Parallel transmission (pTx) medical implant safety testbed

Software Instructions

Revision History

Rev.	Date (YYYY-MM-DD)	Description of change	Author/ Contributions
1.0	2022-11-14	Initial version	Berk Silemek Lukas Winter [illegible]

About

Please see the software documentation for pTx medical implant safety testbed. You can contact Lukas Winter (lukas.winter@ptb.de) if you have any questions, suggestions, or errors. It would be helpful for the project and for the others to reproduce and build upon this work.

Commented [BS1]: Double check.
Todo: Website is going to be updated

If you find pTx medical implant safety testbed useful in your work, please cite following paper:

Winter L, Silemek B, Petzold J, et al. Parallel transmission medical implant safety testbed: Real-time mitigation of RF induced tip heating using time-domain E-field sensors. Magnetic Resonance in Medicine 2020;84:3468–3484 doi: 10.1002/mrm.28379. (https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.28379)

This work was funded by the EMPIR grant 17IND01 MIMAS. The EMPIR initiative is co-funded by the European Union's Horizon 2020 research and innovation program and the EMPIR participating states.

This work has received funding from the European Partnership on Metrology, co-financed by the European Union’s Horizon Europe Research and Innovation Programme and by the Participating States, under grant number 21NRM05 STASIS.

This work is supported by the Open Source Imaging Initiative (OSI²), https://www.opensourceimaging.org/ pTx-implant-safety-testbed

Introduction

The document provides the software structure of the pTx medical implant safety testbed.

The detailed description of the whole project can be found in this link:

https://www.opensourceimaging.org/pTx-implant-safety-testbed

Also, the paper describes the system, and the experiments can be found here:

Winter L, Silemek B, Petzold J, et al. Parallel transmission medical implant safety testbed: Real-time mitigation of RF induced tip heating using time-domain E-field sensors. Magnetic Resonance in Medicine 2020;84:3468–3484 doi: 10.1002/mrm.28379. (https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.28379)

The system utilizes another open-source hardware project "COSI Measure". The detailed version of the COSI Measure system can be found here:

http://www.opensourceimaging.org/project/cosi-measure/

Also in the corresponding paper:

Han H, Moritz R, Oberacker E, Waiczies H, Niendorf T and Winter L, „Open Source 3D Multipurpose Measurement System with Submillimetre Fidelity and First Application in Magnetic Resonance", Scientific Reports, 7:13452, 2017

System Overview

Implant safety testbed consists of various hardware components. [1] Hence, there are some code folders for each module. All scripts can be merged into one folder (except, COSI measure GUI). They separated into 4 folders inside this repo, named as: ADC, Cosi_measure, keithley and Transmit.

These folders includes .c codes for the digitizer m4i.4451 [2] and the transmitter m4i.6622 [3] and Fortran codes implements .c codes as c-routines. The compilation of the codes can be done using the first line of each Fortran files. Everything is implemented under Cent-OS system. Other Linux distributions or windows were not checked.

It is also possible to use other software platforms for the digitizer and transmit cards supporting various environments such as MATLAB and Python. The function names to control the cards are the same throughout different environments. Hence, mitigation to other software platforms is straightforward, but should be checked. For more information, please see the documentation and examples from the manufacturer's website. [3] In addition, please follow the manufacturer's website to install corresponding libraries because they differ by the OS.

ADC

This file contains the functions for the ADC card setup and receiver operation. The .c file (m4iset_extrig_rms.c) implements the functions for the m4i card. The routine uses 3 arguments:

*pvData ------> 16-bit data pointer for the received data

*lMemsize ---> 32-bit memory length for the received data

*posttr -------->32-bit post trigger pointer to adjust the time window of the ADC. Please see manufacturer's documentation for more about post trigger.

The other two Fortran codes (rec_m4i_fifo_tdfilter.f and rec_m4i_tdfilter_rms.f) implements the c routine. They need to be compiled with a Fortran complier. The first line in these files can be taken as an example i.e. copy and paste into the OS terminal using the working folder.

Transmit

Transmit folder contains similar architecture as the ADC implementation. The .c files programs the transmit cards. The script implements the functions for the m4i.6622 transmit functions. The ‘m4iset_8ch.c’ file initiates the card for 8-channel transmit operation using 400 MHz sampling frequency (297MHz output, please see the paper for the reasoning about transmit frequency selection [1]). For 123 MHz (3T). The sampling frequency can be changed to 625MHz. The main function is named ‘m4iset’ takes three arguments as follows:

*pvData -----------> 16-bit data pointer for the first transmit card (each card has 4 transmitter)

*pvData1 ----------> 16-bit data pointer for the second transmit card

* lMemsize -------->32-bit memory length for the transmit data

The Fortran codes implements the .c code and generates the transmit data. In this case the transmit pulse is a 297 MHz sinusoidal signal with a 100 % duty cycle. The Fortran implementation takes two arguments its operation.

mode-----------> This determines the mode of operation. This can be either 0 or 1 as integer. The mode 0 prepares and transmits a composite pulse. The composite pulse is composed of 1 millisecond long signals. The 1 millisecond signals are subsequently transmitted for each channel. The second mode transmits a parallel transmit pulse that is generated based on a mode. The total power is normalized. The length of the pulse is approximately 1 ms.

att--------------->This parameter adjusts the attenuation level for the output. It can take floating values between [0,1].

