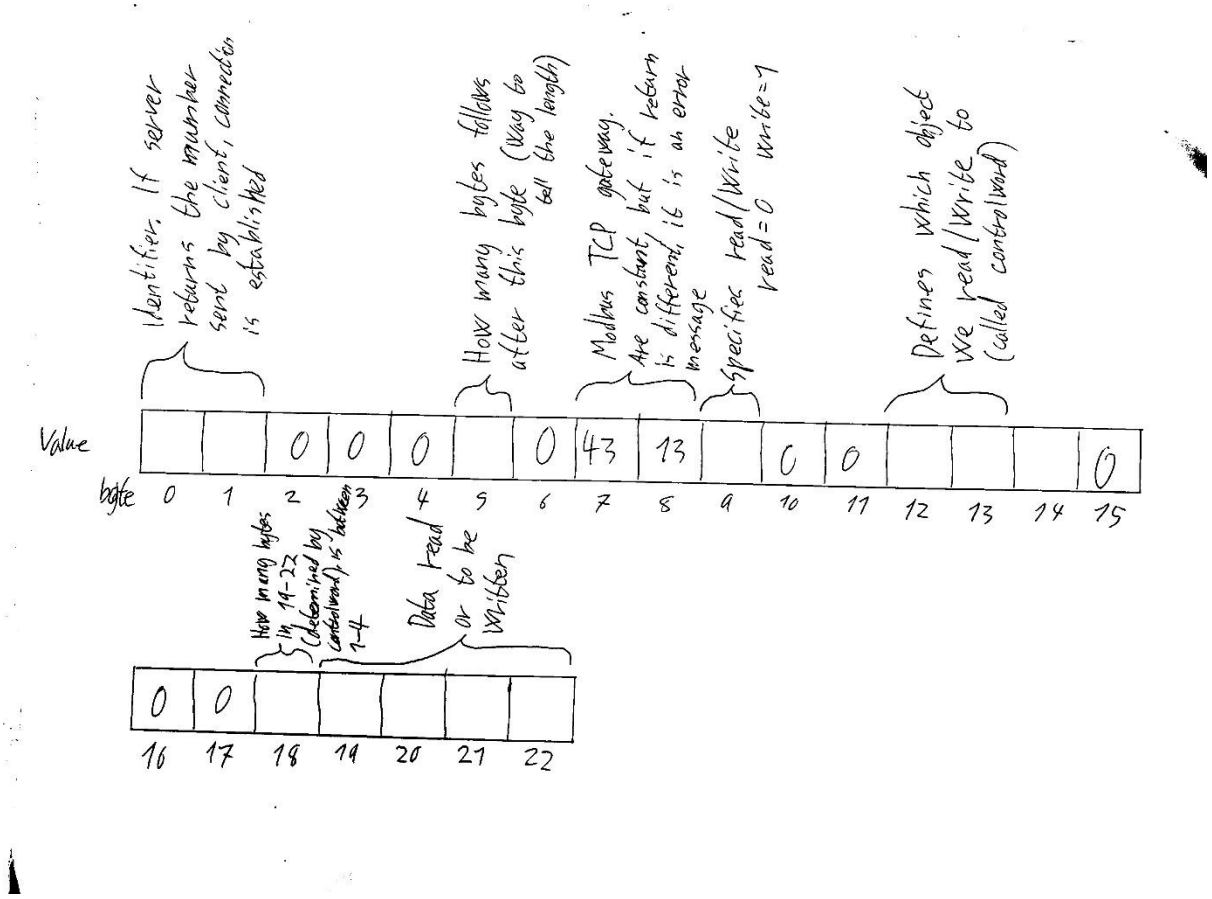


Figure 1 - 23 Byte array for Modbus - Transmission Control Protocol (TCP)

If a value is already determined in the array in the figure, it means the value is a constant.

