Controlling the Fischertechnik TXT interface with ftrobopy

Martin Vogelaar

Version 1.0, September 2018

Table of Contents

1 Web sources

[1] Downey, Allen B., Think Python, a free Python tutorial at Green Tea Press, http://greenteapress.com/thinkpython/thinkpython.pdf

[2] Jacob, Raphael, Programmierung des TX-Controllers mit Python (German), ftpedia-2018-2, p60-66, https://www.ftcommunity.de/ftpedia_ausgaben/ftpedia-2018-2.pdf

[3] King, Peter, Tips on using the fischertechnik TXT Controller (English), ftpedia-2016-3, p31-32, https://www.ftcommunity.de/ftpedia_ausgaben/ftpedia-2016-3.pdf

[4] Fischertechnik Community Firmware Home page (English) https://cfw.ftcommunity.de/ftcommunity-TXT/en/

[5] Fischertechnik Community Firmware documentation in PDF (German) http://chobe.info/dokus/firmware.pdf

[6] Fischertechnik Robotics pages (select TXT) (English), https://www.fischertechnik.de/en/service/downloads/robotics

[7] *ftrobopy,* module on Githib (German) : https://github.com/ftrobopy/ftrobopy

[8] *ftrobopy m*anual (German): https://github.com/ftrobopy/ftrobopy/blob/master/manual.pdf

[9] *fttxpy*, module on Github, (English): https://github.com/ski7777/fttxpy

[10] Stuehn, Torsten, Programmierung des TXT mit Python (German), ftpedia-2017-2, p58-62, https://www.ftcommunity.de/ftpedia_ausgaben/ftpedia-2017-2.pdf

2 Description of this document

After buying the Fischertechnik (FT) TXT controller, we felt the need to document two aspects of working with this interface. These are:

1. Connections necessary to read sensors and to control actuators but also wired or wireless connections to a PC.
2. Programming the TXT with Python

A lot of the provided information can be found on the Internet but the information is scattered and not always easy to find. The level of the information in this document differs per section. Some sections can be used as a starter's guide, others require some knowledge about the subject.

The document is also not a Python course nor a course for learning how to master Linux, but the code in most of the scripts in this document is easy to follow. The reason we focus on Python is related to the fact that this language in its core is simple, and versatile. Another reason is that Python is still underexposed in the world of Fischertechnik interfaces.

A lot of very interesting functionality is available as Python modules. With the availability of Python distributions for science and engineering, like Anaconda (https://www.anaconda.com/download), it became easy to install a very powerful software environment on your workstation or laptop running either MacOS, Windows or Linux. The TXT runs Linux on an ARM processor so the interface is appealing to Linux users and a challenge to discover what it can and cannot do. Most of the descriptions related to Linux will also apply to MacOS.

We describe both online- and offline access to the TXT. In offline mode the code runs on the TXT itself. To enable this feature we downloaded the Fischertechnik Community Firmware (CFW).

Topics we will discuss:

- Installation and use of module ftrobopy
- Example scripts for ftrobopy
- Examples of connections with sensors and actuators
- Connections between TXT en PC, WLAN (Access Point), WLAN (Client), Bluetooth, USB
- Module *ftrobopy* and Windows 10
- Multi threading
- Graphical user interface with PyQt5
- Plotting data with Matplotlib
- Non blocking user input (process keyboard keys)
- Fischertechnik community software, pros and cons
- Access to the TXT with the community software (offline or direct mode)
- Programming the TXT display with PyQt4/PyQt5

3 Connecting the TXT to a computer

3.1 Supported connection types

An important feature of the Fischertechnik TXT interface is the option to connect to it in different ways. In this document we focus on Linux, but most of the information applies to other operating systems too. The possible connections to a PC are:

1. A USB cable connection
2. Wireless connection between PC and TXT
3. Wireless connection with Bluetooth
4. Wireless connection with the TXT as WLAN client

3.2 Windows

For Windows, the connections are set in ROBOPro. The connection type is selected in the ROBOPro COM/USB menu. Other software might require manual setup. For Windows the step are comparable to those for Linux as described below. Command `ping` is available on a Windows prompt.

3.3 Linux

The connections from a Linux PC can be a bit more tricky but the all work fine. I tested these connections on a laptop with Ubuntu 16.04 LTS.

1. **Connection with USB cable.** This is the fastest way to transfer data. No setup is required. The IP is 192.168.7.2. So without any additional software, one can ping the TXT on a command line. with

   ```
   $ ping 192.168.7.2
   ```

2. **Wireless connection with TXT as access point**. This connection type is useful when you only need/want a connection between PC and TXT. WLAN must be setup as Access Point (AP). On the display of the TXT in Network->WLAN setup, the IP 192.168.8.2 will appear. Also a 12 character long password is displayed on the TXT display as AP network security key.
   In the list with available wireless connections on your PC, find and select the connection with an SSID similar to **ft-txt_7640**. Here you need to enter the network security key.
   If you encounter a problem with setting a network address, then (at least for Ubuntu) network configurator, select the connection, select edit with the right mouse button and make sure that the mode is set to *Infrastructure* and not *Access Point.*
   One can test the connection with the ping command:

   ```
   $ ping 192.168.8.2
   ```

3. **Wireless connection with Bluetooth.** Compared to WIFI, this type of connection is less secure, more sensitive to interference and it has a lower bandwidth. The advantage is that it usually easy to make a connection and it can be u useful option if there is no WIFI available.
   In the Network setup on the TXT, select Bluetooth
   You get a pairing code on the TXT display which is needed to pair your TXT with your PC.

In the list with Bluetooth devices on your PC you should see a device name like ROBOTICS TXT 7640
Make sure that your default Bluetooth adapter is visible to other devices. Usually you will get a warning when it is not. In the Bluetooth configuration we marked the TXT device as *Trusted*.
When you scan for new devices, the **TXT will show up as a Bluetooth network** (i.e. not a regular device like a mobile phone or sound bar). Connection attempts from the Bluetooth configuration GUI will not succeed. Instead, we had to turn off WIFI connections and select in the network manager the network provided by TXT interface. After a successful connection, we were able to connect to WIFI again and use both networks active at the same time.
The connection was tested with:

```
$ ping 192.168.9.2
```

On our test machine we saw that there is sometimes some delay and packet loss. Probably this is due to interference.

4. **Wireless connection with the TXT setup as WLAN Client.** This is the connection type where one adds the TXT to the devices which can be addressed by your router. Then you keep access to the Internet and at the same time you can control the TXT. The setup is similar to adding a mobile phone or tablet to your network.
   You need to configure WLAN on your TXT. Select option WLAN Client as the WLAN Mode.
   Go to: Home → Settings → Network → WLAN Setup → WLAN Mode
   Select WLAN-Client and press WLAN Client Setup. Press Scan to find your router.
   Enter the key for your network. Restart the network by pressing the check box, it restarts the network on the TXT. You have a successful connection if an IP appears in the Clients field. Note that in the client field, the TXT should report an IP in the range you router is managing e.g. `192.168.1.242`
   For experiments it is convenient to make the IP static in your router.

8.4V battery pack

9V FT power supply

Note: If you experience connection problems, it sometimes helps to restart the TXT and/or to reconnect your PC to your router.

The TXT works ok with the 8.4V battery set as in the picture above, but for testing purposes it is more stable to use the 9V power supply

We configured the TXT for several access points and experienced that signal strength is important and that at increasing distances from your wireless router, the connection can be troublesome (e.g. cannot get an IP or package loss is significant)

Fischertechnik provides an illustrated short manual for WLAN connections. Currently the URL is:
https://www.fischertechnik.de/-/media/fischertechnik/fite/service/downloads/robotics/txt-controller/documents/10-txt-wlan-client-mode.ashx

This provides a download option for a zip file with a German and an English version.
Test the connection on the command line with the IP address found in the Client field on the TXT with:

```
$ ping 192.168.1.242
```

or whatever IP was shown.

4 Installing Python module ftrobopy for Linux

To program the TXT we use the *ftrobopy* module made by Torsten Stuehn.
It is developed for Linux, but it has been setup in a way that the module also works in Windows (10), see next section.

The easy way to install it is with pip (`pip install ftrobopy`). We installed it from Github. There, the module is currently hosted and maintained. The URL is
https://github.com/ftrobopy/ftrobopy

On Linux one can clone the necessary software in a folder `ftrobopy` with command:

```
$ git clone https://github.com/ftrobopy/ftrobopy.git
```

Linux users can then use either their standard installation of Python or for instance an Anaconda distribution and then install the software with command:

```
$ python setup.py install
```

If you cannot install the module with this command then note that to use the module it is only necessary to have the file `ftrobopy.py` in the folder from which you run your scripts.
This software has been tested with Python 2.7 and Python 3.5 and higher.
As a first test we created a small universal test script that plays sounds on the TXT.
A *ftrobopy* script written in Python starts with the import of module `ftrobopy`.
This file is the required Python module to control the TXT. The Github clone also downloaded the documentation (*manual.pdf*) which is written in German but with enough English annotations and example code to be useful to people who don't read German.

***soundtest.py*: Simple test script, sounds on the TXT**

```
#------------------------------------------------------------------------
# Purpose: Script to check TXT connection and sounds.
# Date:    06-09-2018
# Author:  M. Vogelaar
#
# Remarks: This script should play sounds available to the TXT.
#          Note that a next sound can only be played when the first one
#          is finished. Therefore we check in a loop if the sound
#          was ended. Setting a volume works only when the connection
#          is in 'direct mode'.
# Tested:  Python 3, Ubuntu 16.04, Windows 10
#------------------------------------------------------------------------
import ftrobopy
txt = ftrobopy.ftrobopy('192.168.7.2', 65000)       # Connected by USB cable
#txt = ftrobopy.ftrobopy('192.168.8.2', 65000)      # Connected by WLAN AP
#txt = ftrobopy.ftrobopy('192.168.1.242', 65000)    # Connected by WLAN client
#txt = ftrobopy.ftrobopy('192.168.9.2', 65000)      # Connected by Bluetooth
print('Name of controller: ', txt.getDevicename())
print('Version:', hex(txt.getVersionNumber()))

for i in range(1,27):
      txt.play_sound(i, repeat=1, volume=10)
      while not txt.sound_finished():
          pass
```

4.1 ftrobopy on Windows 10

In this document we show code examples. Most of them are tested and work well in Windows 10. After the 'Spring update' of Windows 10 in 2018, interesting features have been added to Windows to facilitate the migration of Linux applications to Windows. For example
Unix Sockets Support, *OpenSSH* , *Curl* and *Tar* are supported after this update. Commands `curl`, `tar` and `ssh` are available on the Windows command prompt. Unix Sockets support is essential for modules like *ftrobopy* to function on Windows and it appears to work well.

The test in Windows 10 has been done with the Python distribution Anaconda3 5.2.0 (https://www.anaconda.com/download/) and *ftrobopy* 1.80. Make sure that in the Allowed Apps window in the Firewall settings, you select the check box for `python.exe` on your computer to allow it to communicate over the Internet without being stopped by the Firewall.

The *Anaconda prompt* application in Windows is similar to a Linux terminal. It sets the necessary paths to executables and provides an ideal environment for fast development of small applications.

5 Universal Inputs

5.1 Sensors- and actuators data sheets

A handy (but a bit hidden) booklet with information about sensors and actuators is the bundled data sheet `Compendium-Sensores-Actuadores-FT.pdf` which was found at:
https://fischertechnik.com.mx/index/php?controller=attachment&id_attachment=4
It gives a summary about the properties of the devices and shows pictures how to connect them.

5.2 Configure the input and outputs of the TXT with ftrobopy

Module *ftrobopy* provides several convenience functions to set up the TXT. These routines take care of the configuration of the interface but it gives valuable insights if you have some knowledge of the low level configuration procedure.

One can read the current status with function `getConfig()`. This function returns two lists. The first list contains the status of the outputs (motor **M1**..**M4** or single pole output **O1**..**O8**). The second list contains 8 tuples of two elements. The first element is the type of input and the second is the mode of the input (analog or digital). In the lists there are only numbers. An example:

```
M, I = txt.getConfig()
print("Outputs configuration:", M)
print("Inputs configuration:", I)

Outputs configuration: [1, 1, 0, 1]
Inputs configuration: [(1, 1), (0, 0), (3, 0), (1, 1), (1, 1), (1, 1), (1,
1), (1, 1)]
```

The numbers correspond to the following definitions in *ftrobopy*:

Inputs and ftrobopy convenience functions:

Mode		C_ANALOG	C_DIGITAL
Type	nr	**0**	**1**
C_VOLTAGE	**0**	`voltage()`, `colorsensor()` [color sensor]	`trailfollower()` [infrared trail sensor]
C_SWITCH	**1**		`input()` [push button switch, Magnetic sensors (reed contacts)]
C_RESISTOR	**1**	`resistor()` [photo resistor (LDR), heat sensor (NTC), potentiometer]	[photo transistor]
C_ULTRASONIC	**3**	`ultrasonic()` [Ultra sonic distance sensor]	

*A Python list is a sequence with **start index 0**, of which elements can be changed.*
*A Python tuple is a sequence with **start index 0**, of which elements can **not** be changed.*

So to change an input, we have to create a new tuple and insert that in the list. An example makes this more clear. If not, then note that module *ftrobopy* provides high-level functions to configure the TXT without the need of any knowledge about configuration function.

Outputs and ftrobopy convenience functions:

Type	nr	
C_OUTPUT	0	`output()` [light bulb, buzzer, electromagnet, solenoid valves]
C_MOTOR	1	`motor()` [classic motor, encoder motor]

Without the convenience functions one can read and set the configuration with functions `getConfig()`, `setConfig()` and `updateConfig()`. The last function is needed to transfer new settings to the TXT.

Function `getConfig()` returns two lists. The first contains the status of the outputs (always 4 elements). The second contains the status of the 8 inputs. The settings for the inputs are always given as a tuple per input. The first element in the tuple is the type of input (see table above) and the second element is the mode of the input (analog or digital).

The normal procedure is to use `getConfig()` first so that you have two lists with the required dimensions as a result. Then you can change elements in the lists and use `setConfig()` and `updateConfig()` to set and transfer the altered configuration. Note that one cannot change the content of a tuple in Python. Therefore, we need to create a new tuple to set new values for an input.

In the code below, we do exactly this to change the input **`I3`**. A tuple is a sequence between parentheses. So we need to define two elements for **`I3`**.

These basic functions all follow the standard Python way of indexing. So the first element in an sequence has index 0. The convenience routines address the inputs and outputs with their numbers on the TXT, i.e. 1 to 4 for the motors, 1 to 8 for the single pole outputs and 1 to 8 for the inputs. This can sometimes be a bit confusing.

In the next example we show all the actions to set an input to read data from a variable resistance configured in analog mode (LDR, NTC, potentiometer).

