

# **Helmholtz Test Bench Control Software Documentation**

Author:

Martin Zietz

Supervisors:

M.Sc. Markus T. Koller

M.Sc. Lukas-Maximilian Loidold

Institut für Raumfahrtssysteme, Universität Stuttgart

März 2021

# Contents

<b>List of Figures</b>	<b>ii</b>
<b>List of Tables</b>	<b>iii</b>
<b>List of Symbols</b>	<b>iv</b>
<b>1. Software Implementation</b>	<b>1</b>
1.1. Program Structure . . . . .	1
1.2. Program Files . . . . .	2
1.2.1. Main Executable <code>main.py</code> . . . . .	2
1.2.2. Global Variables File <code>globals.py</code> . . . . .	2
1.2.3. Test Bench Control File <code>cage_func.py</code> . . . . .	3
1.2.4. Graphical User Interface <code>User_Interface.py</code> . . . . .	5
1.2.5. Sequence Execution File <code>csv_threading.py</code> . . . . .	7
1.2.6. Configuration Handling File <code>config_handling.py</code> . . . . .	8
1.2.7. Data Logging Handling File <code>csv_logging.py</code> . . . . .	8
1.2.8. Hardware Control Libraries . . . . .	9
1.3. Conversion to Executable . . . . .	9
<b>2. Operating Instructions</b>	<b>10</b>
2.1. Test Bench Assembly Instructions . . . . .	10
2.1.1. Disassembly Procedure . . . . .	10
2.1.2. Reassembly Procedure . . . . .	12
2.2. Software Users Guide . . . . .	13
2.2.1. Installation . . . . .	13
2.2.2. User Interface Elements . . . . .	13
2.2.3. Hardware Connections and Program Setup . . . . .	21
<b>Bibliography</b>	<b>22</b>
<b>A. Example Program Auxiliary Files</b>	<b>23</b>

## List of Figures

1.1. Software file architecture . . . . .	1
1.2. User interface (example state) . . . . .	5
2.1. Test bench assembly drawings . . . . .	10
2.2. Main connector (X-cables temp.) . . . . .	11
2.3. Intermediate (dis-)assembly step . . . . .	11
2.4. Manual input mode user interface . . . . .	14
2.5. CSV sequence mode user interface . . . . .	16
2.6. Data logging configuration page . . . . .	18
2.7. Program settings page . . . . .	19

## List of Tables

2.1. Status display entries . . . . .	13
2.2. Settable program constants . . . . .	20

# List of Symbols

Symbol	Description	Unit
$B$	Magnetic flux density	T
$B_0$	Ambient magnetic flux density	T
$B_{max}$	Maximum achievable magnetic flux density	T
$B_{max}$	Minimum achievable magnetic flux density	T
$I$	Current	A
$I_{max}$	Maximum current	A
$K$	Coil constant	$\frac{T}{A}$
$R$	Electrical resistance	$\Omega$
$U$	Voltage	V
$U_{max}$	Maximum voltage	V

# Glossary

<b>API</b>	Application Programming Interface
<b>COM</b>	Communication Port
<b>CSV</b>	Comma Separated Values file
<b>IRS</b>	Institute of Space Systems at the University of Stuttgart
<b>LED</b>	Light-Emitting Diode
<b>PC</b>	Personal Computer
<b>PSU</b>	Power Supply Unit
<b>UI</b>	User Interface
<b>USB</b>	Universal Serial Bus

# 1. Software Implementation

## 1.1. Program Structure

To operate the test bench, a Python software with graphical user interface (UI) was developed. It controls the used PS2000B Power Supply Units (PSU) as well as the Arduino microcontroller inside the switch box. This chapter focuses on the overall implementation. More detailed information is provided in the form of comments in the source code, which is available on the IRS git server.<sup>1</sup> A users guide can be found in Section 2.2.

Software development and testing were done in Windows 10 and Python 3.7. Some aspects may need to be adapted to use the software on a different operating system or Python version. The code was tested with a PSU or the switch box individually. However, integrated verification with both PSUs and the switch box Arduino connected simultaneously was not possible up to this point, as some of the equipment was located inside the IRS cleanroom.

