fischertechnik ROBOTICS TXT Controller C-Programming Kit Firmware Version 4.1.6

Table of contents

Communication Methods

The online mode communication with the fischertechnik ROBOTICS TXT Controller works via TCP/IP communication to port 65000 on the TXT. The C programming kit contains C++ source code for a class handling the communication. This C++ class serves as an example for implementing the TXT communication protocol as well as a ready to implementation which converts the TXT TCP/IP protocol to a transfer area based protocol as used by the TX controller. The C++ code also contains functions for retrieving JPEG camera frames and sample code for decoding JPEG images to raw YUV.

The C++ wrapper is not necessarily required. In other languages which offer TCP/IP communication, the communication protocol can be implemented directly. The C++ source code my serve as example code for implementation of the protocol in other languages. Of cause users with special needs can also implement their own C++ library, which does not use a TX compatible transfer area.

Note on download mode communication: A separate download kit will be released later, but for those who cannot wait a note: A program running on the TXT can communicate to the ROBOPro app via TCP/IP in the same way as a PC in online mode. You just need to use localhost or 127.0.0.1 as host name when connecting. The sample code uses posix sockets, so a port to Linux should be fairly easy.

Folder structure / example projects

The communication kit has the following folder structure:

- The root folder contains a Microsoft Visual C++ solution file (OnlineSamples.sln)
- Common: C++ header and implementation files for the communication protocol
    - ftProInterface2013SocketCom.h/.cpp: direct TCP/IP communication protocol
    - ftProInterface2013TransferAreaCom.h/.cpp: wrapper for transfer area based communication and sample for direct communication
    - ftProInterface2013JpegDecode.h./cpp: wrapper for JPEG decoder
    - common.h / FtShmem.h: transfer area header files (compatible with TX)
- MotorOnOffSwitch: A sample program (Main.cpp + Microsoft Visual C++ project file) which switches a motor on and off using a switch on master and extension interface.
- Camera: A sample program (Main.cpp + Microsoft Visual C++ project file) which shows how to retrieve and decode a JPEG image from the camera.
- Jpeg: The independent JPEG group JPEG decoder / encoder with Microsoft Visual C++ library project file. The supplied sources are the variant delivered with wxWidgets. The source for the stand alone JPEG decoder and encoder application are not included.

In case you don't have Microsoft Visual Studio 2008 or later, don't worry. All projects are simple and it should be no problem to set them up with any other C environment. Linux environments should already contain a JPEG decoder library, which is compatible with the sources in this kit, so there is no need to compile this on Linux PCs and embedded Linux like the TXT.

Compiling and running the sample programs

If you want to run the sample programs, you first need to compile them. If you use Micorosoft Visual Studio, just open the file OnlineSamples.sln in the root folder of the kit. Right click on the project you want to run (Camera or MotorOnOffSwitch) and select “Set as StartUp project”. Then select “Start debugging” or “Start without Debugging” from the Debug menu (the exact names might be slightly different, depending on the Visual Studio Version). This should automatically compile and link the sample and run it.

It is expected that the TXT is connected via USB if you run the sample programs. If the TXT is connected via WiFi or Bluetooth, you need to change the IP address in Main.cpp in the call to the ftIF2013TransferAreaComHandler constructor.

Description of the Communication Protocol

The TCP/IP communication protocol is described in the header file ftProInterface2013SocketCom.h. The program needs to opens a socket to port 65000 at address 192.168.7.2 (for USB connection, 192.168.8.2 for WiFi and 192.168.9.2 for Bluetooth), sends one or more command packets and receives for each command packet a response packet. Each command and response packet starts with a 4 byte command or response id. The IDs are random numbers and serve as command ID and “magic code” at the same time. The command and response IDs are listed in ftProInterface2013SocketCom.h in the enums `ftIF2013CommandId` and `ftIF2013ResponseId`. For each command and response the header file contains a C structure. For example the structure `ftIF2013Command_QueryStatus` belongs to the command `ftIF2013CommandId_QueryStatus`. The C structures are sent to and received from the socket as little-endian binary data. Intel based PCs and the TXT are little-endian but some ARM based embedded controllers might be configured as big-endian, so that the byte order must be swapped on send and receive. If you don't know what big- or little-endian means, look for endianness in wikipedia. But as long as you use a PC to control the TXT, you don't need to worry about endianness.

Most of the commands and responses are straight forward fixed length commands. The only more complicated command is `ftIF2013CommandId_ExchangeDataCmpr`, which does a compressed data transfer and uses variable length data. The file `ftProInterface2013SocketCom.h/.cpp` contains two C++ classes for compressing and decompressing the transfer data.

It follows a description of the individual commands, in the order in which they are usually issued.

ftIF2013CommandId_QueryStatus

This command requests the name of the TXT controller and the software version number. It can also be used as ping command to check if the connection is OK.