***configIOtxt.py*: read the configuration of the TXT**

```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to configure the TXT for I/O
# Date:    20-09-2018
# Author:  M. Vogelaar
#
# Remarks: The script shows a function to convert the status of the TXT
#          from numbers to text. It changes the mode for input 3 and shows
#          the changes.
#-------------------------------------------------------------------------------
import ftrobopy

def showConfig():
    M, I = txt.getConfig()
    Mt = []
    for i,m in enumerate(M):
        if m == 0:
            Mt.append("Output O%d and O%d"%(2*i+1, 2*i+2))
        else:
            Mt.append("Output M%d"%(i+1))
    It = []
    for i, inp in enumerate(I):
       if inp[1] == 0:
           mode = "Analog"
       else:
           mode = "Digital"
       if inp[0] == 0:
          It.append("Input I%d type: Voltage, mode: %s"%(i+1,mode ))
       if inp[0] == 1:
          It.append("Input I%d type: Switch or resistor, mode: %s"%(i+1,mode ))
       if inp[0] == 3:
          It.append("Input I%d type: Ultrasonic, mode: %s"%(i+1,mode ))
    return Mt, It

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable

M, I = txt.getConfig()
print("Type of return values M, I, I[0]:", type(M), type(I), type(I[0]))
print("Outputs configuration:", M)
print("Inputs configuration:", I)
Mt, It = showConfig()
print(*Mt, sep='\n')  # Unpack list and print with newlines
print(*It, sep='\n')

i3 = 2                                      # Note this is input I3
I[i3] = (txt.C_RESISTOR, txt.C_ANALOG)
txt.setConfig(M, I)
txt.updateConfig()
Mt, It = showConfig()
print("\nAfter configuration of I3:")
print(It[i3])
```

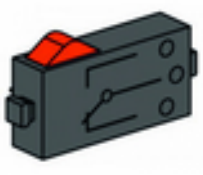
5.3 Reading the counter inputs

As an example, we want to read the counter value of c1 and display the value only when it changed. In the manual of *ftrobopy* we find the routines `getCurrentCounterInput()` and `getCurrentCounterValue()`.
With the first function we check if there ara counter values changed. The second function then reads that value. One can specify which counter (or all) are probed. Note that counter **C1** is the first counter but it is identified with value 0.
Connect **C1** to ground with a wire to increase the counter or use a pushbutton switch.

***getcounter.py*: Reading counter values**

```
#--------------------------------------------------------------------------
# Purpose: Script to get a counter value from the TXT (Python 3)
# Date:    06-09-2018
# Author:  M. Vogelaar
#
# Remarks: Demonstration how to read a value from one of the counter
#          inputs, c1, c2, c3 or c4. Here we read the value of c1
#          and print the counter value only when it has changed.
#          Script demonstrates use of endless loop. Abort on command line
#          with ctrl C
#          Note that the first counter c1 has index 0
#--------------------------------------------------------------------------
import ftrobopy

#txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
txt = ftrobopy.ftrobopy('192.168.8.2', 65000)       # Connected by WLAN AP
#txt = ftrobopy.ftrobopy('192.168.1.242', 65000)    # Connected by WLAN client
#txt = ftrobopy.ftrobopy('192.168.9.2', 65000)      # Connected by Bluetooth

print('Name of controller: ', txt.getDevicename())
print('Version:', hex(txt.getVersionNumber()))

c1 = 0
while 1:
    if txt.getCurrentCounterInput(c1):
        print(txt.getCurrentCounterValue(c1))
```

5.4 Digital input



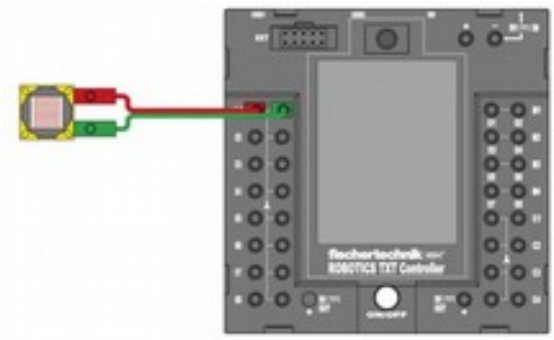
On the TXT, the inputs **I1**..**I8** can be used as digital inputs. In principle the values can only get the values 0 (no contact) or 1 if **I1** is connected to ground e.g. by pressing a pushbutton switch. Module *ftrobopy* provides a class input with a method `state()` which stores the current state open (0) or close (1).

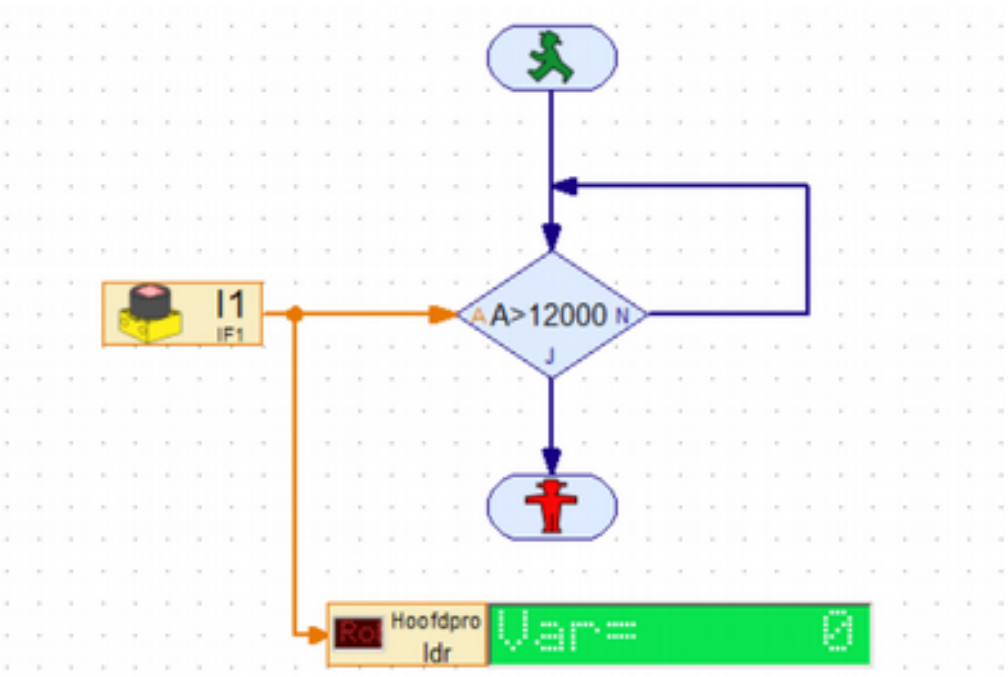
***digitalinput.py*: Digital input with a switch or reed contact**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to check the state of a digital input
# Date:    10-09-2018
# Author:  M. Vogelaar
#
# Remarks: Create an input object. This class has only one method, called
#          state() and this method returns 1 if the input is closed
#          or 0 when it is open.
#          There are two functions available here. One is method state()
#          for input object I1. Note that input 1 must be addressed with 1
#          and not 0 here.
#          The second function is getCurrentInput(). Here we need to specify
#          the index of the input and input 1 corresponds to 0.
#--------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB
cable

print("Change state of Input 1. Abort with ctrl-c")
I1 = txt.input(1)  # Now 1 means input 1 here. Be careful!
state = I1.state()
while 1:
   ns = I1.state()
   if ns != state:
      print("state(), getCurrentInput():", ns, txt.getCurrentInput(0))
      state = ns
   txt.updateWait()
```

5.5 Light-dependent resistance (LDR)

The LDR in our example is connected to **I1** and ground.
The ROBOPro setting of the input is Analog 15 kOhm.





***LDRreader.py*: Reading values from a light dependent resistor**

```
#------------------------------------------------------------------------
# Purpose: Script to demonstrate how to configure the input for a
#          light-dependent resistor (LDR) and to read its values.
# Date:    14-09-2018
# Author:  M. Vogelaar
#
# Remarks: In this script we configure input I1 as a resistance meter
#          with a range of 15kOhm. An LDR is connected to input 1.
#          Function getConfig() returns the current configuration for
#          the motors and inputs in a tuple. The second element is an
#          array with inputs. Input 1 has index 0.
#          For an LDR (or NTC) we use the configuration tuple
#          txt.C_RESISTOR, txt.C_ANALOG
#          Print a resistance value only when it has been changed.
#          The script stops when you cover the LDR (the resistance will
#          then be equal or bigger than 15kOhm
#------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable

i1 = 0
M, I = txt.getConfig()
I[i1] = (txt.C_RESISTOR, txt.C_ANALOG)
txt.setConfig(M, I)
txt.updateConfig()

ohm = 0
while 1:
   txt.updateWait()
   newohm = txt.getCurrentInput(i1)
   if newohm != ohm:
      print("Current resistance is", newohm, "Ohm")
      ohm = newohm
   if newohm == 15000:
      break
```

This approach does not use object orientation. Later we will show an example that does.
The difference between the ROBOPro program and the Python script is that we have to 'configure' the input. The LDR is suitable to read in analog mode (Use constant `C_ANALOG`). To set the universal input to 15 kOhm, you need to use the in *ftrobopy* defined constant `C_RESISTOR`. The inputs are configured in an array. Input `I1` has index 0 so we modify only `I[0]`.
The configuration is a Python tuple. The first element in the tuple is the input type and the second sets the input to digital or analog.

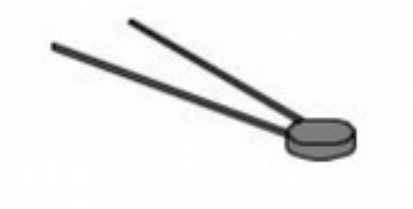
Module *ftrobopy* provides a class `resistor` for LDR's and NTC's. Using a resistor object, you don't need to configure the inputs of the TXT. It is automatically set to the right configuration. The object has a method `value()` which we use to read the current resistance. The code (see example below) is more clear than the example where we configure the inputs explicitly.

***LDRreaderOO.py*: Reading the LDR object oriented**

```
#-------------------------------------------------------------------------
# Purpose: Script to demonstrate how to create an resistor object
#          for a light-dependent resistor (LDR) and to read its values.
# Date:    14-09-2018
# Author:  M. Vogelaar
#
# Remarks: In this script we use the resistor class to read the values
#          of an LDR.
#          Print a resistance value only when it has been changed.
#          The script stops when you cover the LDR (the resistance will
#          then be equal or bigger than 15kOhm
#-------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connected by USB cable

R = txt.resistor(1)      # Here input 1 is 1

ohm = -1                 # Initialize to non-existing value
while 1:                 # Start endless loop
   txt.updateWait()
   newohm = R.value()
   if newohm != ohm:
      print("Current resistance is {:d} Ohm     \r".format(newohm), end='\r')
      ohm = newohm
   if newohm == 15000:
      break
```

Python 3 trick: In the example above we use the Python 3 `print()` *function with argument end with a carriage return character* `'\r'` *only to prevent a new line. The effect is that the output remains on the same line.*

5.6 NTC resistor 1.5kOhm



An NTC resistor (Negative temperature coefficient resistor or thermistor) are widely used as temperature sensors. Their main characteristic is that their resistance varies as a function of temperature: the hotter, the less resistance. The value can be read in the same way as the LDR. Method `ntcTemperature()` converts the measured resistance to a temperature.

***NTCreaderOO.py*: Reading data from an NTC resistor**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to create an resistor object
#          for a light-dependent resistor (LDR) and to read its values.
# Date:    14-09-2018
# Author:  M. Vogelaar
#
# Remarks: In this script we use the resistor class to read the values
#          of an NTC.
#          Print a temperature value only when it has been changed.
#--------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable

R = txt.resistor(1)   # Here input 1 has identifier 1 (is not array element)

temp = -1
while 1:
   txt.updateWait()
   newtemp = R.ntcTemperature()
   if newtemp != temp:
      print("Current temperature is {:.1f} C     \r".format(newtemp), end='\r')
      temp = newtemp
```

5.7 Potentiometer

A potentiometer is a manually adjustable variable resistor with 3 terminals.

It can be used in combination with the TXT to set speed of a motor or intensity of a lamp depending on the rotation of the stick.

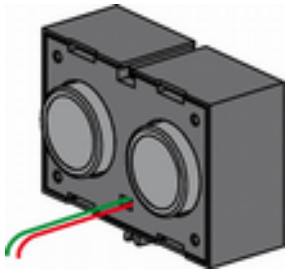
The FT potentiometer uses two of them. It can be connected to one of the inputs on the TXT and the ground. The FT potentiometer has a range from approximately 100 Ohm to 5100 Ohm.

The input is configured to measure resistance and its mode is analog. So we can use the function `resistor()` again as we did in the previous example, but let's do the exercise with function `setConfig()` again.

***potentiometer.py*: Read data from a potentiometer**

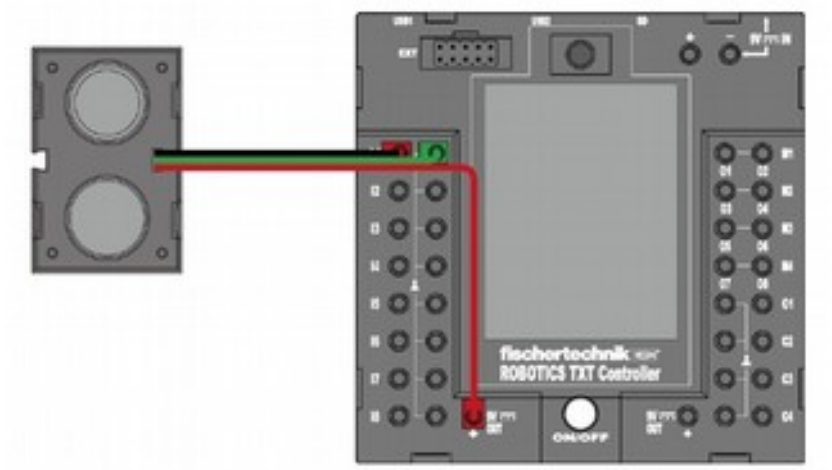
```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to configure the TXT to read data
#          from a potentiometer.
# Date:    20-09-2018
# Author:  M. Vogelaar
#
# Remarks: The script shows a function to convert the status of the TXT
#          from numbers to text. It also demonstrates how to set an input
#          to read a potentiometer. This is a device which has variable
#          resistance depending on the rotation of the stick.
#          The program stops when the resistance is bigger than 5000 Ohm.
#-------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable

i3 = 2                                             # Note this is input I3

M,I = txt.getConfig()
I[i3] = (txt.C_RESISTOR, txt.C_ANALOG)
txt.setConfig(M, I)
txt.updateConfig()

ohm = 0
while 1:
   txt.updateWait()
   newohm = txt.getCurrentInput(i3)
   if newohm != ohm:
      print("Current resistance is", newohm, "Ohm", end="\r")
      ohm = newohm
   if newohm > 5000:
      break
```

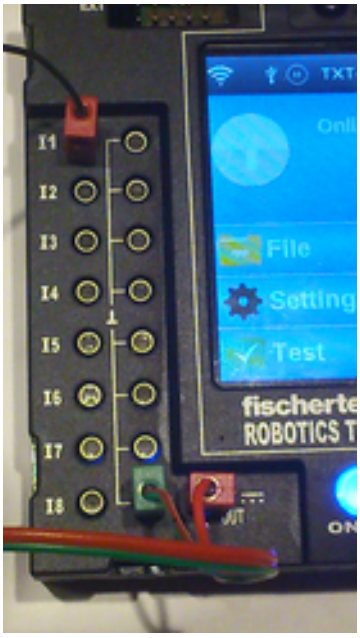
5.8 Ultrasonic Distance Sensor



This sensor uses sound to measure the distance between an object and the sensor. The echo of the ultrasonic sound takes
a certain period in time to be received and the time between transmission and reception is a distance measure.
The software returns a value in centimeters.

The detector has 3 cables. The power of +9V is connected through the red cable. The green cable is connected to ground and the black
cable connects to one of the interface inputs **`I1`** to **`I8`**. The input type must be set to 'ultrasonic'





Connection distance sensor to TXT

Using the inputs on the controller can require an extra setting to be able to process the input of, for example, a sensor. With *ftrobopy* we have to configure the inputs. First we get the configuration with `getConfig()`. This gets this current configuration for the output (in variable **`M`**) and the input (in variable `I`). We don't change the output so we can re-use `M` in our script. The input that we use in out example is the first (**`I1`**). This is the first element if the array **`I`** so we address it with `I[0]` and `I[0]` sets the configuration for the input. We want to use the ultrasonic sensor and

therefore we need to set the input to an analog input. Module *ftrobopy* defines names for these settings. After updating the configuration with method `updateConfig`(), the interface is ready for use with the sensor. We use method `getCurrentInput`() with index 0 to read the input.