The program file architecture is shown in Figure 1.1. This is meant to give an overview of the structure, therefore it does not show all interactions between the files.

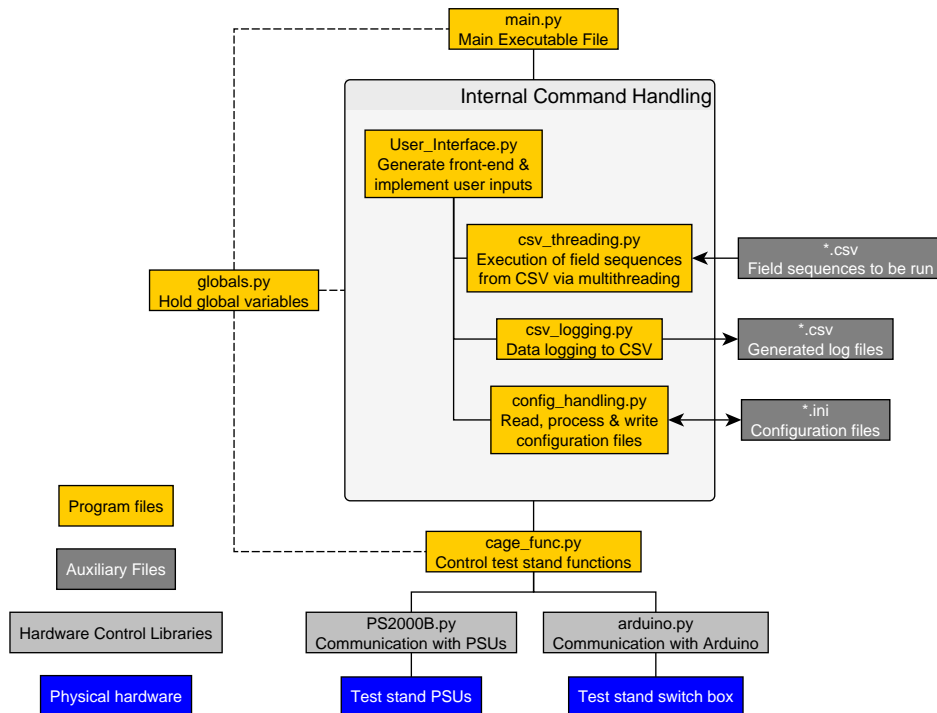


Figure 1.1.: Software file architecture

<sup>1</sup>[https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz\\_Test\\_Bench.git](https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz_Test_Bench.git)

Upon execution of `main.py`, the test bench devices (PSUs and switch box Arduino), program objects and variables are initialized. Next, the UI (controlled by `User_Interface.py`) is set up and displayed. Program elements needed for more complex functionalities are coded in `csv_logging.py`, `config_handling.py` and `csv_threading.py`.

All program elements use the classes in the `cage_func.py` file to control and read data from the test bench. Communication with the actual hardware (power supply units and switch box Arduino) is achieved via the libraries in `PS2000B.py` and `Arduino.py`, which were taken from online sources [1, 2] with only minor modifications.

`globals.py` is used to easily pass frequently used variables between the different program files. It contains mainly variable initializations, that can be accessed and changed by the other program elements after importing it.

The application is able to operate with one or all test bench devices disconnected. For example, it is possible to generate a magnetic field in only one test bench axis with just a single PSU. The program also provides error handling for a variety of device disconnects and user mistakes. This includes protection against the commanding of excessive currents and voltages, as well as the display of warnings when attempting to enter a potentially dangerous value in the settings.

## 1.2. Program Files

### 1.2.1. Main Executable `main.py`

The main executable file contains the framework code for start and end of the program.

During initialization the information from the configuration file is read out to set the necessary constants. Then the test bench is prepared using the `setup_all` function (see Section 1.2.3), the UI is initialized and its `mainloop` started.

The `program_end` function in this file ensures a safe shutdown of all equipment at the end of the program. It is immediately called when the user closes the window. The function stops any other operational threads (see Section 1.2.5) and powers down the PSUs and Arduino. It also asks the user to store possible unsaved log data and then destroys the application window.