Command Structure	**ftIF2013Command_QueryStatus**
m_id	Command id = ftIF2013CommandId_QueryStatus
Response Structure	**ftIF2013Response_QueryStatus**
m_id	Response id = ftIF2013ResponseId_QueryStatus
m_devicename	Name of the TXT controller
m_version	Version code, e.g. 0x04010600 for version 4.1.6

ftIF2013CommandId_StartOnline

This command puts the TXT in online mode. A download program (if active) is stopped, the touch user interface of the TXT is blocked and the compressed data transfer is reset.

This command must be called before any ftIF2013CommandId_ExchangeData, ftIF2013CommandId_ExchangeDataCmpr or ftIF2013CommandId_UpdateConfig command.

It is recommended to issues a ftIF2013CommandId_UpdateConfig command for each active master / extension before the first data exchange command.

Command Structure	**ftIF2013Command_StartOnline**
m_id	Command id = ftIF2013CommandId_StartOnline
Response Structure	**ftIF2013Response_StartOnline**
m_id	Response id = ftIF2013ResponseId_StartOnline

ftIF2013CommandId_UpdateConfig

This command requests the name of the TXT controller and the software version number. It can also be used as ping command to check if the connection is ok.

Command Structure	**ftIF2013Command_UpdateConfig**
m_id	Command id = ftIF2013CommandId_UpdateConfig
m_config_id	A configuration id counter which starts at 0 and needs to be incremented on each change of configuration. An update config command is ignored, if the m_config_id does not change. The first configuration usually has m_config_id=1.
m_extension_id	0 for master, 1 for extension (or higher numbers when more extensions are supported by the TXT firmware).
m_config	A structure containing configuration data. For compatibility reasons the structure is the same as in the TX C programming libraries. The most relevant fields are the universal input configuration. Please note that you need to configure the kind of input (resistance, voltage, …) and the choice if it is analog or digital. E.g. a usual switch input is a digital resistance input. The motor outputs needs to be configures as either combined motor output or as 2 independent pairs. This is used to control power saving of the motor outputs. Please note that motors are always programmed in 512 steps. The 8/512 step distinction in ROBOPro is handled inside of ROBOPro.
Response Structure	**ftIF2013Response_UpdateConfig**
m_id	Response id = ftIF2013ResponseId_UpdateConfig

ftIF2013CommandId_ExchangeData

This is the simpler of two online-mode data exchange commands. It support only a master controller, no extensions. The advantage of this command is, that the data transmission is not compressed and straight forward.

Command Structure	**ftIF2013Command_ExchangeData**
m_id	Command id = ftIF2013CommandId_ExchangeData
m_pwmOutputValues	The (PWM) pulse with modulation values between 0 and 512 for the 8 outputs. A motor output uses two consecutive values (M1 uses index 0 and 1). If the output is used as motor output, always one of the 2 outputs is 0, while the other value is between 0 and 512.
m_motor_master	This is used to synchronize one motor to another motor. 0 means the motor is independent. 1..4 means the motor is synchronized to motor 1..4. If a value of 5..8 is given, it is possible to program deviations from the synchronization using the m_motor_distance values. This is called “sync error injection”. This is useful e.g. for closed loop trail tracking. If this value is changed, m_motor_command_id must be incremented.
m_motor_distance	If this value is not 0, the motor will stop after the corresponding counter input counted the given number of pulses. In sync error injection mode, this is used as described above. If this value is changed, m_motor_command_id must be incremented. Distance commands automatically reset the counter.
m_motor_command_id	This value needs to be incremented whenever m_motor_master or m_motor_distance change. m_pwmOutputValues can be changed without incrementing the command id. A distance command is finished, if the m_motor_command_id in the response structure has the same value.
m_counter_reset_command_id	If this values is incremented, the corresponding counter is reset. The reset is finished, if m_counter_command_id in the response structure has the same value.
m_sound_command_id	This value must be incremented, whenever m_sound_index or m_sound_repeat change, or to play the same sound again.
m_sound_index	Index of the sound to play, 0=no sound. The index numbers of the sounds are given in appendix A at the end of this document. Please note that it is possible to exchange the sound files on the TXT filesystem in folder /opt/knobloch/SoundFiles using ssh and/or scp. The file name must start with a 2 digit number stating the sound index. The sound files are 8-bit mono 22050 Hz and can be created e.g. with Audacity.
m_sound_repeat	A repeat count for the sound. 0 means indefinite. Sounds can be stopped at any time by sending a new sound command with m_sound_index.