***ultrasonic.py*: Read data from the ultrasonic sensor**

```
#-------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the value of the ultrasonic
#          sensor.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: I an endless loop we read the values of the ultrasonic sensor
#          and print its measured distance.
#          The program aborts the loop if the distance between object and
#          sensor is less than 5 centimeters.
#          The input must be configured with device C_ULTRASONIC and type
#          C_ANALOG. These values are given as a tuple as the first element
#          of list I.
#-------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable

M, I = txt.getConfig()
I[0] = (txt.C_ULTRASONIC, txt.C_ANALOG)
txt.setConfig(M, I)
txt.updateConfig()
txt.updateWait()
stopcondition = False
while not stopcondition:
   txt.updateWait()
   d = txt.getCurrentInput(0)
   s = "Measured distance: {:5d}".format(d)
   print(s, end='\r')
   stopcondition = (d < 5)
```

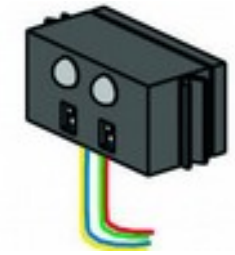
Module *ftrobopy* defines some convenience methods. One of these methods is called `distance`() and the method is defined for objects from class *ultrasonic*. If we instantiate the object, we need to give the number of the input as given on the interface, which in our example is 1.
There is no need to configure the input on the TXT because this is done by the ultrasonic object initialization.
Method distance returns integers. The values are calibrated to output distances in centimeters. The resolution of the sensor is 0.5 cm so it makes sense to have an integer output.

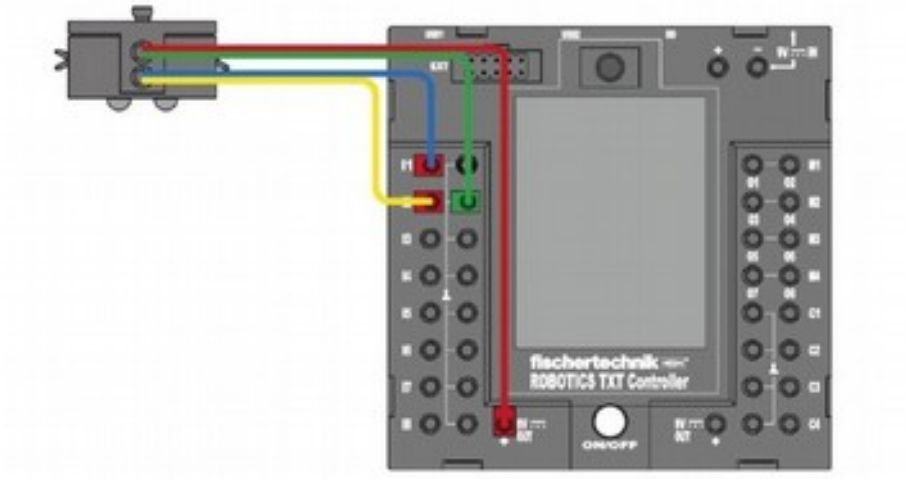
***ultrasonicOO.py*: Reading data from the ultrasonic sensor II (object oriented)**

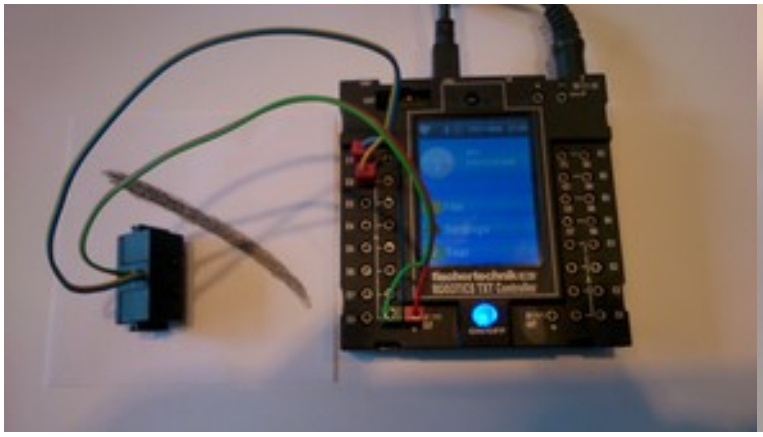
```
#-------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the value of the ultrasonic
#          sensor in an object oriented (OO) way.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: I an endless loop we read the values of the ultrasonic sensor
#          and print its measured distance.
#          The program aborts the loop if the distance between object and
#          sensor is less than 5 centimeters.
#          Method distance() returns a value in cm.
#          Note that the second argument of ultrasonic() is wait=True
#          (set by default). Then the routine takes care of updateWait()
#          itself. The updateWait() in the loop is to prevent unnecessary
#          readings shorter than the update time of the interface.
#-------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable

ultra = txt.ultrasonic(1)   # Input 1 has identifier 1
stopcondition = False

while not stopcondition:
   txt.updateWait()
   d = ultra.distance()
   s = "Measured distance: {:5f}".format(d)
   print(s, end='\r')
   stopcondition = (d < 5)
```

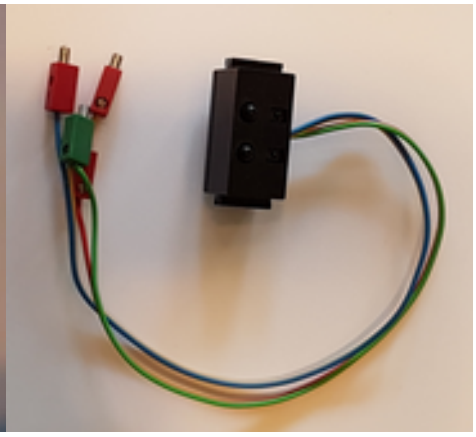
5.9 Infrared Trail Sensor

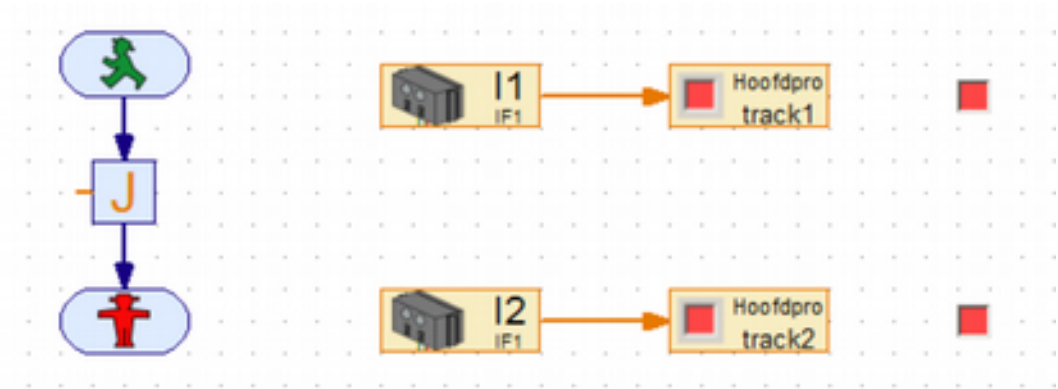
The infrared trail device contains two binary sensors . We have to configure two universal inputs to digital inputs. The mode is set to 10V. The red wire is for the power supply of the sensor. The green is connected to ground. The blue and yellow wires are connected to the universal inputs. The sensor sets the input if it encounters a black trail. Therefore, a track should be smaller than the distance between the two receptors (or as wide to trigger both receptors at the same time). A deviation of the track can be compensated by a change in direction.











***trailsensor.py*: Trail sensor binary data**

```
#------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the values (0 or 1) of the trail
#          sensor in an object oriented (OO) way.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: I an endless loop we read the values of the trail sensor.
#          We need two inputs for this and therefore we also create two
#          trailfollower() objects. A trailfollower object has a method
#          state which returns 0 or 1.
#          Abort with ctrl-c
#------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB
cable

trail1 = txt.trailfollower(1)   # Input 1 has identifier 1
trail2 = txt.trailfollower(2)

while 1:
   txt.updateWait()
   state1 = trail1.state()
   state2 = trail2.state()
   s = "Trail state: 1:{:d} 2:{:d} ".format(state1, state2)
   print(s, end='\r')
```

***trailsensorVoltage.py*: Trail sensor voltage data**

```
#------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the voltages of the trail
#          sensor in an object oriented (OO) way.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: I an endless loop we read the values of the trail sensor.
#          Instead of reading the binary state, we read voltages in mV.
#          The script shows that between black and white, the step in voltage
#          is high and therefore there is not much to gain reading the
#          voltages instead of the binary values.
#------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB
cable

trail1 = txt.voltage(1)   # Input 1 has identifier 1
trail2 = txt.voltage(2)

while 1:
   txt.updateWait()
   state1 = trail1.voltage()
   state2 = trail2.voltage()
   s = "Trail state: 1:{:f} 2:{:f} ".format(state1, state2)
   print(s, end='\r')
```

The last script demonstrates that we can use function voltage to read the voltages produced by the trail sensors but because the transition between black and white is significant, the sensor reduces to a sensor with binary output.

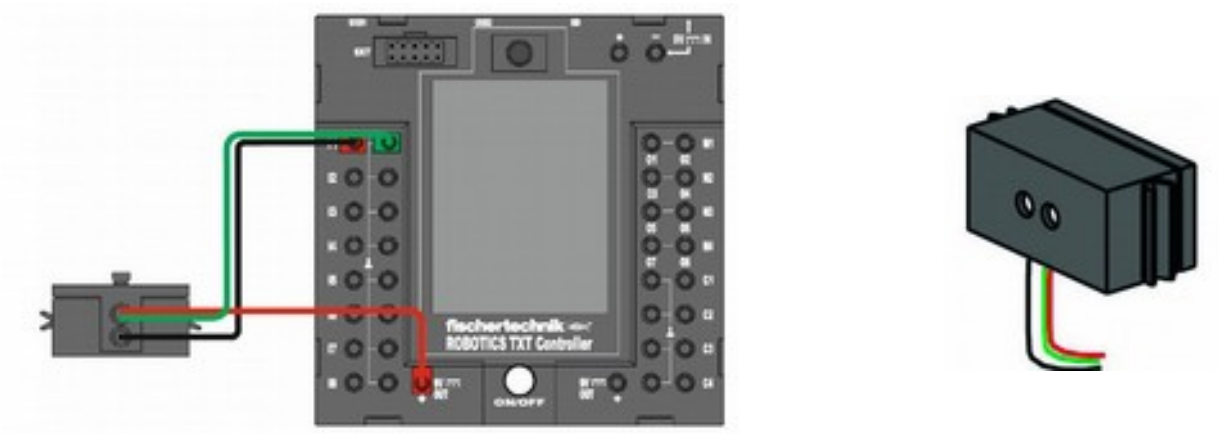
Now we want to take action when a certain transition occurs. The next example uses some Boolean arithmetic to react to transitions. Note that we inverted the state so that on a trail implies `True` (1).

trailTransitions.py*: *Acting on trail transitions

```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to use the trail sensor output and
#          the transitions to direct (hypothetical) motors.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: In fact, this script is an exercise in Boolean arithmetic.
#          We invert the status of the sensors so that we can use
#          variable 'black' which is True if we are on trail.
#-------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB cable

trail1 = txt.trailfollower(1)   # Input 1 has identifier 1
trail2 = txt.trailfollower(2)

black1 = not trail1.state()
black2 = not trail2.state()

if not (black1 and black2):
   print("Set sensor on track", end='\r')
   while not (black1 and black2):
      txt.updateWait()
      black1 = not trail1.state()
      black2 = not trail2.state()

while 1:
    txt.updateWait()
    black1 = not trail1.state()
    black2 = not trail2.state()
    s = "Trail state: 1:{:d} 2:{:d} ".format(black1, black2)
    print(s, end='\r')
    if not black1 and not black2:
        print("Set sensor on track", end='\r')
        while not (black1 and black2):
            txt.updateWait()
            black1 = not trail1.state()
            black2 = not trail2.state()
    if not black1:
        while not black1:
            black1 = not trail1.state()
            txt.updateWait()
            print("Turn motor right            ", end='\r')
    if not black2:
        while not black2:
            black2 = not trail2.state()
            txt.updateWait()
            print("Turn motor left            ", end='\r')
```

5.10 Color Sensor

The color sensor send light out by a red LED. Different colors reflect a different amount of light and therefore in different sensor values.



Module *ftrobopy* has a class `colorsensor` which provides two methods. Method `value()` return a sensor value as an integer. Method `color()` returns the measured color as a string. The sensor should have a distance to a colored object between 5 and 10 mm. Note that values can also be read with the `voltage()` function.

Connection color sensor to TXT

In the example script below, we can observe that the resolution of the values should be enough to distinguish more colors than the three names that *ftrobopy* returns. In practice, one will use the `values()` method only.

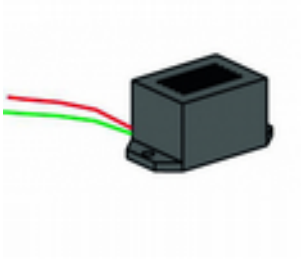
***colorsensor.py*: Reading values from the color sensor**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the values of the color sensor
#          sensor in an object oriented (OO) way.
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: I an endless loop we read the values of the color sensor.
#          Abort with ctrl-c
#--------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB
cable

col = txt.colorsensor(1)

while 1:
   txt.updateWait()
   colval = col.value()
   colname = col.color()
   s = "Color: {:5d} {:>6s} ".format(colval, colname)
   print(s, end='\r')
```

6 Outputs

6.1 Lamp, buzzer

The output sockets `M1`..`M4` can be used for motors and for lamps. But a lamp or buzzer needs to be connected to only one socket from **`M1`**..**`M4`** and the other connection is to ground (⊥). In fact this gives 8 output connections lamps. These outputs are called single pole outputs and are labeled **`O1`** to **`O8`** and work in only one direction (which is trivial for devices like lamps and buzzers).



***lamps.py*: Set brightness of a lamp**

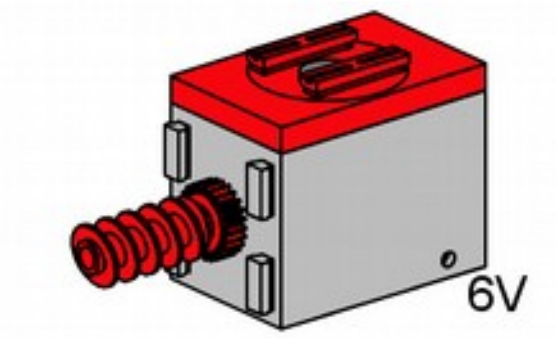
```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to use outputs O1..O8 for lamps
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: Create an output object and use method setLevel() to set the
#          brightness of a lamp.
#-------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable
lamp = txt.output(6)           # Lamp between output 7 and ground
lamp.setLevel(450)             # Set the brightness between 0-512
txt.updateWait(2)              # Let it on for two seconds
txt.stopAll()                  # Stop all output
txt.updateWait()               # Wait one interface cycle
print("Finished")              # Finish program
```

Module *ftrobopy* provides a convenience routine for output devices like a lamp or a buzzer. It is method `setLevel`() of the out class created by function `output`(). The lamp is on for a certain amount of time set by the argument in `updateWait`(). After that time, we set all outputs to zero with function `stopAll`() as an alternative for `setLevel`(0). Then we wait a short while so that the interface can process this last command and finish the program.

The buzzer can be quite loud !
The setting `setLevel`(32) seems a reasonable level.

*Buzzer connected to **O8***

6.2 Classic motor



Fischertechnik motors can be connected to one of the motor outputs **M1** to **M4**. You need to connect to both sockets of one motor output. With function `motor()` we create a motor object and the speed of the motor is set with its method `setSpeed()`.

***classicmotor.py*: Run classic motor on motor output M1..M4**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to use the motor outputs
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: The motor speed is set with method setSpeed()
#--------------------------------------------------------------------------
from time import sleep
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000) # Connect interface by USB cable

M1 = txt.motor(1)
M1.setSpeed(32)
sleep(1)
M1.stop()
```