- Byte 0 and 1 are checking numbers. They do not affect anything, but will be returned in the answer from the Dryve. Useful for checking that your information was received, although it is not necessary.
- Byte 5 is the amount of bytes that follows it. It is in the range between 13 and 17.
- Byte 9 determines if you are reading or writing to an object. 0 = read, 1 = write.
- In byte 12 and 13 you type which object you are writing/reading to/from. Objects are 4 digit long hex numbers, so you split them in two. E.g., if you want to write to object 6041, you type 60h in byte 12, and 41h in byte 13. Remember these are hex numbers, and not decimals. In Python, you write hex numbers with a leading “0x”: 0x60 and 0x41. If you want to write in decimals, you need to convert them first: 60h

= 96 and 41h = 65. In the end, this array is sent as actual bytes. Therefore, when you have written your array in Python, you then need to use the bytes() function to convert it before you send it.

- Byte 14 is a subindex. Some objects have sub objects, and this byte is used to access them. In every other case, leave this as 0.
- Byte 18 is the amount of data that follows it. In a write statement you know this yourself. In a read statement, it is how many bytes you the object contains (this can be read in the documentation).
- Byte 19 to 22 contains the information that you want to write, or which is being read. See the documentation for formatting and what information lies on which bytes or bits.

HOW TO GET STARTED

Before you can begin, these things need to be in place:

- UR10 turned on. It supplies power to the Dryve. Note that the E-stop button on the UR10 teaching pendant also stops the rail.
- Turn on all the analogue and digital outputs on the UR10 teaching pendant, and then turn them off again. This will eliminate the E08 error upon start up. (There is probably a specific combination of outputs that need to be toggled, but we have not found this. Turning them all on and off resolves the issue every time).
- Connect to the Dryve from your computer via an Ethernet cable (either directly or through a switch/router).
- Set digital input 7 to high in the user interface (how to access this interface is described later).

HOW TO CREATE A TCP/IP CONNECTION

For this, you need 3 things.

1. The Dryve IP address.
2. The Dryve port number.
3. The “socket” library in Python.

The IP address is displayed on the small screen on the Dryve once you connect an ethernet cable to it. Note that it is only displayed once. The port number can be retrieved from the Dryve user interface. The socket library is a default library, and can be retrieved by writing “import socket” in Python.

The Dryve has an user interface, which can be accessed by entering its IP address in a browser. Note that if HTTPS is turned off inside the interface, some browsers won't connect. In our experience, Chrome denies an unsecure non-HTTPS connection, while Firefox allows it if you tell it so. Inside the interface you can see status and change a lot of the objects directly through the GUI.

Once you have all the information you need, create a socket object in Python, and then connect. In our case, "s" is our socket object, and will be used every time we send something over the TCP/IP connection.

```
import socket
import time

HOST = "172.31.1.101"
PORT = 503

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
```

Snippet 1 - Import and setup

HOW TO SEND BYTE ARRAYS

Firstly, format your array with either decimals or hex numbers (or a combination of the two). Then convert it to a byte array. See figure.

```
shutdown = [0, 0, 0, 0, 0, 15, 0, 43, 13, 1, 0, 0, 96, 64, 0, 0, 0, 0, 2, 6, 0]
shutdown_array = bytearray(shutdown)
```

Snippet 2 - Shutdown

Next you need to send the information. For this you can copy this function provided in a manufacturer example:

```
#Definition of the function to send and receive data
def sendCommand(data):
    #Create socket and send request
    s.send(data)
    res = s.recv(24)
    #Print response telegram
    print(list(res))
    return list(res)
```

Snippet 3 - Socket setup

Just input your converted byte array into this function. You will see that the Dryve sends a respond message in your terminal. Hooray! Now you can read and write to objects in the Dryve.

HOW TO CORRECTLY START UP THE DRYVE

The Dryve has a rather complicated start up routine, as described in the documentation. This is not handled automatically. You must implement this into your code.