Keithley for thermistor measurements

This folder contains two Python scripts. The script Keithley_Interface_25Hz.py reads resistance value from Keithley2000 multimeter and save it to a text file with system timestamps. The calculated temperature value is saved as well. Note that this temperature calculation is thermistor specific. Please revisit thermistor parameters for a different product.

You can run the script via a terminal or with ide. The measurement parameters of the multimeter were adjusted manually. Therefore, the device must be taken to the resistance measurement manually. The filter and averaging in the device were also set to off to enable fast operations. The LCD is turned off from the script. You can enable it later.

The serial connection is adjusted to 19200 baudrate. The multimeter should also be adjusted to the same baudrate. In addition, the serial connection port is not automatically adjusted. One should determine the USB port manually and change it in script.

The timing settings are about the limits of the Keithley2000 in this NPLC settings. The device may give errors, beeps, and sends some incorrect data. Not all of them are handled. You can decrease the reading rate by changing the 'rate' parameter. About 13-15Hz is usually ok, when LCD is open and NPLC is in default mode.

The other script plots the data for every 3 seconds. The matplotlib backend is adjusted to Tkagg for Linux compatibility. It should also work in Windows. Other backends like Qt can be realized as well.

COSI Measure modifications

This folder has two python scripts for the automated mapping. One script has some wraparound of the previous implementation from Han et al. [4] The other one works under the server computer that prepares and sends the coordinates. An ethernet cable is needed between host (robot computer) and server (m4i computer).

The cosi_gui_testbed.py script is an improved version [4] but not significantly different. There is an individual homing option for each axis and each direction. If wanted, the buttons (x+, x- ,y+ ,y- ,z+ ,z-) can be used individually. Compared to previous implementation, the probe measurement is disabled. That is: when robot goes to a coordinate, it does not perform a measurement. The measurement part is mitigated to server part that is implemented in another python script (server_GUI.py).

The ethernet connection parameters are given in the script. A TCP/IP protocol is used for the ethernet communication. The robot computer's ethernet settings can be configured before the connection. For security, a button is placed into Cosi_GUI application and labelled as 'Connect external PC'. When it is clicked, Cosi_GUI is deactivated and awaits for the incoming connection and data. After the connection is established, the robot waits for custom commands for the operation. Three keywords are "MOV", "OK" and "CLOSECONN".

The MOV command is for the movement to absolute coordinate. The command structure can be in these two forms: MOVXXXxxYYYyyZZZzz or MOVXXXYYYZZZ. The first one is for submillimeter precision is 2, first 3 digits are the coordinates in mm for xyz. The second one is in millimeter. Please pad with zeros. i.e. 58.23 is 05823 or 058 for the host part. When the robot finishes its operation (i.e. moving to desired coordinate), it acknowledges the movement and sends back a “MOVOK” command to a host.

“CLOSECONN” terminates the connection. This activates Cosi_GUI application again. If the connection is terminated unexpectedly, it can be killed using another terminal. Otherwise, the application must be restarted. (Please see the screenshots for how to kill an open TCP/IP process without restarting the whole Cosi_GUI application)

“OK” command can be sent for a connection check. The robot will respond this command with an “ok” message.

Server_GUI is the part implements the mapping. Currently cartesian mapping is supported by the GUI, which follows the path like a snake game. For each position, it transmits a composite pulse and calculates a worst-case vector [1]. Next, a circularly polarized, orthogonal projection and worst case transmit pulses are sent and the corresponding files are saved into a Measurements1 folder. For each position the files are saved into a folder according to the measurement number. The means of each measurement are also displayed in the GUI. Currently, creating different measurement folders was not implemented. Therefore, please check if there is an empty “Measurements1” folder in the directory.

An arbitrary path file can also be used with the GUI. For this, a path file using the same structure from Han et al. [4] should be prepared and named as ‘PathFile.txt’. And the default coordinate values from the GUI must be kept the same.

Please note that there is no limitation about the numbers that you can enter to the GUI. Therefore, make sure that COSI measure is in the physical spatial boundaries of the robot. A coordinate that is outside of the physical coordinates may damage the robot components. The physical dimensions of the device under test should also be considered. It is highly encouraged to determine the customized spatial boundaries of the mapping area before any operation with COSI Measure.

References

[1] L. Winter *et al.*, “Parallel transmission medical implant safety testbed: Real-time mitigation of RF induced tip heating using time-domain E-field sensors,” *Magn. Reson. Med.*, vol. 84, no. 6, pp. 3468–3484, 2020, doi: https://doi.org/10.1002/mrm.28379.

[2] “M4i.4451-x8 | Spectrum.” https://spectrum-instrumentation.com/de/m4i4451-x8 (accessed Jan. 26, 2021).

[3] “M4i.6622-x8 | Spectrum.” https://spectrum-instrumentation.com/de/m4i6622-x8 (accessed Jan. 26, 2021).

[4] H. Han, R. Moritz, E. Oberacker, H. Waiczies, T. Niendorf, and L. Winter, “Open Source 3D Multipurpose Measurement System with Submillimetre Fidelity and First Application in Magnetic Resonance,” *Sci. Rep.*, vol. 7, no. 1, p. 13452, Dec. 2017, doi: 10.1038/s41598-017-13824-z.