For exceptions that were not caught by lower level error handling, the main file also includes a global error protection. In such a case an error message is displayed and the `program_end` function called. This is to ensure a safe shutdown, regardless of the current program state.

### 1.2.2. Global Variables File `globals.py`

This file holds global variables and constants that are used often and by more than one module. Examples of these are `PS2000B` and `ArduinoCtrl` objects that represent the test bench devices, `Axis` objects for each spatial axis but also status indicators like `exitFlag`, which signals the end of the program to all modules.

The file also contains the default values for all constants used to run the test bench, like coil constants, resistances and maximum currents. These are stored in a hard-coded dictionary, which can be used to generate a default configuration. The dictionary also includes the minimum and maximum safe value for each constant. These limits are used to warn users if they attempt to set a value that may damage equipment. In the future, it may be advantageous to store this information in a separate configuration file to allow easier modification.



### 1.2.3. Test Bench Control File `cage_func.py`

This file contains all classes and functions directly related to the operation of the test bench. It includes the two main control classes **Axis** and **ArduinoCtrl**, functions for initialization and shutdown as well as for controlling more than one axis at a time.

#### Axis Control Class **Axis**

This is the main class representing an axis (x,y,z) of the test bench. It contains static and dynamic status information about the axis in its attributes and the means to command the axis in its methods. During program initialization an object of this class is created for each of the three axes and stored in `globals.py`

Apart from methods to get status information from the appropriate PSU, the class contains the primary way to command the test bench in form of the **set\_signed\_current** method. Its parameter is a positive or negative current value in Ampere. The method first checks if this value exceeds the safe limits defined in the program configuration. If not, the appropriate PSU channel and the switch box Arduino are commanded to supply the desired current and to actuate the polarity change relay as needed.

To generate a specific magnetic field, the **set\_field** method can be called. It takes a given field value  $B$  in Tesla and calculates the needed current:

$$I = \frac{B - B_0}{K} \quad (1.1)$$

$B_0$  is the background magnetic field in the measurement area for this axis in Tesla and  $K$  is the coil constant in  $[\frac{T}{A}]$ . The calculated value  $I$  is passed to the **set\_signed\_current** method to command the test bench. The **set\_field\_simple** method works the same way, but without subtracting the ambient field.

Similarly the minimum and maximum achievable field values  $B_{min}$  and  $B_{max}$  are calculated in the class initialization, mainly in order to provide this information to the user:

$$B_{max} = B_0 + I_{max} * K \quad (1.2)$$

$$B_{min} = B_0 - I_{max} * K \quad (1.3)$$

Here  $I_{max}$  is the maximum allowed current, as defined in the configuration page of the UI.

#### Arduino Control Class **ArduinoCtrl**

This is the main class used to control the Arduino microcontroller inside the switch box. It inherits the **Arduino** class from the Arduino command application programming interface (API) [2]. This provides a way to pass commands directly through the class object, which is created during program initialization and stored in `globals.py`.

Most commands to the switch box are passed directly through the inherited **Arduino** class. For example, the `digitalWrite()` method is called to energize a pin that actuates a specific relay. The purpose of the **ArduinoCtrl** class is mainly to provide supporting functionality directly related to the specific use case on the test bench, including:

- Configuration of output pins used to trigger polarity switch relays
- Checking Arduino connection and output pin status
- **safe** method to set all used output pins to "LOW" for safe shutdown

**Setup Function** `setup_all`

This is the main initialization function. It is used at application start-up and during the runtime whenever the user updates settings or (re)connects a device. Its main tasks are:

- Read the latest information from the configuration object (see Section 1.2.6)
- Initiate communication with switch box Arduino and create object of `ArduinoCtrl` class
- Initiate communication with PSUs
- Create object of class `Axis` for each axis (x,y,z)

During these tasks it also handles different error cases, like disconnected devices or wrong settings in the configuration object.

**Shutdown Function** `shut_down_all`

This function is used to safely shut down all devices at the end of the program or if an error occurs. It is called in the `program_end` function of `main.py`. Its primary tasks are:

- Secure all connected PSUs
  - Set voltages and currents on both channels to 0 V and 0 A
  - Disable power outputs on both channels
- Secure Arduino
  - Set all used output pins to "LOW"
  - Close serial connection
- Show message to user with shutdown status of all devices and any errors encountered

During this process different error cases, like disconnected devices, are handled and the user is informed.