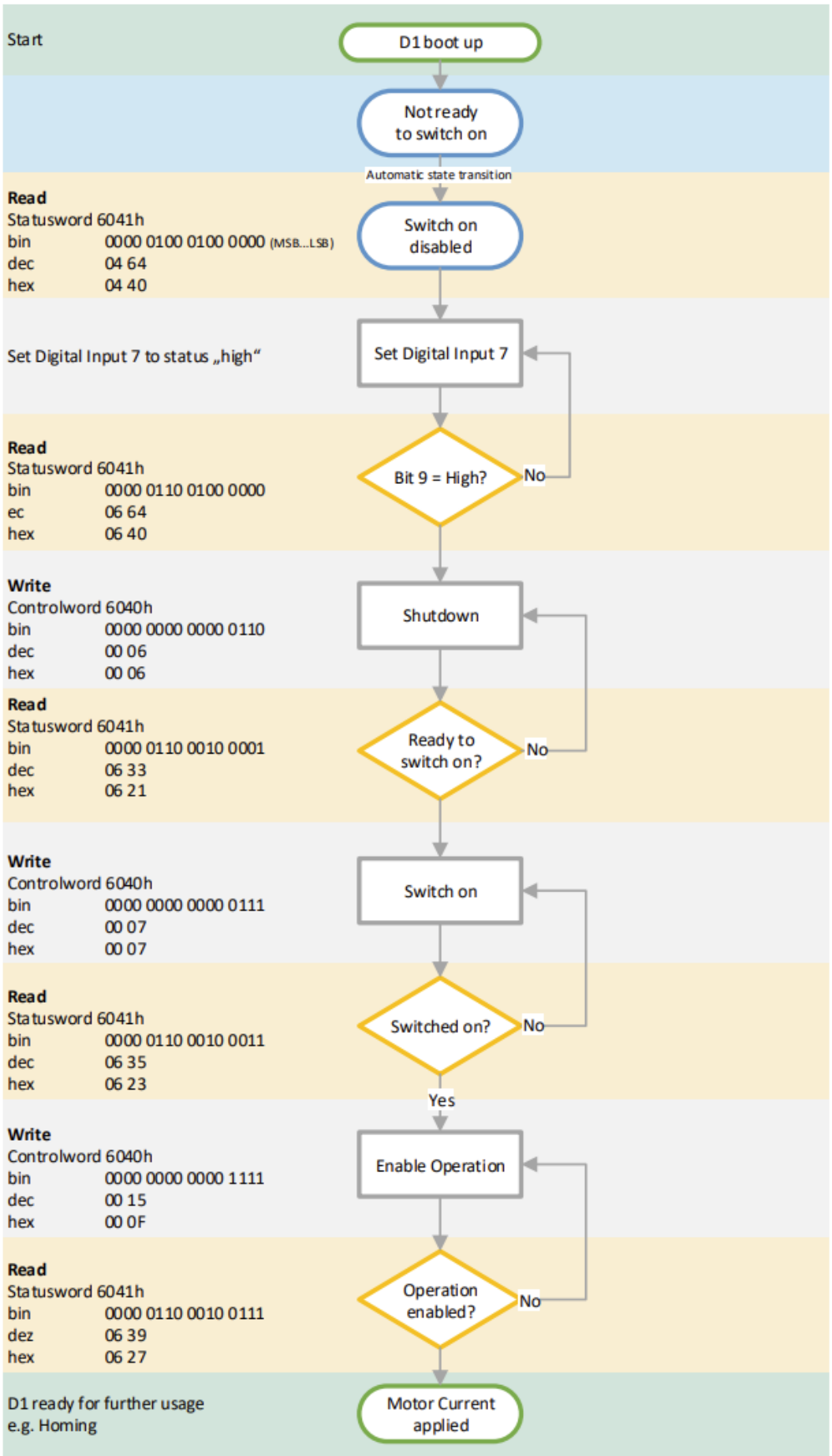

Start
D1 boot up
Not ready to switch on
Automatic state transition
Read
Statusword 6041h
bin 0000 0100 0100 0000 (MSB...LSB)
dec 04 64
hex 04 40
Switch on disabled
Set Digital Input 7 to status „high“
Set Digital Input 7
Read
Statusword 6041h
bin 0000 0110 0100 0000
ec 06 64
hex 06 40
Bit 9 = High?
No
Write
Controlword 6040h
bin 0000 0000 0000 0110
dec 00 06
hex 00 06
Shutdown
Read
Statusword 6041h
bin 0000 0110 0010 0001
dec 06 33
hex 06 21
Ready to switch on?
No
Write
Controlword 6040h
bin 0000 0000 0000 0111
dec 00 07
hex 00 07
Switch on
Read
Statusword 6041h
bin 0000 0110 0010 0011
dec 06 35
hex 06 23
Switched on?
No
Yes
Write
Controlword 6040h
bin 0000 0000 0000 1111
dec 00 15
hex 00 0F
Enable Operation
Read
Statusword 6041h
bin 0000 0110 0010 0111
dez 06 39
hex 06 27
Operation enabled?
No
D1 ready for further usage
e.g. Homing
Motor Current applied