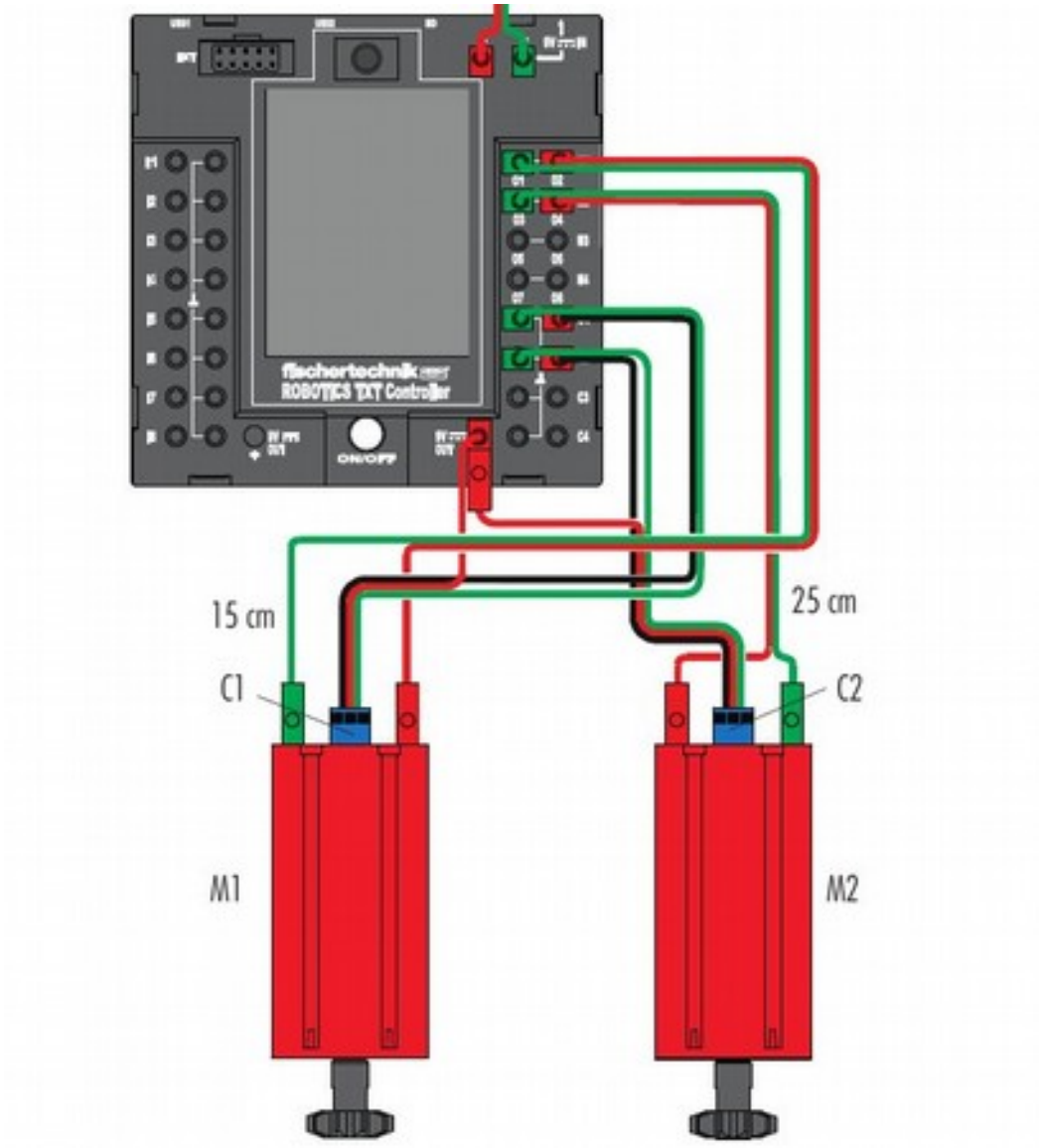
Note that the old FT motors run at 6 Volt, so you should limit your speed level for this type of motors to a modest level.

6.3 Encoder motors

The motor has 5 connections. Connect the power connection to **M1** (or **M2**, **M3**, **M4**).
In the interface test program, the motor acts like a standard FT motor if not connected to a counter.

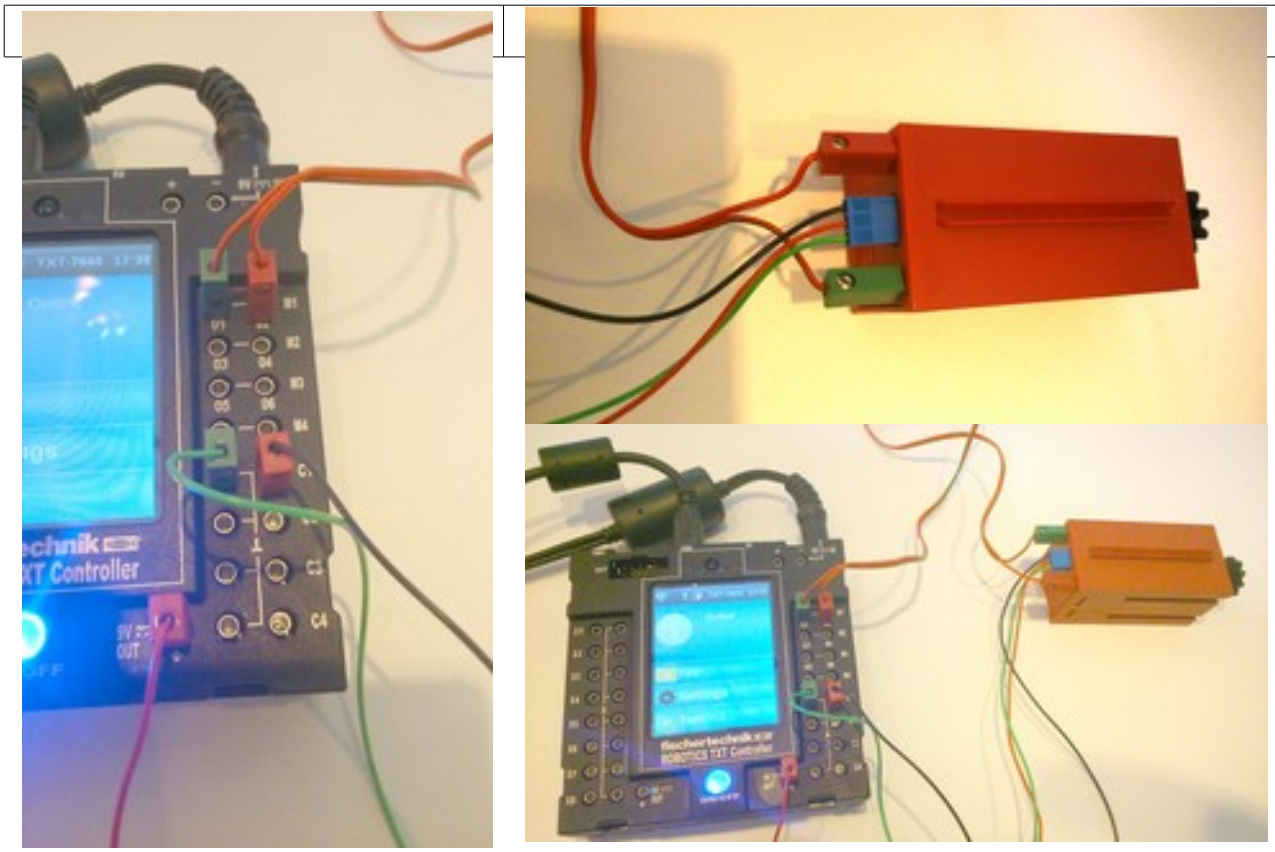
6.3.1 Counter connector

To connect a counter you need the 3 wires cable with the blue connector. This is the power supply for the counter in the motor.
The black cable is to connect to a counter input on the interface. The red cable must be connected to a +9V output on the interface. The green cable is connected to any input on the interface which shows a ground symbol.



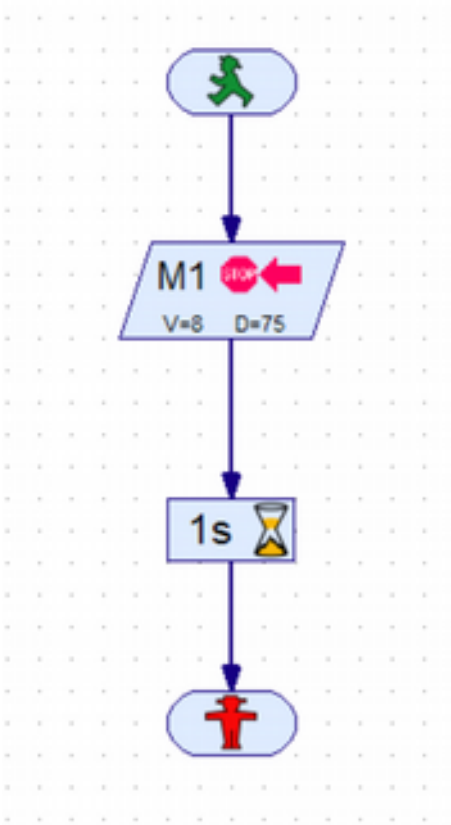
Note that schemes for connections of motors and sensors like the one above, can be found at the official Fischertechnik site (in German these are called Zusatzblatt):

https://www.fischertechnik.de/-/media/fischertechnik/fite/service/downloads/robotics/txt-controller/documents/zusatzblatt-txt-schaltplaene-fuer-txt-training-lab,-d-,pdf.ashx



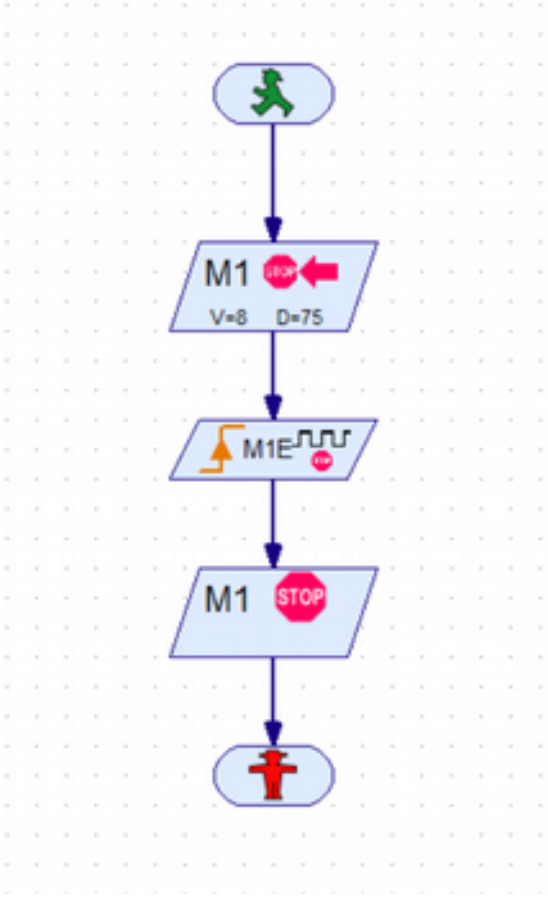
With the counter attached to the counter input on the interface, the interface test program shows that if the motor runs, also the counter in the test program runs.
The encoder motors also have a gearbox with a transmission ratio of 25:1 then one revolution of the motor axis corresponds to 75 pulses of the encoder.

First let's discuss a ROBOPro example and then a similar example with *ftrobopy*.



Solution A

- Encoder motor (level 1) and set it to M1, Distance and set the distance to 75
- Sleep. Set to 1 sec.



Solution B

- Encoder motor (level 1) and set it to M1, Distance and set the distance to 75
- Waiting for input (level 1) and set it to input ME1
- Encoder motor. Set to M1 and Stop

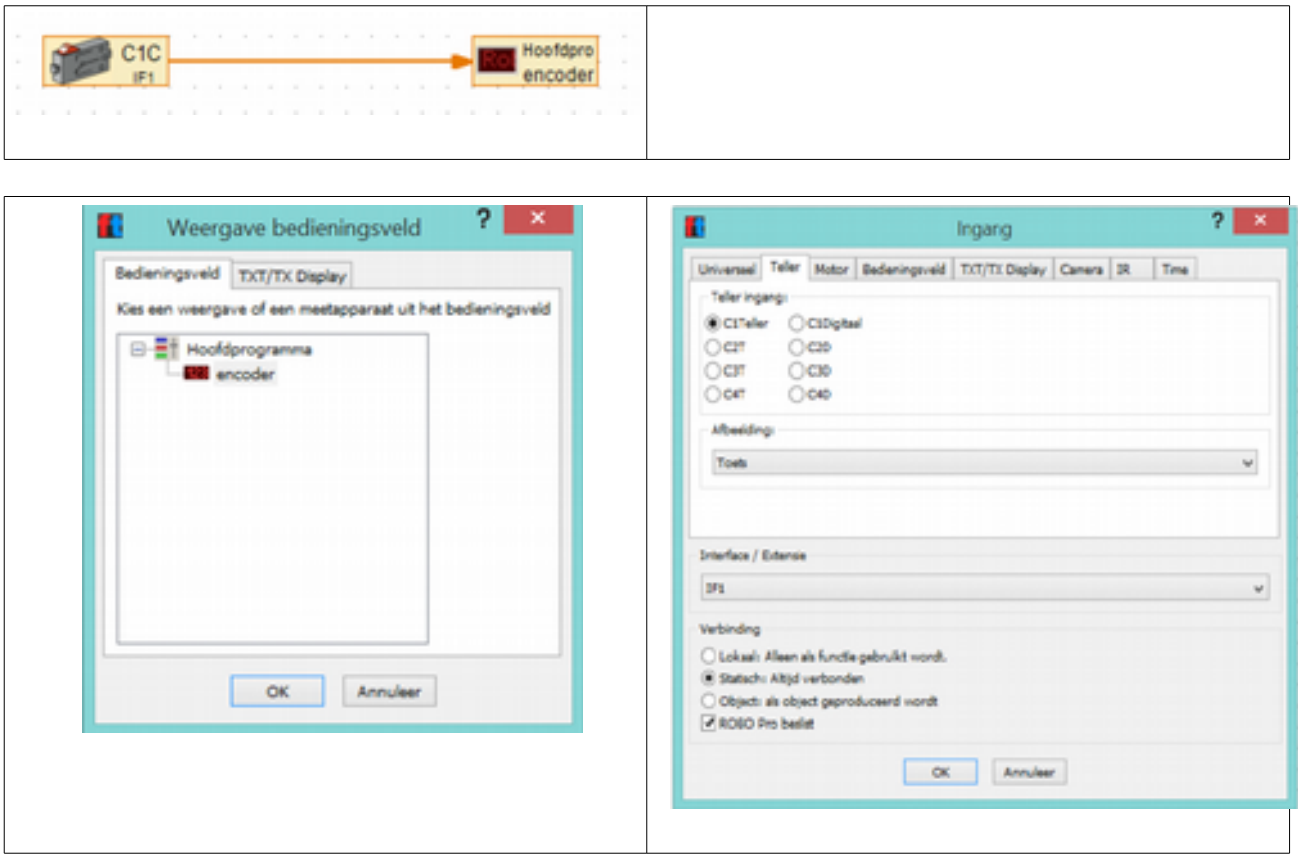
The encoder motor element is set to 75 steps (remember that you need to right click the element to set its properties). If one naively would use only the encoder motor element only, then nothing happens. The reason is that the interface connection is closed before the motor could have started or finish its revolution. So in solution A, we give the interface some time to finish. The sleep element is set to 1 sec which seems enough to finish 75 steps.