**Other Functions**

In addition to those discussed above, the file also contains functions to:

- Command currents or magnetic fields on all axes, based on a given vector
- Check if all devices are still connected
- Check if a given value is within the safe limits defined in `globals.py`

### 1.2.4. Graphical User Interface User\_Interface.py

This file is used to construct the UI, that the user interacts with to control the test bench. It contains several classes, each with code to build a specific section of the UI as well as methods to perform the actions that are triggered by different user inputs in that section.

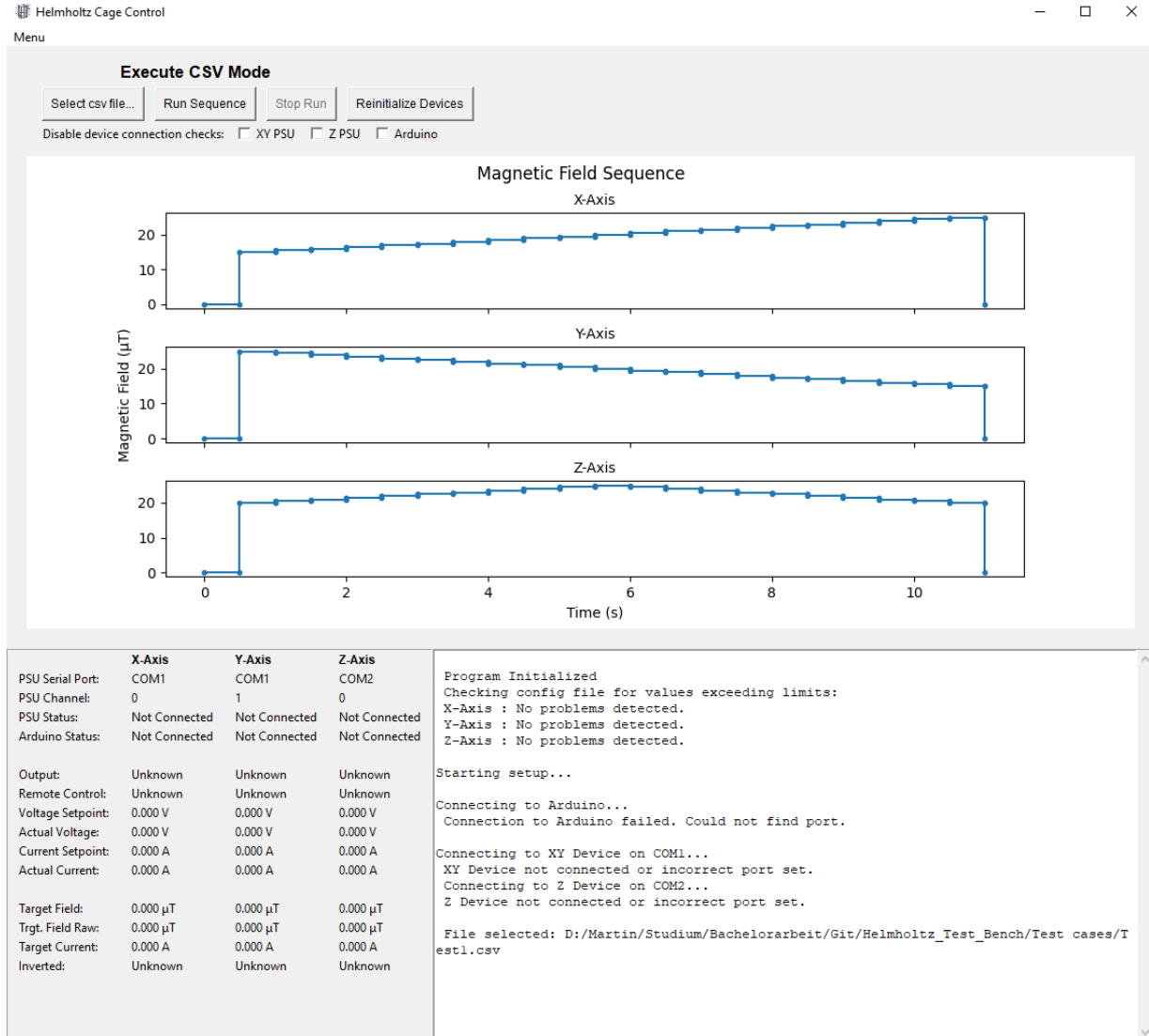


Figure 1.2.: User interface (example state)