Figure 2 - State Machine Visualization after Boot Up

This is just a matter of changing several objects in the right order. You can either follow this diagram or copy the start-up code from our library. It is a matter of sending messages and waiting for the correct responses.

IMPORTANT: FOR CORRECT MOVEMENT, LOAD THE CONFIGURATION FILE FROM OUR GITHUB INTO THE DRYVE. IT IS IMPORTANT THAT GEAR RATIOS AND FEED CONSTANTS ARE SET CORRECTLY. THESE DO NOT CHANGE, GIVEN THAT THE RAIL AND CONTROLLER IS THE SAME AS AT THE TIME OF WRITING. THE CONFIGURATION FILE CAN BE LOADED IN THROUGH THE USER INTERFACE.

HOW TO START HOMING

If you wish to use absolute positioning, like our library lets you do, you need to home first. This is simply telling the Dryve to find and/or set a home position, which it calculates all the following positions from. Firstly, you must set the robot into homing mode. This is done by accessing the object 6060h and writing 6 into it. Alternatively, use this function, input variable “6” into it. It also checks that the correct mode has been set before letting go.

```
def set_mode(mode):

    #Set operation modes in object 6060h Modes of Operation
    sendCommand(bytearray([0, 0, 0, 0, 0, 14, 0, 43, 13, 1, 0, 0, 96, 96, 0, 0, 0, 0, 1, mode]))
    while (sendCommand(bytearray([0, 0, 0, 0, 0, 13, 0, 43, 13, 0, 0, 0, 96, 97, 0, 0, 0, 0, 1])) != [0, 0, 0, 0, 0, 14, 0, 43, 13, 0, 0, 0, 96, 97, 0, 0, 0, 0, 1, mode]):

        print("wait for mode")

        #1 second delay
        time.sleep(1)
```

Snippet 4 - Sending of array, changing mode of track

Now is the time to decide how you want to home. See the documentation for all options. The rail includes two sensors at each end, and these are perfect for homing. They are called limit switches. If you intend to use them, make sure they are set to low in the user interface, as they then will give a positive signal when activated.