But if we vary our number of steps, we should also vary the time in the sleep element. That's asking for a different solution where we only stop the program after finishing the required number of steps. The element “waiting for input” has a special option to wait for a motor to finish. For motor 1 this input is ME1. So it knows which counter to read and when the counter is equal to the required

number of steps, it triggers an input action. This can only work if the black cable for the counter is connected to the counter with the same number as the motor output on the interface.

We added an encoder motor element with a Stop action in case we want to use standard motor commands after executing the required number of steps. It releases the encoder motor. The step is not necessary if you stay in encoder mode.

6.3.2 Reading the encoder motor counter



If you need to control the progress of the encoder motor, you can display the number of steps that is counted by counter **C1**. So from category inputs/outputs, we select 'universal input' and change its configuration to show the content of **C1** which, as stated before, is associated with the motor connected to **M1**. As a reader we configured the reader element to connect to the variable output element in the configuration window. In the function field, one needs to make a connection between reader and output element.

6.3.3 Looking for the equivalent for Linux

The Python example below shows how to connect to the controller by USB cable. It also shows how to use objects. For instance, class `motor` is used to instantiate a motor object and several methods can be used to set properties (`setSpeed`(), `setDistance`()).
In the ROBOPro example we included elements which show the number of steps that the motor turned. In Python we achieve the same result (in a terminal with a simple print function) by inspecting the current value of counter **`C1`** and printing this value until the motor finished.

***encodermotor.py*: Encoder motor with counter**

```
#-------------------------------------------------------------------------
# Purpose: Script to demonstrate how to run the encoder motor for a given
#          number of steps
# Date:    09-09-2018
# Author:  M. Vogelaar
#
# Remarks: Create a motor object. Set speed and distance. Read the distance
#          using either method getCurrentCounterValue() for counter 1 or
#          getCurrentDistance() which is a method of the motor class. The last
#          option results in cleaner code.
#          In the loop, we continue to print values until the motor has
#          reached the required distance. To avoid too many cycles (i.e.
#          cpu use), we use updateWait() which waits until the next data
#          exchange with the TXT
#
#-------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable

mot1 = 0
M1 = txt.motor(1)       # Create a motor object
M1.setSpeed(512)        # Set its speed
M1.setDistance(75)      # Set the number of steps

while not M1.finished():
   n = txt.getCurrentCounterValue(mot1)    # Read the counter M1 -> index 0
   cd = M1.getCurrentDistance()
   print("Current distance motor 1=", cd, n)
   txt.updateWait()
```

6.3.4 Synchronize motors

Assume you have a second motor M2 and want to synchronize it with M1. Then tell each motor object to synchronize with the other:

```
M1.setDistance(75, syncto=M2)
M2.setDistance(75, syncto=M1)
```

Note: If you want to synchronize **TXT commands**, use function `SyncDataBegin`() and `SyncDataEnd`()

7 Miscellaneous devices

7.1 Camera

7.1.1 Creating an image on disk

Images from the USB camera can be retrieved in online mode. Method `startCameraOnline`() starts a process on he TXT to send the data trough port 65001 and at the client side it starts a Python thread which retrieves the frames in *jpeg* format. Starting the process on the TXT takes around 2 seconds. So we pause with the `sleep`() function after initializing the camera. If you installed package *PIL*, you can display the result immediately with function `Image.open`().

***camera.py*: Getting an image from the camera and show it**

```
#-------------------------------------------------------------------------
# Purpose: Script to get a camera image from the USB camera connected
#          to the TXT (Python 3)
# Date:    06-09-2018
# Author:  M. Vogelaar
#
# Remarks: This script should read and display a camera frame.
#-------------------------------------------------------------------------
import ftrobopy
from PIL import Image
import time

#txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
#txt = ftrobopy.ftrobopy('192.168.8.2', 65000)      # Connected by WLAN
txt = ftrobopy.ftrobopy('192.168.1.242', 65000)     # Connected by WLAN client
#txt = ftrobopy.ftrobopy('192.168.9.2', 65000)      # Connected by Bluetooth

print('Name of controller: ', txt.getDevicename())
print('Version:', hex(txt.getVersionNumber()))

txt.startCameraOnline()
time.sleep(2.5)
im = "TXTimage.jpg"
pic = txt.getCameraFrame()

with open(im,'wb') as f:
   f.write(bytearray(pic))

txt.stopCameraOnline()
image = Image.open(im)
image.show()
```

Besides package *PIL*, there is also package *opencv*. Anaconda users can install it with:

```
conda install opencv
```

It is an image and video processing library for facial recognition and detection, license plate reading, photo editing, advanced robotic vision, optical character recognition etc.
Start importing *opencv* in Python with: `import cv2`

7.1.2 Image processing with opencv

We compiled a small example to filter out the green component from the image we created with the camera. It demonstrates that the data is accessible through NumPy arrays which are the standard arrays for numerical calculations in Python. We can address any element in an array. In the example we replaced all values corresponding to green by zero. Note that the data in a jpeg is arranged with the three colors first, then the order is the x axis data and then y axis data. In Python we address this in the way C/C++ does, so the order becomes rows, columns, color in the index of array `im`.

```
import cv2
import numpy as np

print("Press a key in the image window to abort script. Sometimes it takes a
while to abort.")

# read image into NumPy array
im =  cv2.imread("telephone.jpeg")
print("Prove that cv2.imread gives a NumPy array. Type:", type(im),
np.shape(im))

# height, width and number of planes
height, width, bpp = np.shape(im)

# Set second plane (green) to image value 0
# Note that the first axis x contains three values for each pixel
# The second axis y is the horizontal axis of the image
# The third axis z is the vertical axis of the image
# The order can be confusing wrt to 3D data where the order is z,y,x
colorplane = 1
for hor in range(0, width):
    for ver in range(0, height):
        im[ver,hor,colorplane] = 0

# Show the result
cv2.imshow('matrix', im)
cv2.waitKey(0)
```

Numerical processing in Python is based on array from class ndarray as defined in Package NumPy. An element can be addressed with an index. The first index is 0. For a matrix `M` with an x, y and z axis, the order of addressing elements is M[z,y,x] similar to the programming language C. The order seems counter intuitive and you need some exercise to get used to it.

7.2 Infrared controller

The TXT can receive signals from the joystick controllers. We didn't test the Bluetooth version. The infrared version works fine (provided that the battery works well).
In a previous version of *ftrobopy*, one could read the status of both joysticks and buttons with one function called `getCurrentIr()`. This function is now considered obsolete. One should use the more Pythonic functions `joystick()`, `joybutton()` and `joydipswitch()`.

Infrared remote control with joysticks

The script combines the status of the left joystick with the setting of the speed of an encoder motor.
The joystick is selected with the first parameter in function `joystick()`. The parameter is called joynum and the left joystick is set with 0. We use method `updown()` to find the amplitude of the stick which is a floating point number between -1 and 1. We want to use this value to set the speed of the motor which is a number between -512 and 512 by multiplying the joystick value with 512. The method `setSpeed()` accepts only integers as parameter so we have to convert our floating point value to an integer with function `int()`.

- Note that in the script we get the joystick values in a loop but only change the speed if the new value for the calculated speed differs from the previous one.
- Note also that in the loop be inserted `updateWait()` which calls the `sleep()` function to prevent unnecessary CPU cycles.
- Note that the 'old' remote IR control (picture below) **is not** compatible with TXT.

Old remote control not usable with TXT

***joystickIRmotor.py*: Read values from infrared remote control (version with joysticks)**

```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the value of the infrared remote
#          control on the txt controller. Note that this is the controller
#          with the two joysticks.
# Date:    10-09-2018
# Author:  M. Vogelaar
#
# Remarks: Start an encoder motor with a limited distance and speed 0
#          Use the left joystick and read the up and down state with
#          method updown(). The returned numbers are floating point numbers
#          between -1.0 and 1.0. Convert these numbers to integers (required
#          type in setSpeed()) with values between -512 to 512 to control
#          the speed of the motor. Update the speed only when its value
#          changed.
#-------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable

dist = 200              # Arbitrary distance
M1 = txt.motor(1)       # Create a motor object
M1.setSpeed(0)          # Set its speed
M1.setDistance(dist)    # Set the number of steps

# Linker joystick=0, all remote controls=0, infrared=0
joystick = txt.joystick(0,0,0)

speed = 0
while 1:
    newspeed = int(joystick.updown()*512)
    if newspeed != speed:
       speed = newspeed
       M1.setSpeed(speed)
       # Not necessary to update with incrMotorCmdId()
       newdist = M1.getCurrentDistance()
       if newdist != dist:
           print("Current distance motor 1=", newdist)
           dist = newdist
    txt.updateWait()
```

The controller can also be read with the lower level function `getCurrentIr()`. In the code below we implemented a function `statusIR()` which converts a status returned with `getCurrentIR()` to a human readable status. This is a useful script to check your remote control.

joystickIRstatus.py*: *Print status of infrared remote control in text

```
#------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to read the value of infrared remote
#          control on the txt controller. Note that this is the controller
#          with the two joysticks.
# Date:    14-09-2018
# Author:  M. Vogelaar
#
# Remarks: This script uses the function getCurrentIR() to obtain all
#          status values of the remote control at once.
#          End script with ctrl-c
#------------------------------------------------------------------------------
import ftrobopy

def statusIR(inp):
   #-------------------------------------------------
   # How to read the sequence for the infrared data?
   # For the default dip switch setting, only indices
   # 0 to 5 are relevant. Other indices become relevant
   # for other settings of the dip-switches.
   # 0 -> LEFT stick: left-right -15 to 15
   # 1 -> LEFT stick: up-down -15 to 15
   # 2 -> RIGHT stick: left-right -15 to 15
   # 3 -> RIGHT stick: up-down -15 to 15
   # 4 -> On button: 1, Off button 2
   #-------------------------------------------------
   if inp[0]:
      if inp[0] > 0: print("LEFT stick Right", inp[0])
      else: print("LEFT stick Left", inp[0])
   if inp[1]:
      if inp[1] > 0: print("LEFT stick UP", inp[1])
      else: print("LEFT stick Down", inp[1])
   if inp[2]:
      if inp[2] > 0: print("RIGHT stick Right", inp[2])
      else: print("RIGHT stick Left", inp[2])
   if inp[3]:
      if inp[3] > 0: print("RIGHT stick UP", inp[3])
      else: print("RIGHT stick Down", inp[3])
   if inp[4]:
      if inp[4] == 1: print("Button ON")
      if inp[4] == 2: print("Button OFF")

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB
cable

# Start reading the infrared data from the controller
while 1:
   txt.updateWait()
   IR = txt.getCurrentIr()
   statusIR(IR)
```

7.3 I2C devices

We did not test any I2C devices. There is a good description at:
https://reivilofischertechnik.weebly.com/how-to-connect-an-isup2c-device-to-the-robotics-txt.html

Currently, there is no support for I2C in *ftrobopy* (version 1.80, sep 2018), see
https://github.com/ftrobopy/ftrobopy/issues/8
but there is an intention to do so in the future.

8 Controlling loops with a keyboard key

8.1 `input()` function

In the previous example and in various others, we implemented an infinite loop. Usually this is is a loop that starts with

```
while 1:
```
or
```
while True:
```

The standard way to abort this loop is by implementing some condition or by pressing ctrl-c on the keyboard to abort the program.
This doesn't give you the control you need for user interaction. Of course you can use Python's function `input()` to prompt a user to enter a string but this blocks the loop until at least the Enter key has been pressed. So we have to find an alternative approach.

8.2 Class nbc

A small Python class seems the best solution for catching keyboard key strokes in a non-blocking way. The class can be copied from the code below and stored in a file with name *nbc.py*. The code was found on Stackoverflow.

***nbc.py*: Class nbc.py which defines a non blocking console**

```
import sys
import select
import tty
import termios

class NonBlockingConsole(object):

    def __enter__(self):
        self.old_settings = termios.tcgetattr(sys.stdin)
        tty.setcbreak(sys.stdin.fileno())
        return self

    def __exit__(self, type, value, traceback):
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, self.old_settings)

    def get_data(self):
        if select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], []):
            return sys.stdin.read(1)
        return False
```

Note that the 'magic' methods `__enter__` *and* `__exit__` *allows you to implement objects which can be used easily with Python statement* `with`.

The next script will show this, It has been tested with Python 3 in Linux.

***readKeyboardNBC.py*: Reading the LDR in a non blocking console**

```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate how to configure the input for a
#          light-dependent resistor (LDR) and to read its values.
# Date:    14-09-2018
# Author:  M. Vogelaar
#
# Remarks: In this script we configure input I1 as a resistance meter
#          with a range of 15kOhm. An LDR is connected to input 1.
#          Function getConfig returns the current configuration for
#          the motors and inputs in a tuple. The second element is an
#          array with inputs. Input 1 has index 0.
#          For an LDR (or NTC) we use the configuration tuple
#          txt.C_RESISTOR, txt.C_ANALOG
#          Print a resistance value only when it has been changed.
#          The script stops when you cover the LDR (the resistace will
#          then be equal or bigger than 15kOhm
#
#          Class nbc defines a non blocking console. The loop is aborted
#          with the escape key.
#-------------------------------------------------------------------------------
import ftrobopy
from nbc import NonBlockingConsole

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable

i1 = 2
M, I = txt.getConfig()
I[i1] = (txt.C_RESISTOR, txt.C_ANALOG)
txt.setConfig(M, I)
txt.updateConfig()

ohm = 0

with NonBlockingConsole() as nbc:
   while 1:
      txt.updateWait(0.2)
      newohm = txt.getCurrentInput(i1)
      if newohm != ohm:
         print("Current resistance is {:5d} Ohm".format(newohm), end='\r')
         ohm = newohm
      if newohm == 15000:
         txt.stopOnline()
         break
      if nbc.get_data() == '\x1b':  # x1b is ESC
         txt.stopOnline()
         break
```

8.3 curses

A second non blocking method to catch key strokes uses module *curses*. This module is a screen-painting and keyboard-handling facility, see for instance:

https://docs.python.org/3.3/howto/curses.html

Curses programming is easy to learn and is a fair alternative to Graphical User Interfaces like QT. In the next example we use *curses* in a non blocking mode:

***LDRreaderOOcurses.py*: Reading LDR data with curses screen- and keyboard handling**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to create an resistor object
#          for a light-dependent resistor (LDR) and to read its values.
# Date:    14-09-2018
# Author:  M. Vogelaar
# Remarks: In this script we use the resistor class to read the values
#          of an LDR.
#          Print a resistance value only when it has been changed.
#          The script stops when you cover the LDR (the resistace will
#          then be equal or bigger than 15kOhm
#
#          The script uses module curses to handle screen and keyboard
#          strokes.
#--------------------------------------------------------------------------
import curses
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
R = txt.resistor(1)      # Here input 1 has id 1

stdscr = curses.initscr()
curses.cbreak()
stdscr.keypad(1)
stdscr.addstr(0, 0,"Hit 'q' or 'ESC' to quit")
stdscr.refresh()
stdscr.nodelay(True)     # Keyboard in non blocking mode
curses.noecho()

ohm = -1                 # Initialize to non-existing value
while 1:                 # Start endless loop
   txt.updateWait(0.2)
   newohm = R.value()
   if newohm != ohm:
      s = "Current resistance is {:d} Ohm     \r".format(newohm)
      stdscr.addstr(2, 0, s)
      stdscr.refresh()
      ohm = newohm
   if newohm == 15000:
      break
   key = stdscr.getch()
   if key == 27 or key == ord('q'):   # keyboard ESC or 'q'
      break

txt.stopOnline()  # Disconnect from TXT
curses.endwin()   # Restore terminal
```

9 Multi threading

9.1 Blinking light in a thread

For ROBOPro in Windows, parallel processing with threads is very well explained in
http://reivilofischertechnik.weebly.com/parallel-processing.html

For Linux we have to find solutions ourselves. Luckily it is easy to apply multi threading in a script. The core module is called *threading*. From that module we use function `Thread()` to assign a function to a thread.
Usually you want to make the thread a daemon thread. With daemon threads, we start the thread and can forget about it. When the program quits, all daemon threads are killed automatically. The next script is a program that blinks a light in a thread while the main program deals with user input.

***blinkingInThread.py*: Blinking light in thread**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how implement a blinking light in a thread.
# Date:    17-09-2018
# Author:  M. Vogelaar
#
# Remarks: Setup a blinking light in a function used in a thread.
#--------------------------------------------------------------------------
import threading
import ftrobopy
from time import sleep

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)  # Connect interface by USB cable

def blink(lamp):
   # Function used in thread to blink a lamp
   while 1:
      lamp.setLevel(256)
      sleep(0.5)
      lamp.setLevel(0)
      sleep(0.5)


lamp = txt.output(6)          # Lamp between output 6 and ground
t1 = threading.Thread(target=blink, args=(lamp,))
# Thread is a 'daemon' thread so that it stops when the program ends.
t1.daemon = True
t1.start()

# Dummy part, just to demonstrate that the thread works
while 1:
   s = input("Enter dummy: ")
   print("Watch blinking light")
```

9.2 Reading counters in a thread

If we create a function for reading a counter, we can use multi threading to read for instance two counters with the same function while leaving overall control to the main program. Below is the most simple example where we again use module `threading`.