Response Structure	**ftIF2013Response_ExchangeData**
m_id	Response id = ftIF2013ResponseId_ExchangeData
m_universalInputs	Values of the 8 universal inputs. Depending on the configuration this is either a analog value or 0 or 1 for digital inputs.
m_counter_input	The current values (0 or 1) of the 4 counter inputs.
m_counter_value	The count value of the 4 counter inputs. The counter value can be reset by incrementing m_counter_reset_command_id.
m_counter_command_id	This value changes to the last m_counter_reset_command_id in the command structure after a reset command finished.
m_motor_command_id	This value changes to the last m_motor_command_id in the command structure after a motor distance command is finished.
m_sound_command_id	This value changes to the last m_sound_command_id in the command structure after a sound playback finished.
m_ir	This array of structures contains the infrared remote control input values. The values are given once for each combination of switches, so that up to 4 remote controls can be used. The 5th structure (with index 4) responds to a control with any switch setting. Please note, that the switch states are also submitted, but not when the switch changes. They are only submitted when some other value changes.

ftIF2013Command_ExchangeDataCmpr

The detailed description of this command is beyond the scope of this document. Please refer to the documentation of the structure in ftProInterface2013SoecketCom.h, the classes CompressionBuffer and ExpansionBuffer in the same file and the usage of the command in the member function DoTransferCompressed of the ftIF2013TransferAreaComHandler class. The field names and meanings are the same as for the uncompressed data transfer command.

The TXT checks the transmitted CRC of the ***decompressed*** data. If the CRC doesn't match, the online mode is aborted. This way the compression algorithms as well as the data transmission is checked.

ftIF2013CommandId_StopOnline

This command puts the TXT back into idle mode. After this command, data exchange and update config commands should no longer be sent.

Command Structure	**ftIF2013Command_StopOnline**
m_id	Command id = ftIF2013CommandId_StopOnline
Response Structure	**ftIF2013Response_StopOnline**
m_id	Response id = ftIF2013ResponseId_StopOnline

ftIF2013CommandId_StartCameraOnline

This command starts a camera server on port 65001. It usually takes about 1..2 seconds to start the server. Please note, that the camera start and stop commands are sent to port 65000. Only the frames are transferred over port 65001.

Command Structure	**ftIF2013Command_StartCameraOnline**
m_id	Command id = ftIF2013CommandId_StartCameraOnline
m_width	Camera frame width, usually 320. Please see the documentation for the StartCamera function of the ftIF2013TransferAreaComHandler class for supported resolutions.
m_height	Camera frame height, usually 240
m_framerate	Camera frame rate, usually 30
m_powerlinefreq	Frequency of the powerline. This is used to avoid flickering video images with artificial illumination.
Response Structure	**ftIF2013Response_StartCameraOnline**
m_id	Response id = ftIF2013ResponseId_StartCameraOnline

ftIF2013DataId_CameraOnlineFrame

This command is an exception of the usual command/response protocol. The camera frame lag would be to high, if the application would have to request a frame before being able to receive it. The camera server sends a frame to the TCP/IP socket as soon as it is started. The application first reads the frame and then sends an acknowledge. After receiving the acknowledge, the server sends the next frame to the socket. Also please note, that this communication happens on port 65001, not on port 65000 as all other communication.

Data Structure	**ftIF2013Response_CameraOnlineFrame (received)**
m_id	Data id = ftIF2013DataId_CameraOnlineFrame
m_numframeready	Number of frames ready in the queue. If this number gets >1, the transfer is too slow and the lag increases.
m_framewidth	Camera frame width
m_frameheight	Camera frame height
m_framesizeraw	Decompressed size of the frame, usually width*height*2
m_framesizecompressed	Size of the frame compressed as JPEG. This number of bytes follows directly after the structure. Typically first this header structure is read and then the data into a separate memory.
m_framedata	0-sized dummy array receiving the JPEG compressed frame data.
Acknowledge Structure	**ftIF2013Acknowledge_CameraOnlineFrame (sent)**
m_id	Acknowledge id = ftIF2013AcknowledgeId_CameraOnlineFrame

ftIF2013CommandId_StopCameraOnline

This command stops the camera server on port 65001.

Command Structure	**ftIF2013Command_StopCameraOnline**
m_id	Command id = ftIF2013CommandId_StopCameraOnline
Response Structure	**ftIF2013Response_StopCameraOnline**
m_id	Response id = ftIF2013ResponseId_StopCameraOnline

The ftIF2013TransferAreaComHandler class

This class is provided as an example implementation of the protocol described above as well as a convenient wrapper to a transfer area based protocol. The usage of this class is shown and documented in the two sample programs MotorOnOffSwitch.h and Camera.h. Also the header file ftProInterface2013TransferAreaCom.h contains documentation on each function. Please note the comments on thread safety and other topics at the beginning of the header file.

Appendix A: List of sound files

01_Airplane.wav

02_Alarm.wav

03_Bell.wav

04_Braking.wav

05_Car-horn-long.wav

06_Car-horn-short.wav

07_Crackling-wood.wav

08_Excavator.wav

09_Fantasy-1.wav

10_Fantasy-2.wav

11_Fantasy-3.wav

12_Fantasy-4.wav

13_Farm.wav

14_Fire-department.wav

15_Fire-noises.wav

16_Formula1.wav

17_Helicopter.wav

18_Hydraulic.wav

19_Motor-sound.wav

20_Motor-starting.wav

21_Propeller-airplane.wav

22_Roller-coaster.wav

23_Ships-horn.wav

24_Tractor.wav

25_Truck.wav

26_Augenzwinkern.wav

27_Fahrgeraeusch.wav

28_Kopf_heben.wav

29_Kopf_neigen.wav