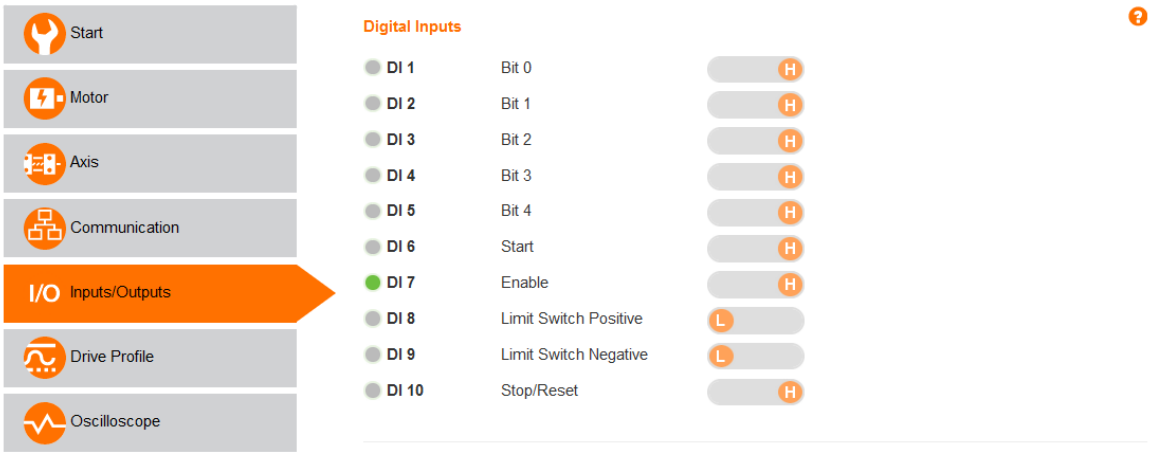


Figure 3 - IO digital Pins

Component 1 - Limit Switch Negative

We use the limit switch negative method. This means the robot will home until it finds the limit switch negative, and then set the home position here. You choose the method by writing to object 6098h (see documentation for homing). You also need to set necessary velocities and acceleration. Once all the parameters are set, you can execute by writing to the control word, and setting bit 4. It will then execute.

```
# Start Homing 6040h
sendCommand(bytearray([0, 0, 0, 0, 0, 15, 0, 43, 13, 1, 0, 0, 0x60, 0x40, 0, 0, 0, 0, 2, 0x1f, 0]))
```

Snippet 5 - 6040h disable pin 4

Check the statusword for it to confirm that the target is reached, and it is finished. It can be accomplished by this loop:

```
while (sendCommand(status_array) != [0, 0, 0, 0, 0, 15, 0, 43, 13, 0, 0, 0, 0x60, 0x41, 0, 0, 0, 0, 2, 39, 22]):
    print("wait for Homing to end")
    # 1 second delay
    time.sleep(1)
```

Snippet 6 - 6041h

Once homing is complete, enable operation again. This can be done by sending this array:

```
enableOperation = [0, 0, 0, 0, 0, 15, 0, 43, 13, 1, 0, 0, 0x60, 0x40, 0, 0, 0, 0, 2, 15, 0]
enableOperation_array = bytearray(enableOperation)
```

Snippet 7 - 6040h enable pin 4

HOW TO MOVE THE RAIL

Once you want to move the rail, you must switch mode again. We use position-based movement, and therefore switch mode to 1 in object 6060h.

Start by defining your position in object 607Ah. This object can accept 4 bytes. It depends on which SI unit settings have been set, but with our configuration file the distance is given in millimetres. Since the rail is only a couple of meters long, we only write to 2 of the 4 available bytes. Then define velocity, acceleration and deceleration in objects 6081h, 6083h and 6084h. Make sure to define these below the max velocity and acceleration (set in the user interface).

Once all parameters are set, execute and wait for the statusword to report a finished movement.

```
# Check Statusword for target reached
while (sendCommand(status_array) != [0, 0, 0, 0, 0, 15, 0, 43, 13, 0, 0, 0, 0x60, 0x41, 0, 0, 0, 0, 2, 39, 22]):
    print("Wait for next command")

    print("targetpos in loop:")
    sendCommand(bytearray([0, 0, 0, 0, 0, 13, 0, 43, 13, 0, 0, 0, 0x60, 0x7a, 0, 0, 0, 0, 4]))
    # 1 second delay
    time.sleep(1)
```

Once this is done, enable operation again. Congratulations, you are in full control of the rail!

USING OUR READY-TO-USE LIBRARY

If you just want something that works as fast as possible, or don't want to implement this yourself, you are free to use our library. This library contains five functions which handle everything.

Still, you need to make sure that the configuration file loaded in the Dryve is the one supplied in our Github. Furthermore, you might need to disable the E08 error as described and enable digital input 7.

At the start of your program you need to run the dryveInit() function. Beware that the robot will start homing, and therefore move to the limit switch negative. From here, the Dryve is ready to accept commands. All measurements are in millimetres.

- targetPosition() lets you designate a target position the robot will move to.
    - *NOTE: Highest possible value is 1950mm. Do not enter values below 1 mm, as it will trigger the limit switch, and give an error that needs to be reset in the user interface.*
- profileVelocity() lets you change the velocity in the targetPosition() function.
- targetVelocity() is a mode which lets you set a target velocity, and the robot will start moving with this velocity.
    - *NOTE: The robot does not stop unless you set the target velocity back to 0.*
- getPosition() retrieves the current position.

To install the library, you can use pip. Just type "py -m pip install createdryverail" in your command terminal.

GitHub with code and configuration file: https://github.com/Nicher1/Dryve-repository-for-dummies

Good fortune in your endeavours!

Created by Elias Thomassen Dam and Nichlas Overgaard Laugesen