***readCountersInThread.py*: Simple threading example reading two counters**

```
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate a very simple threaded application
# Date:    06-09-2018
# Author:  M. Vogelaar
#
# Remarks: This script defines a function which reads a counter and displays
#          a value when it is changed (e.g. connect c1 with ground).
#          The function is used in a thread.
#          The main program waits for user input to abort the program.
#          At the same time, counter values are displayed for c1 and c2
#-------------------------------------------------------------------------------
import ftrobopy
import threading

#txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
#txt = ftrobopy.ftrobopy('192.168.8.2', 65000)      # Connected by WLAN
txt = ftrobopy.ftrobopy('192.168.1.242', 65000)     # Connected by WLAN client
#txt = ftrobopy.ftrobopy('192.168.9.2', 65000)      # Connected by Bluetooth

print('Name of controller: ', txt.getDevicename())
print('Version:', hex(txt.getVersionNumber()))

def counter(c):
    while(1):
       if txt.getCurrentCounterInput(c):
          print(txt.getCurrentCounterValue(c))

# Main
t1 = threading.Thread(target=counter, args=(0,))
t2 = threading.Thread(target=counter, args=(1,))
# Thread is a daemon thread so that it stops when the program ends.
t1.daemon = True
t2.daemon = True
t1.start()
t2.start()
s = input("Press enter to stop")
```

9.3 Threading example encoder motor with digital input

Now we discussed both the encoder motor and a digital input, we can combine these into a script using threading. In the thread we check whether input I1 is closed. If so, the function in the thread stops the motor. The function in the thread `emergency()` is designed to accept two objects as input parameter. Note that one can extend this example very easily by adding more objects in the tuple in function `Thread()`.

***motorEmergencyThread.py*: Emergency stop with threading**

```
#-------------------------------------------------------------------------
# Purpose: Script which runs an encoder motor with an emergency routine
#          in a thread
# Date:    09-09-2018
# Author:  M. Vogelaar
#
# Remarks: An encoder motor can be interrupted in various ways.
#          In this script we use a thread based on a function which
#          has input parameters 'inp' and 'mot' which are input- and
#          motor objects. If the pushbutton switch on the input 1 is pressed,
#          it stops the motor.
#-------------------------------------------------------------------------
import ftrobopy
import threading

txt = ftrobopy.ftrobopy('192.168.7.2', 65000) # Connect interface by USB cable

def emergency(inp, mot):
    #-------------------------------------------------------------------------
    # Routine which checks digital input.If switch pressed, then stop motor
    #-------------------------------------------------------------------------
    while 1:
       if inp.state() == 1:
          mot.stop()
       txt.updateWait()

M1 = txt.motor(1)       # Create a motor object
M1.setSpeed(256)        # Set its speed
M1.setDistance(500)     # Set the number of steps
I1 = txt.input(1)       # Create input object
txt.updateWait()        # Wait a bit before starting the thread

t1 = threading.Thread(target=emergency, args=(I1, M1))
# Thread is a daemon thread so that it stops when the program ends.
t1.daemon = True
t1.start()

# While tho motor runs, print its distance
while not M1.finished():
   cd = M1.getCurrentDistance()
   print("Current distance motor 1=", cd)
   txt.updateWait()
```

9.4 Threading example in a Jupyter notebook

A very convenient way to develop Python programs is a Jupyter notebook which runs in a browser.
At http://jupyter.org/ we find the following description of a notebook:

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Copy the code below in a Jupyter Notebook cell and press shift-enter to execute it.

***ftrobopyExamples.ipynb*: Blinking lights in a Jupyter notebook cell**

```
import threading
import ftrobopy
from time import sleep

txt = ftrobopy.ftrobopy('192.168.7.2', 65000) # Connect by USB cable

stop = False
def blink(lamp, interval=0.5):
    # Function used in thread to blink a lamp
    global stop
    while not stop:
        lamp.setLevel(256)
        sleep(interval)
        lamp.setLevel(0)
        sleep(interval)

lamp = txt.output(6)          # Lamp between output 6 and ground
t1 = threading.Thread(target=blink, args=(lamp,0.2))
t1.start()
`
# Dummy part, just to demonstrate that the thread works
while 1:
    s = input("Enter dummy: ")
    print("Watch blinking light")
    if s != '':
        stop = True
        break

txt.stopOnline()
sleep(0.5)
print("Does thread still run?", t1.isAlive())
```

Running the blinking lights script in a Jupyter notebook requires some modifications to the script with respect to the original script.

If you execute this cell, the execution of the code is stopped when a non empty input at the “Enter dummy” prompt was given, but you will observe that the TXT interface is still online. So the first difference between script and notebook is that we need a function to stop the TXT explicitly.

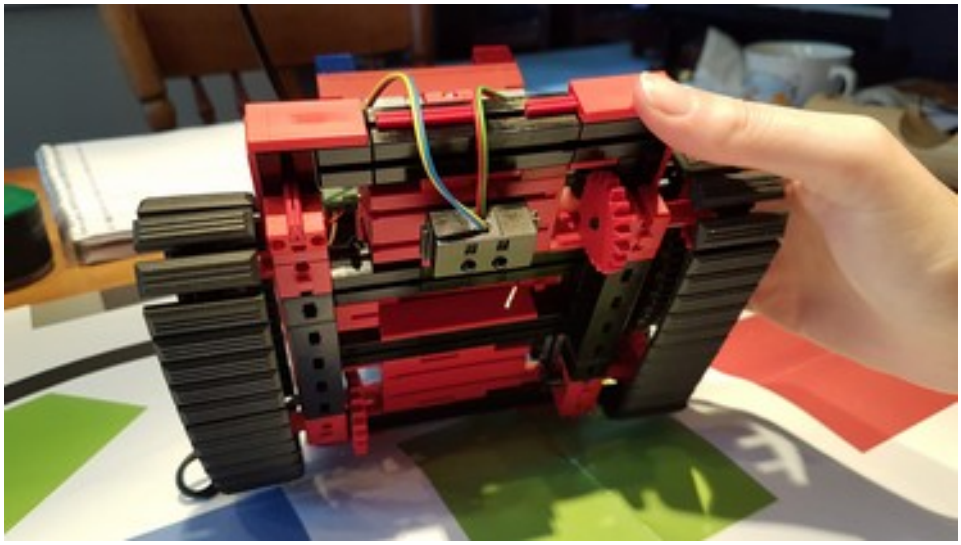
This function is `stopOnline()`.

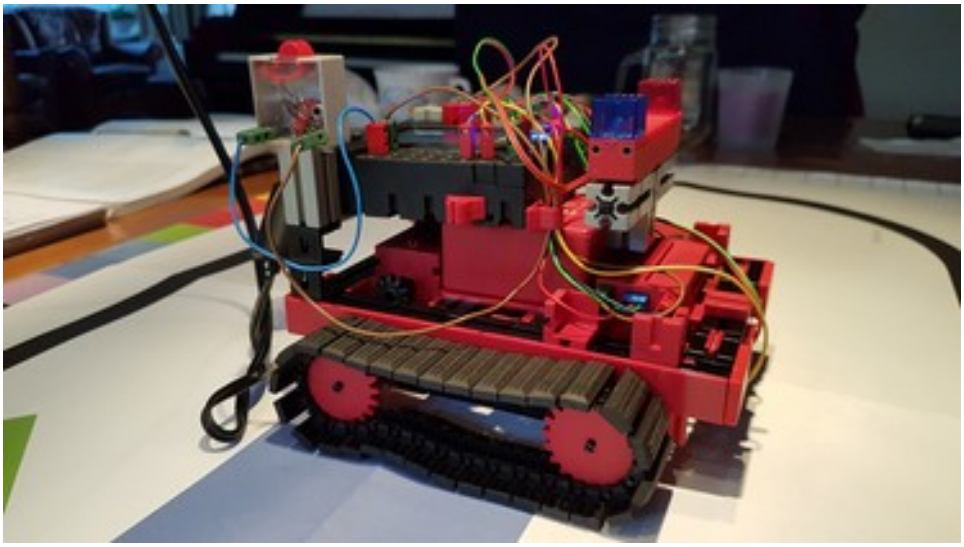
But there is a second problem. When the cell has been executed and we end the loop with a non empty input, the thread continues to run. Even when it has been set as a daemon, there is no program end when the code in the cell is executed, so the thread is not stopped and will continue to run in the notebook. It can then only be killed by restarting the notebook's kernel.

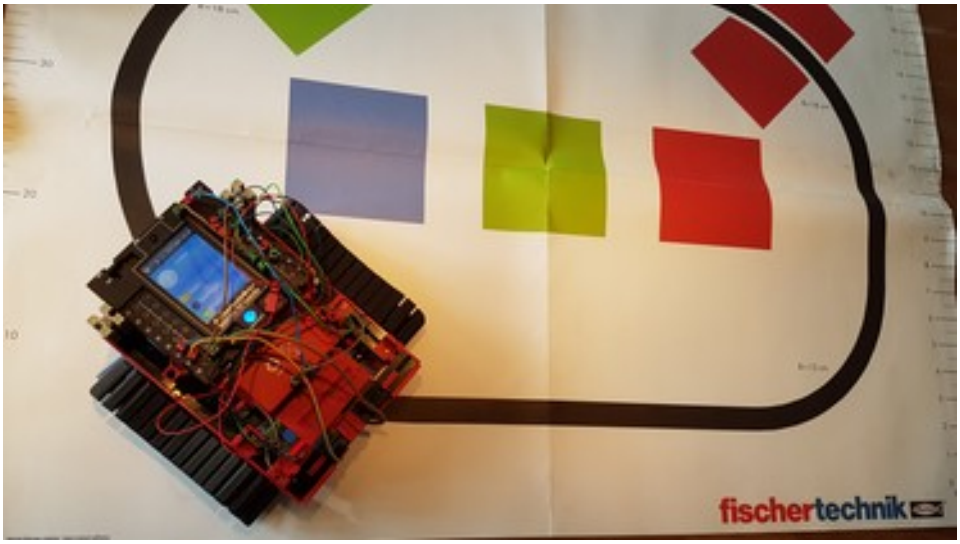
One way to stop the thread is to let it read a global variable. This variable is set in the main while loop. When we finish the code in the cell, the variable `stop` will be set and read by the thread function which will then finish. We give it some time to finish with function `sleep`. Whether the thread really ends, can be checked with method `isAlive()`, which we used in the last line in the script.

10 Trail follower

In the next example we combine the trail sensor, the encoder motors, a pushbutton switch for an emergency stop and the technique of multi threading, to make a trail following robot.







trailrobot.py: Trail following robot

```
#--------------------------------------------------------------------------
# Purpose: Script to setup a trail follower robot
# Date:    16-09-2018
# Author:  M. Vogelaar
#
# Remarks: After start, the script asks to set the robot on the trail
#          (black track on FT sheet). The program is stopped after aborting
#          the script with ctrl-c or by pressing the switch on the robot.
#
#          Script requests first to set the robot on the trail. Then the
#          blinking lamps warn that the robot is following the trail and
#          the motors start running while correcting for curves. When it
#          loses the trail completely (both sensors do not see the track)
#          an alarm starts for a short while. If the switch on the robot
#          is pressed, it is detected in the emergency thread and the
#          program will be aborted.
#--------------------------------------------------------------------------
import threading
import ftrobopy

#txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connect interface by USB cable
txt = ftrobopy.ftrobopy('192.168.1.242', 65000)      # Connected by WLAN client

trail1 = txt.trailfollower(6) # Trail sensor input 6,7
trail2 = txt.trailfollower(7)

black1 = not trail1.state()   # Read the status of the trail sensors
black2 = not trail2.state()

Motor_rechts = txt.motor(2)   # Setup 2 encoder motors
Motor_links  = txt.motor(1)

lamp = txt.output(7)          # Lamp on output 7 and ground
I1 = txt.input(1)             # Emergency switch on input 1

stop =  False

def finish(M1, M2, I1, lamp):
    # Function for in a thread
    # If the emergency switch has been pressed,
    # all devices are stopped and Boolean 'stop'
    # is set to True to signal the main program
    # that it can finish
    global stop
    while 1:
        if I1.state():  # If pressed then True
            M1.stop()
            M2.stop()
            lamp.setLevel(0)
            txt.stop_sound()
            stop = True
            break       # Abort this while loop
        txt.updateWait()


t1 = threading.Thread(target=finish, args=(Motor_rechts, Motor_links, I1, lamp))
# Thread is a 'daemon' thread so that it stops when the program ends.
t1.daemon = True
t1.start()
```

Continued

```
# Loop until robot is set on track
if not (black1 and black2):
   s = "{:20s}".format("Set sensor on track")
   print(s, end='\r')
   while not (black1 and black2):
      txt.updateWait()
      black1 = not trail1.state()
      black2 = not trail2.state()

lamp.setLevel(450)   # Lamp on when on track


# Start the main loop. This controls the motors depending
# on the state of the trail sensor.
while not stop:
    txt.updateWait()
    black1 = not trail1.state()
    black2 = not trail2.state()
    if black1 and black2:
        s = "{:20s}".format("On trail")
        print(s, end='\r')
        Motor_rechts.setSpeed(450)
        Motor_links.setSpeed(450)
    else:
        Motor_rechts.stop()
        Motor_links.stop()
    if not black1 and not black2:
        s = "{:20s}".format("Set sensor on track")
        print(s, end='\r')
        Motor_rechts.stop()
        Motor_links.stop()
        txt.play_sound(2, repeat=1)
        while not (black1 and black2) and not stop:
            #txt.updateWait()
            black1 = not trail1.state()
            black2 = not trail2.state()
        txt.stop_sound()
    if not black1:
        while not black1 and not stop:
            black1 = not trail1.state()
            Motor_rechts.setSpeed(500)
            Motor_links.setSpeed(0)
            #txt.updateWait()
            s = "{:20s}".format("Turn motor left")
            print(s, end='\r')
    if not black2:
        while not black2 and not stop:
            black2 = not trail2.state()
            Motor_rechts.setSpeed(0)
            Motor_links.setSpeed(500)
            #txt.updateWait()
            s = "{:20s}".format("Turn motor right")
            print(s, end='\r')
```

11 Creating graphical interfaces with PyQT5

It is not difficult to create graphical interfaces with Python and PyQT5. If you have a template you can vary on the code to build your own Graphical User Interface (GUI).