The general layout of the application window is shown in Figure 1.2. At the bottom left is a status display, that is constantly updated with the current state of all connected devices. To its right is a text area that can be used to print out information to the user, similar to a basic console output. The top half is the main area, used to display the interactive elements. Its content changes depending on which mode the user has selected. At the very top of the window a drop-down menu is used to switch between the different modes. For more detailed usage instructions please refer to Section 2.2.

The interface is setup through a central object of class `HelmholtzGUI`, which inherits the `Tk` main application class from the `tkinter` library. The other UI element classes (except the top menu) inherit from `tkinter.Frame` and are placed in the layout by the main object. This structure is based on a proposal by H. Kinsley [3]. Each class is briefly described below.

#### **Main Application Class `HelmholtzGUI`**

An instance of this class represents the application window. The program `mainloop` is running on this object. The major UI elements are initialized here and placed at their appropriate positions. This class's only method is `show_frame`, which is used to switch between the different operating modes by displaying their respective frame in the main area.

#### **Menu Bar Class `TopMenu`**

The menu bar at the very top of the application window is constructed by this class. It contains methods to implement each option of the menu. At this time, it is only used to switch between modes, but more functionalities could be added in the future if needed.

#### **Static Manual Input Mode Class `ManualMode`**

The manual input mode interface is used to manually command static values of magnetic fields or currents on the test bench. It is placed in the main area of the application window, the layout can be seen in Figure 2.4. The methods provided in this class interface mainly with the `cage_func.py` file to command the devices.

#### **CSV Sequence Execution Mode Class `ExecuteCSVMode`**

This class constructs and operates the interface for executing magnetic field sequences from a Comma Separated Values (CSV) file. It is placed in the application's main area, its layout is shown in Figure 2.5.

Apart from a method to load CSV files, the class manages the separate thread that needs to be created every time a sequence is run. For this purpose, it interfaces mainly with the `csv_threading.py` file to initialize, start and (if needed) stop the thread object. More details on the multithreading functionality is provided in Section 1.2.5.

#### **Settings Page Class `Configuration`**

The program settings page is constructed in this class. It also controls the reading and writing of configuration files. Like the actual test bench control modes, the settings page is placed in the application's main area. Its layout can be seen in Figure 2.7.

The class generates and reads out all entry fields the user can use to set program constants like used serial ports, resistances and current limits. It interfaces with the `config_handling.py` file to store this information in a `ConfigParser` object and to read and write it from and to configuration files (see Section 1.2.6).

Individual entry fields can be highlighted to point out values that exceed the safe limits defined in `globals.py` to the user.

#### **Data Logging Configuration Page Class `ConfigureLogging`**

This class generates the page for configuring the logging of test bench data to a CSV file. Similarly to the settings page it is placed in the main area, its layout is shown in Figure 2.6.

The page provides a number of checkboxes used to select what specific data is to be logged.

This information is stored in a list of string keys. Whenever a row of data needs to be logged, a method in the class calls the `log_datapoint` function in `csv_logging.py`. To indicate what data to log, the string key list is passed as the function parameter (see Section 1.2.7).

There are two options to control when and how often data is logged: in regular intervals and on event. These can also be used at the same time. The information, which of these is enabled, is stored in two boolean attributes of the class object. For logging in regular intervals a method in the `ConfigureLogging` class object periodically calls itself.

For logging on event, a data point is generated whenever a significant change on the test bench is commanded. As of now, this is simply done by inserting a piece of code in every function where it was seen as appropriate. However, this is somewhat inconsistent and carries the risk of missing some state changes. A better balance between logging consistency and excessive data generation should be devised in a future update.