***motorSliderQT5.py*: Graphical user interface with PyQT5**

```
#--------------------------------------------------------------------------
# Purpose: Script which uses a PyQt5 slider to control a motor output
# Date:    15-09-2018
# Author:  M. Vogelaar
#
# Remarks: Use a simple layout for basic QT widgets. The most important one
#          is the slider. If the slider changes, its value is used to adjust
#          the speed of motor M1.
#--------------------------------------------------------------------------

import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import (QApplication, QGridLayout, QPushButton,
                             QWidget, QSlider, QLabel)
import ftrobopy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000) # Connect interface by USB cable

class Window(QWidget):
    def __init__(self, parent=None):
        super(Window, self).__init__(parent)

        self.buildLayout()
        self.setWindowTitle("FT slider demo")
        self.resize(400, 100)
        self.M1 = txt.motor(1)          # FT related: Create motor object
        self.M1.setSpeed(0)

    def buildLayout(self):
        slider = QSlider(Qt.Horizontal)
        slider.setFocusPolicy(Qt.StrongFocus)
        slider.setTickPosition(QSlider.TicksBothSides)
        slider.setTickInterval(10)
        slider.setSingleStep(1)
        slider.setMinimum(0)
        slider.setMaximum(512)
        slider.valueChanged.connect(self.valuechange)
        self.slider = slider
        self.levellabel = QLabel()
        self.levellabel.setText("{:4d}".format(0))
        self.quit = QPushButton("Quit")
        self.quit.clicked.connect(self.closeEvent)

        grid = QGridLayout()
        grid.addWidget(self.slider, 0, 1)
        grid.addWidget(self.levellabel, 0, 0)
        grid.addWidget(self.quit, 1, 0, 1, 2) # column span is 2
        self.setLayout(grid)
```

```
    def valuechange(self):
        # Get the (changed) value from the slider and adjust speed
        level = self.slider.value()
        self.M1.setSpeed(level)
        txt.updateWait()
        self.levellabel.setText("{:4d}".format(level))

    def closeEvent(self, event):
        self.M1.stop()
        txt.updateWait()
        txt.stopOnline()    # Abort connection to TXT
        self.close()
        QApplication.quit()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec_())
```

A PyQt5 application runs an event loop until an event is generated to close the application.. The loop is started with `exec_()` which is a method of class `QApplication`. Method `quit()` aborts the loop. In the example we generate a close event through method `closeEvent()` which is connected to a Quit button. When a user drags the slider, the method `valuechange()` will be called which updates the label with the new value of the slider and sets the speed of the motor with the same value. To make this possible we did set a range on the slider equal to the range in motor speed, i.e. between 0 and 512.

If you want to learn about PyQT then find a free PDF version of the book
Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming by Mark Summerfield.

The book documents QT version 4. Current distributions use QT5. The Python binding is called PyQt5 and uses a different approach to connecting functions to events. But the differences are minor and the book is still very useful.

12 Plotting real time sensor data with Matplotlib

If you want to monitor data acquisition from a sensor in real time, you need a setup where the plot rebuilds when new data has been read. This can be done with plot package *Matplotlib* in a simple way. This package provides an 'animation' module to create animations.

***realTimeDataPlot.py*: Plot real time observed data from a sensor with Matplotlib**

```
#--------------------------------------------------------------------------
# Purpose: Script to demonstrate how to plot real time data values as
#          function of time with Matplotlib
# Date:    20-09-2018
# Author:  M. Vogelaar
#
# Remarks: The script collects data from a sensor (e.g. LDR) at I3.
#          It plots the last 50 points
#--------------------------------------------------------------------------
import datetime as dt
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import matplotlib.dates as mdates
import ftrobopy

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ts = []          # List to store observation time
data = []        # List to store sensor data

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
R = txt.resistor(3)

def readsensor(counter, ts, data):
    # Read input 3 on TXT
    sensorvalue = R.value()

    # Add sensor data to list
    ts.append(dt.datetime.now())
    data.append(sensorvalue)

    # Store no more than 25 data points
    ts = ts[-25:]
    data = data[-25:]

    # Plot data points and connect with lines
    ax.clear()
    ax.xaxis.set_major_formatter(mdates.DateFormatter("%H.%M:%S"))
    ax.plot(ts, data)
    ax.scatter(ts, data, color='r')

    # Adjust plotpreperties
    plt.xticks(rotation=45, ha='right')
    plt.ylim(0, 16000)                  # Set limit to accommodate LDR range
    plt.subplots_adjust(bottom=0.30)
    plt.title('Time')
    plt.ylabel('Resistance (Ohm)')
    plt.grid()

animation = animation.FuncAnimation(fig, readsensor,
                                    fargs=(ts, data), interval=1000)
plt.show()
```

Each step in the animation is a new plot of the last 25 measurements. An LDR is probed every second and the measured resistance is stored in a list. The time of the measurement is also stored in a list. The time is recorded with function `datetime.now()` and formatted to give hours, minutes and integer seconds only.

To limit the amount of stored and plotted data points, we take the last 25 data points from the list and use the Python trick with a negative index number to slice it.

The time labels along the x-axis take a lot of space. We use function `subplots_adjust()` to create this space at the bottom of the plot. In function `xticks()` we set the rotation and the horizontal alignment of the x labels.

There are many examples with *Matplotlib* routines for interactive plotting. The approach with function `FuncAnimation()` is currently the most robust and works both for Python 2 and Python 3.

Package *Matplotlib* is part of the Anaconda distribution.

13 Python on the FT community firmware

13.1 Execute ftrobopy scripts on the TXT (direct mode)

To execute Python scripts on your TXT controller you need to install the Fischertechnik TXT community software. We followed the installation steps on:

http://cfw.ftcommunity.de/ftcommunity-TXT/en/getting-started/installation.html

and copied the three necessary files to a 32 GB microSD card. You can do this on any PC which can read/write an SD card. The next step is to tell the system that it can boot from an SD card. The documentation wants you to use a program like Putty, but it is easier to use SSH (which is also available on a Windows 10 installation). You need to know the WLAN IP address assigned to the TXT for example:

```
$ ssh ROBOPro@192.168.1.242
```

On the command line type:

```
$ sudo /usr/sbin/boot_sd_nand
```

to make the TXT bootable from a SD card. If the card is removed, you will boot the TXT in the standard environment so in principle nothing can be damaged. The first time you boot from an SD it takes a while before the TXT is ready. Subsequent logins are faster.

Besides the navigation on the touchscreen of the TXT, one can enter the interface in a browser. The connection is a HTTP connection. A web address (URL) can be for example http://192.168.1.242/ The web interface has an upload facility for the scripts that you want to run on the TXT. In the tutorial https://cfw.ftcommunity.de/ftcommunity-TXT/en/programming/python/tutorial-1.html you will find a section “Upload it to the TXT” which describes how to use this facility.

For our Python scripts on a Linux platform, it can be much easier to upload to the TXT. First we need to know which user names can be used to login to the TXT. We find the necessary information in http://cfw.ftcommunity.de/ftcommunity-TXT/en/advanced/password-policy.html

User ftc can login without a password. In the example below we use either the USB connection or connect over the WLAN.

`$ ssh ftc@192.168.7.2` (USB)

`$ ssh ftc@192.168.1.242` (WLAN with assigned IP address)

Note that immediately on the TXT console, a prompt appears to allow or cancel access. Keep your eyes on the TXT.

This user `ftc` has a home directory `/home/ftc/` where we made a new folder called scripts. Then from a Linux console we type

```
$ scp soundtestDirect.py ftc@192.168.1.242:/home/ftc/scripts/
```

Note that for all access to the TXT, a dialog will appear on the TXT for confirmation. The community software provides a Python 3 environment with *ftrobopy*. The easiest way to start your script on the TXT is to tell it to use localhost for a connection and start it for example with:

```
$ python soundtestDirect.py
```

Besides user `ftc` you can also become `root` on the TXT. Both user accounts are described on the community web page: http://cfw.ftcommunity.de/ftcommunity-TXT/en/advanced/password-policy.html

The TXT runs a Linux system so we can apply some standard commands to make it an executable file and run it from the command line.. First we add a so called *SheBang* line which tells the system which interpreter must be used to execute the script. This is a standard line:(see example):

```
$ cd scripts
$ nano soundtestDirect.py
```

Now add the line:

```
#!/usr/bin/env python
```

as the first line of your script. Then:

```
$ chmod u+x soundtestDirect.py
$ ./soundtestDirect.py
```

***soundtestDirect.py*: Run sound check script in 'direct' mode**

```
#!/usr/bin/env python
#-------------------------------------------------------------------------------
# Purpose: Script to demonstrate ftrobopy in direct mode
# Date:    06-09-2018
# Author:  M. Vogelaar
#
# Remarks: This script should play 3 sounds available to the TXT.
#          Note that a next sound can only be played when the first one
#          is finished. Therefore we check in a loop if the sound
#          was ended. Setting a volume works only when the connection
#          is in 'direct mode'.
#          Volume control works in direct mode (volume= argument)
#-------------------------------------------------------------------------------
import ftrobopy

txt = ftrobopy.ftrobopy('127.0.0.1', 65000)         # Localhost, direct mode
#txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # Connected by USB cable
#txt = ftrobopy.ftrobopy('192.168.8.2', 65000)      # Connected by WLAN
#txt = ftrobopy.ftrobopy('192.168.1.242', 65000)    # Connected by WLAN client
#txt = ftrobopy.ftrobopy('192.168.9.2', 65000)      # Connected by Bluetooth

print('Name of controller: ', txt.getDevicename())
print('Firmware version:', txt.getFirmwareVersion())

for i in range(1,3):
      txt.play_sound(i, repeat=1, volume=(i-1)*30+10))
      while not txt.sound_finished():
          pass
```

If you want to use the community software with ROBOPro, on a Windows PC, you need to escape to the old TXT console with button FT-GUI. This is also true for *ftrobopy*. You can return to the community environment by pressing the on/off button on the TXT for a short moment.
This can fail if you messed around with the setting in the old FT environment. Then press for a longer while until the TXT starts to reboot.

Scripts written with *ftrobypy* cannot connect from a PC while in the community environment. Here also, one has to start the old environment by pressing icon FT-GUI on the TXT. At the moment of writing, the issue is known to the developers. Since firmware version 0.9.3, the TXT has a **two-factor authentication**. To connect to the TXT by any network you have to accept the connection on the display of your TXT. This works for `ssh` and `scp` but there is no dialog when a *ftrobopy* script is started on your PC.

The community software adds much for the Python developer on Linux but is perhaps of limited use for Windows users.

More information about how to use the community software can be found at:
http://cfw.ftcommunity.de/ftcommunity-TXT/en/getting-started/usage.html

13.2 Start Python scripts on a standalone TXT

The upload facility on the web interface described on https://cfw.ftcommunity.de/ftcommunity-TXT/en/programming/python/tutorial-1.html in section "Upload it to the TXT" is not necessary to install apps.

These are the steps:

On the TXT, make a directory under `/home/ftc/apps` as in:

```
$ cd /dev/ftc/apps
$ mkdir Soundtest
```

Copy your application to that folder with secure copy:

```
$ scp sountest.py ftc@192.168.1.242:/home/ftc/apps/Soundtest/
```

Find a nice 64x64 pixels icon copy it and give it the required name icon.png

```
$ scp music-icon.png ftc@192.168.1.242:/home/ftc/apps/Soundtest/icon.png
```

Make a text file called manifest according to the rules in http://cfw.ftcommunity.de/ftcommunity-TXT/en/programming/python/tutorial-1.html

A simple manifest file for this application is for example:

```
[app]
name: Sound test
category: Tests
author: M. Vogelaar
icon: icon.png
desc: Sound and volume check
exec: soundtest.py
managed: yes
uuid: someuniqueidentifier1234
version: 1.0
firmware: 0.9.4
```

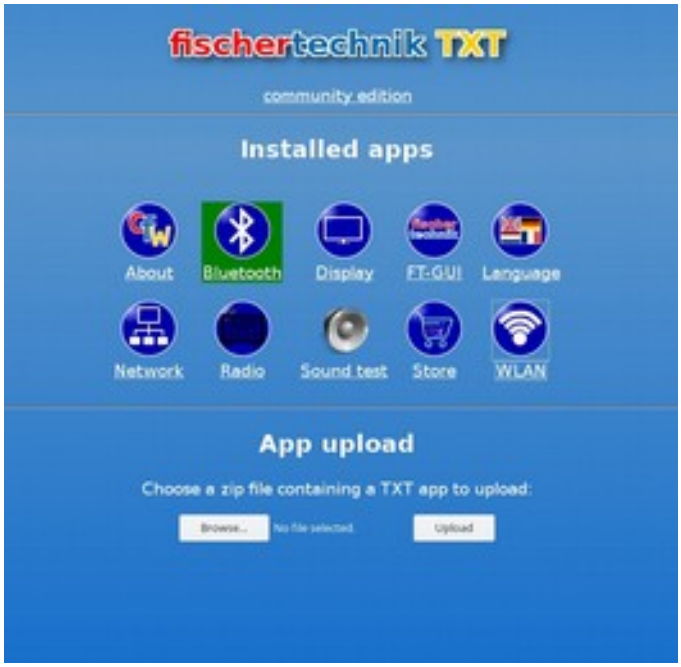
Copy this manifest with `scp` to directory `Soundtest` on the TXT and reboot the TXT. The app will appear on the TXT display under folder DEMOS and can be started by pressing the corresponding icon..

For small changes in the program you can use editor *vi* or *nano* on the TXT. Editor *nano* is very easy to use and is the first choice usually for people who do not want to invest in learning *vi*.

13.3 Starting applications from the TXT web page

The web page of the TXT, started with the URL `http://192.168.1.242/` shows also the application you just installed yourself and you can start any of such applications from the web interface. The application runs then on the TXT.

Note that the web interface can only be accessed by the WLAN IP (i.e. not by the IP of the USB connection)



TXT web page on PC showing new app Sound test

Notes about connection options with the community software

- `scp`, `ssh` connections are possible to the TXT if it configured as WLAN client and connected with a USB cable connection.
- We could not find options to configure the TXT as access point
- We could not setup a Bluetooth connection between PC and TXT. The setup fails, probably caused by the two step authentication in the community software. It shows up as network device FT-TXT-CFW in the Bluetooth but no pin number is presented and the one generated from the FT firmware does not work (nor does 0000).
- Going from the new environment to the old with the app FT-GUI on the TXT gives you the same restrictions. You cannot get Bluetooth connections and cannot use the TXT as access point (AP).

13.4 Programming the TXT touch screen on a PC

The FT Community software installs PyQt4 as the package to create Graphical User Interfaces (GUI's) with. At the moment of writing this document, September 2018, the most obvious choice is PyQt5. For a while, the Anaconda distribution included the (almost) compatible alternative called PySide but this generated a lot of compatibility issues and nowadays PyQt5 is installed by default.

When developing your software to run on the TXT with a GUI, you currently have to stick to PyQt4. On the forum https://forum.ftcommunity.de/viewtopic.php?f=33&t=4064&p=30379 there is a discussion about this but no intentions are shown to upgrade to PyQt5.

So how do we get the simple script `test.py` on
http://cfw.ftcommunity.de/ftcommunity-TXT/en/programming/python/tutorial-1.html
to run if you want to play with it on a PC with PyQt5? Here are the steps:

1. `$ git clone https://github.com/ftCommunity/ftcommunity-TXT.git`
2. In the directory where you develop test.py you need the file TouchStyle.py as in:
   ```
   $ cp ftcommunity-
   TXT/board/fischertechnik/TXT/rootfs/opt/ftc/TouchStyle.py .
   ```
3. You need some theme material. First create the directory themes/default and copy all the files from the same folder in the GIT download to themes/default in your develop directory. The Github page is: https://github.com/ftCommunity/ftcommunity-TXT/tree/master/board/fischertechnik/TXT/rootfs/opt/ftc/themes/default

   ```
   $ cp ftcommunity-
   TXT/board/fischertechnik/TXT/rootfs/opt/ftc/themes/default/*
   themes/default/
   ```
4. Apply some changes to `TouchStyle.py` to get things to work at least partially. We show the difference with the original file with `diff` in Linux:

```
10,29c10,11
<
< # Can we run this program with either pyqt4 or pyqt5?
< pyqt5 = True
< try:
<    from PyQt5.QtCore import pyqtSignal, Qt, QThread, QObject
<    from PyQt5.QtGui import *
<    from PyQt5.QtWidgets import *
< except:
<    try:
<       from PyQt4.QtCore import *
<       from PyQt4.QtGui import *
<       pyqt5 = False
<    except:
<       raise
<
< if pyqt5:
<    print("Using PyQt5")
< else:
<    print("Using PyQt4")
<
---
> from PyQt4.QtCore import *
> from PyQt4.QtGui import *
749,751d730
< if pyqt5:
<     QInputContext = QObject
<
824,825c803
<             if not pyqt5:
<                self.setInputContext(TouchInputContext(self)) # PyQt4 only
---
>             self.setInputContext(TouchInputContext(self))
```

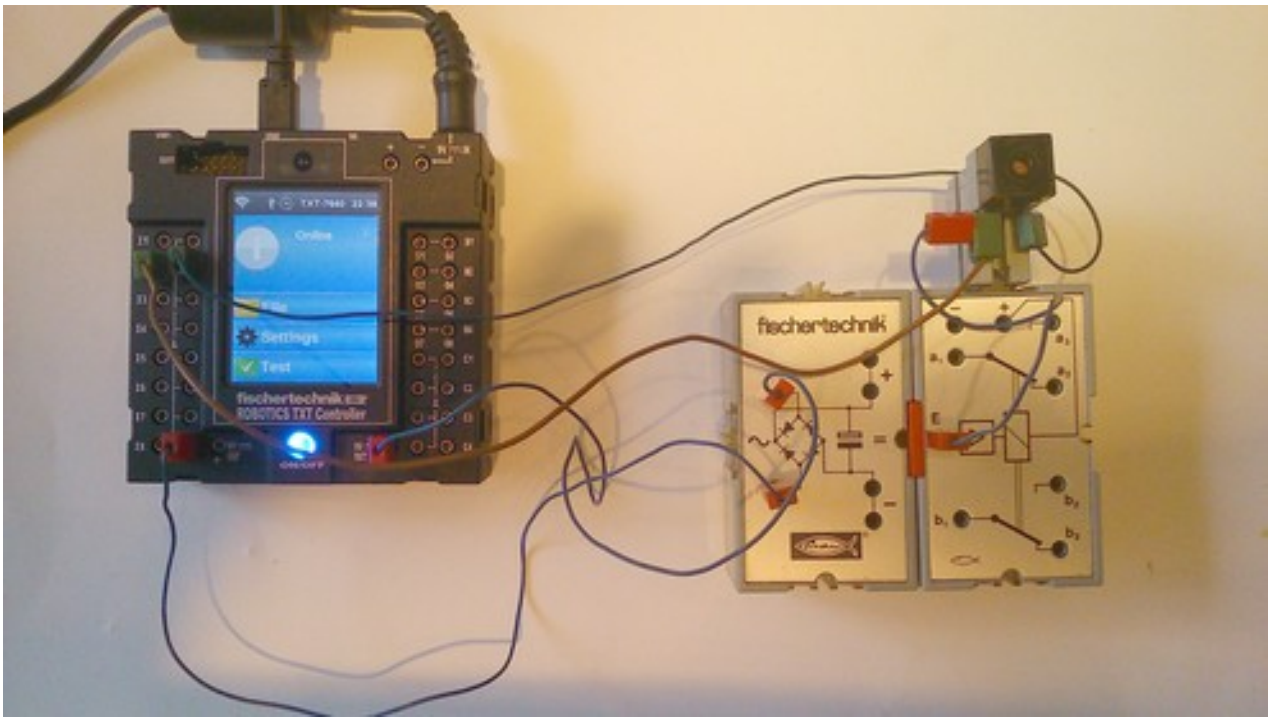
With the above differences applied to TouchStyle.py, we get a module which acts exactly as the unchanged module in PyQt4 but also works in PyQt5, except for the virtual keyboard.

This situation is not ideal but it gives us an option to develop a script in our Anaconda environment on a PC using PyQt5 and with little or no modifications to migrate the script to run under PyQt4 on the TXT itself.

14 Appendix

14.1 Connections with the 'Silberlingen'

It is possible to combine the old electronic blocks ('Silberlingen') with the TXT controller. Both operate with 9V power supply. You have to use the same power supply for both systems. That can be done by connecting the rectifier block input with the 9V output of the TXT controller and the ground on the controller. There is a small drop in voltage due to the capacitor in the rectifier. This drop can be avoided by connecting the rectifier + output directly with the +9V of the



TXT connected to 'old' electronic blocks

TXT and the – output of the rectifier with the ground on the TXT.

The relay will switch if it is connected to the ground. For the circuit in the picture above, we used a photo cell (LDR) on the TXT to generate a signal for the relay. If the light is sparse, the input **I1** is set to 0. If it increases, a threshold is exceeded and the input is set to ground and therefore also the relay is set to ground, which sets the relay to on. This setup can be tested with the interface test in the ROBOPro software or with a simple Python script.

Perhaps more practical is the circuit in the next picture. Here we use the amplifier function of the base electronic block to switch the TXT controller. We copy an old connection scheme from the book Hobby 4 Part 1.
The picture below shows that we connected the power from the TXT to the + and – sockets of the rectifier.
Instead of connecting **A1** to **E1** of the relay, we connect **A1** to **I1** of the controller.
This seems to work fine. Above a certain threshold, set by the amplifier, the LDR will switch on the lamp on the amplifier and **A1** is then connected to "-" TXT and the electronic blocks share the same ground so **I1** on the TXT an input is generated as if a pushbutton switch was pressed, **I1** on the TXT must therefore be configured as digital input.

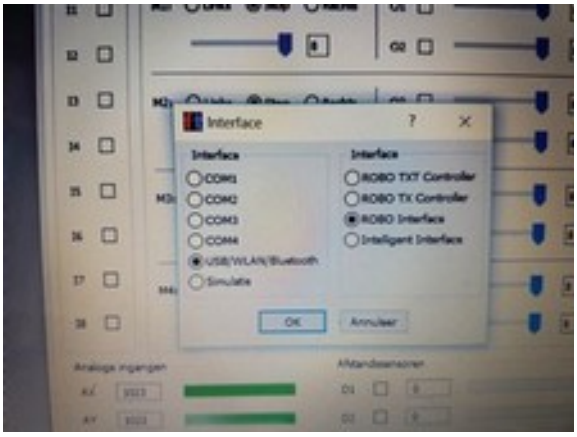
Electronic blocks generate input on input 1 of TXT

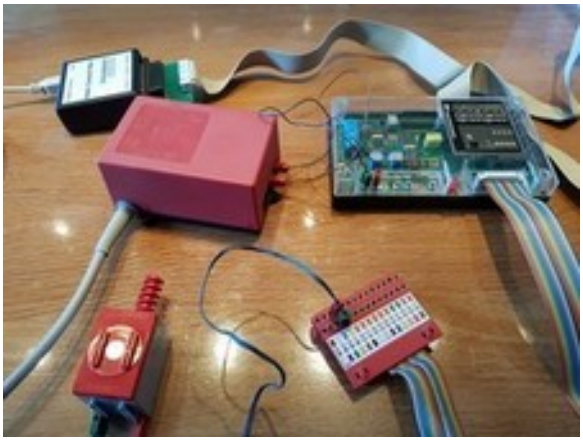
14.2 Robo Connect Box

It's a challenge to connect a Universal parallel Interface (30520) from 1991 to a laptop, using the *Robo Connect Box* from Knobloch electronics: https://knobloch-gmbh.de

We tested two interfaces which were connected by a flat cable (master/slave configuration).
This setup did not work (there were some warnings about this also in the Knobloch documentation).
So we disconnected the interfaces and connected only one of them to the *connect box* as in the pictures.
The box is connected to the PC with a USB cable.

C





To test the interface we connected a motor to M1. The ROBOPro software was updated to the latest version 4.2.4. In the software one needs to select the 'ROBO Interface' and USB as the port.
The power is a 8V FT power supply. We connected + of the power supply with + on the interface (and – with –).

It works well, but we don't see options to combine it with the TXT because it is impossible to mix the required software. In theory it is possible to run both *ftrobopy* for the TXT and ROBOPro for the Universal interface on a Windows 10 system but communication between those will be impossible.

14.3 Using the ROBO TX Controller with module fttxpy

The information in this section can be useful for people who own both a TX and a TXT controller. The community software is not required to use the module described in this section.

In ftpedia-2018-2, Raphael Jacob wrote an article about programming the 2009 TX controller with Python in Linux, see: https://www.ftcommunity.de/ftpedia_ausgaben/ftpedia-2018-2.pdf
The module in the article is called *fttxpy*. It can be cloned from Github with command:

```
$ git clone https://github.com/ski7777/fttxpy.git
```

and it is installed with:

```
$ python setup.py install
```

Currently (September 2018), the module is in a beta phase.
One needs to manually install the modules *pyserial* and *pyudev* with pip:

```
$ pip install pyserial
$ pip install pyudev
```

The software needs access to a file `/dev/ttyACM0` (last digit can differ) but this file is readable and writable for root and a group *dialout* of which a user is not a automatically a member set by default. So you should add the user to group dialout with:

```
$ sudo adduser someusername dialout
```

(where *someusername* is the name of the user who wants to use the fttxpy module) and reboot your PC. If that does not work, (like on an Ubuntu 16.04 system) you can change the default permission of the device.The device is created when you connect the TX to your PC with a USB cable. When the device is visible in `dev`, you can change its permissions with:

```
$ chmod o+rw /dev/ttyACM0
```

as root or with sudo.

After a reboot of your PC, this command needs to be applied again which is not very convenient. To make it more permanent, you can add a *udev* rule as follows:

```
$ cd /etc/udev/rules.d/
$ ls
$ sudo nano 50-myusb.rules
```

or whatever editor you are used to, add the single line:

```
KERNEL=="ttyACM[0-9]*",MODE="0666"
```

and save the file. A program that includes module *fttxpy* should work now for any user. To test the module, create a script with the following lines:

```
from fttxpy import fttxpy
tx = fttxpy()
```

Execute it and if successful you can expect a message in your terminal similar to:

```
Found 1 TX-Controller(s). Automatically selecting the first one!
Connected to ROBO TX-288 Version 1.30
```

TX controller

The module has high-level functions which are the same as in module *ftrobopy*. The following example shows this similarity. The main difference is that we need to create a `robotx`() object first. A convenience function like `updateWait`() in *ftrobopy* is not available in *fttxpy* but can be replaced by `time.sleep`().

Working with *fttxpy* on the PC with the TXT in online mode often loses the connection. Rebooting the TX can help sometimes. If not, you can change the USB cable to another USB socket on the PC. It takes some fiddling to get things to work but if you are patient it will work in the end.

***fttxpyTest.py*: Test program for fttxpy on the 'old' TX controller**

```
#-------------------------------------------------------------------------------
# Purpose: Script to read an input on the TX with module fttxpy
# Date:    21-09-2018
# Author:  M. Vogelaar
#
# Remarks: Module fttxpy is in beta phase
#          fttxpy needs module serial. and udev. This must be installed with
#          pip install pyserial
#          pip install pyudev
#-------------------------------------------------------------------------------
from time import sleep
from fttxpy import fttxpy

tx = fttxpy()
print("Change state of Input 1. Abort with ctrl-c")
master = tx.robotx()
I1 = master.input(1)
state = I1.state()

while 1:
   ns = I1.state()
   if ns != state:
      print("state() =", ns)
      state = ns
   sleep(0.2)
```

14.4 Combining the TX and TXT

In one Python script on Linux, one can control both the TX and TXT !

In the next script we show how to do this. It's a program that waits until the light for an LDR on the TX is bright enough to switch I1 on the TX.. A sound on the sound module will be activated Then an NTC on the TXT starts to print its current temperature. After a certain number of measurements, the TXT will trigger another sound on the sound module and the program will finish. Note that when you combine the controllers with each other or with other devices, the power supply is from one source only. In our example we use the 9V power output on the TXT to power the TX and the sound module. Note that it also possible to use separate power supplies as long there is no contact between the two controllers.
The 'Remarks' lines in the next script describe which I/O is used and what the script does.

TXT, TX and sound module

***TxandTXTsoundmodule.py*: Combine the TX and TXT in one Python program, part 1**

```
#-------------------------------------------------------------------------------
# Purpose: Script to read an input on the TX with module fttxpy and read
#          a temperature on the TXT while both connected to the sound module
#          (Linux only)
# Date:    27-09-2018
# Author:  M. Vogelaar
#
# Remarks: We assume you have permissions to access /dev/ttyACM0
#          Otherwise read the tutorial how to create a udev rule.
#
#          The program reads an LDR on I4 on the TX controller first.
#          If above a certain threshold, the state of I1 changes.
#          A sound on the sound module is activated with O1 on the TX.
#          Then the TXT starts to measure a temperature with an NTC
#          on I3 on the TXT.
#          After 500 measurements, the loop stops and another sound is
#          activated on the sound module, this time by O8 on the TXT
#          The script can be aborted with ctrl-c
#-------------------------------------------------------------------------------
from time import sleep
import ftrobopy
from fttxpy import fttxpy

txt = ftrobopy.ftrobopy('192.168.7.2', 65000)      # TXT Connected by USB cable
tx = fttxpy()                                      # TX  Connected by USB cable

print("Light LDR on Input 4 on TX to start the TXT. Abort program with ctrl-c")
master = tx.robotx()
I1 = master.input(4)
O1 = master.output(1)
state = I1.state()

while 1:
   ns = I1.state()
   if ns != state:
      print("state() =", ns)
      state = ns
      break
   sleep(0.2)

O1.setLevel(255)
sleep(0.2)
O1.setLevel(0)

# We don't need these objects anymore
del O1
del I1
```

TxandTXTsoundmodule.py: (continued) Combine the TX and TXT in one Python program

```
print("Start measuring temperature 500 times on the TXT")

NTC = txt.resistor(3)  # LDR on I3 on TXT
O8 = txt.output(8)

temp = -1
i = 0
while 1:
    txt.updateWait()
    newtemp = NTC.ntcTemperature()
    if newtemp != temp:
        print("Current temperature is {:.1f} C     \r".format(newtemp), end='\r')
        temp = newtemp
    i += 1
    if i == 500:
        O8.setLevel(255)
        sleep(0.2)
        O8.setLevel(0)
        break

txt.stopOnline()
```

15 TXT tips

15.1 Changing the battery

A very handy sheet with tips for the TXT can be found at:
http://users.tpg.com.au/users/p8king/fischer/txt_tips.pdf
Here we read for example how to change the battery (Lithium CR2032) in the TXT.

15.2 Wired connections

Often experiments fail if you use old and frequently used Fischertechnik wires and little plugs because the often connect very well so before debugging your script, make sure the hardware is ok.

15.3 Python Index numbers

The modules we discussed use Python lists. The first element of any Python sequence has index 0. In *ftrobopy* there are several routines which expect such an index number as input or return a list where the first element is addressed with 0 (e.g. `getConfig`()). But all high-level routines expect a number equal to the input or output socket (e.g. `resistor`(1) for input 1). So it is important to keep an eye on that to prevent a lot of necessary debugging of your code.

15.4 Downloads

All scripts and this tutorial (tutorial.pdf) can be downloaded from:
https://www.dropbox.com/sh/mlzpa62h4udz3bc/AAA-Dk_6FQc-tohMSI0ep1cta?dl=0