### **Status Display Class `StatusDisplay`**

The status display (bottom left in Figure 1.2) is used to monitor the state of the test bench devices. The individual values are displayed and updated through labels tied to variables of type `tkinter.StringVar()`, which are stored in a dictionary. The `update_labels` method then polls the current values from the `Axis` and `ArduinoCtrl` objects described in Section 1.2.3 and updates the label variables. This is done both periodically and on major test bench commands.

### **Output Console Class `OutputConsole`**

The output console in the bottom right of the UI (see Figure 1.2) is generated in this class. The main body is a `tkinter.Text` widget. Printing of information to it is done via the custom `ui_print` function, which can be used exactly like the built-in `print`.

## **1.2.5. Sequence Execution File `csv_threading.py`**

This file contains code for executing a timed sequence of magnetic field vectors from a CSV file. To do this without interfering with the UI, it must run in a separate thread.

The main class for this is `ExecCSVThread`. It inherits the `Thread` class from the `threading` library, so each of its instances represents a unique thread. Its main method, apart from those needed to start and stop it, is `execute_sequence`. It takes the desired sequence in the form of a `numpy` array and commands the test bench at the appropriate times. When a new field vector is set, all related actions need to be performed before the scheduler returns to the main thread. A `threading.Lock` object is used to ensure this. The method also continuously checks that all devices are still connected and that the run has not been aborted by user input or closing of the main application window.

Apart from the main class this file contains functions to read data from a CSV file to a `numpy` array and check that no values in it exceed the test bench limits. There is also a function to generate a plot of the data for display in the UI. The line plot is modified to reflect the discrete and nearly instantaneous change of fields in the test bench (see result in Figure 1.2).

### 1.2.6. Configuration Handling File `config_handling.py`

This file contains functions and variables for reading and writing configuration files. The processing is done using the `configparser` library.

The current program configuration is stored in `CONFIG_OBJECT`, an instance of `ConfigParser`. It contains all configurable information and can be stored in its entirety in a `.ini` file. The `write_config_to_file` and `get_config_from_file` functions are used to read/write the entire `CONFIG_OBJECT` from/to such a file. An example is provided in Appendix A.

To access or change specific values the `read_from_config` and `edit_config` functions are provided. The latter also checks if a value is within the safe limits defined in `globals.py`. If it is not, the user is warned and asked to confirm, before the value is written to `CONFIG_OBJECT`.

The `check_config` function does this check for all values of a provided configuration object. If excessive values are found, a warning message is displayed and the configuration UI page opened, where they are highlighted.

The last function of the file is `reset_config_to_default`, which overwrites the entire `CONFIG_OBJECT` with the default values defined in `globals.py`.

### 1.2.7. Data Logging Handling File `csv_logging.py`

This file handles the logging of test bench status data to a CSV file, using the `pandas` library. Within the program the logged data is stored in a `pandas.DataFrame` object, which represents a table with column headers and data rows.

The information about what data can be logged and how to access the specific values is stored in a central dictionary. Its keys are the names of the values as they are displayed in the UI, e.g. "Voltage Setpoint" or "Actual Current". The keys are used as handles for accessing the dictionary elements and as labels for the checkboxes in the logging configuration UI page (see Figure 2.6). The `ConfigureLogging` class of the UI generates a list containing the keys belonging to the ticked checkboxes, which is then passed to the logging functions to indicate what data should be logged. The keys are intentionally identical to the labels used for the status display. This makes it easier for the user to know what each value means. It may also allow unifying some of the code for these two functionalities in the future.

The dictionary elements themselves are a string representation of the respective attribute names in the `Axis` class. They are used with the `getattr` command to get the values of these attributes from the individual axis objects.

To log a data point, the mentioned list of keys is passed to the `log_datapoint` function. For each of its elements, the data from all three axes is read out using `getattr` as described above. To form the correct number of column headers, the key list is expanded to include each item three times, for example `["Target Field"]` becomes `["Target Field X", "Target Field Y", "Target Field Z"]`. Together with the values this forms a new `DataFrame`, which is then appended to the previously logged data.

When the logging is finished, the entire `DataFrame` can then be saved to a CSV file using the `write_to_file` function. The file also provides functions to allow the user to choose a filename and clear the logged data.

### 1.2.8. Hardware Control Libraries

#### **PS2000B PSU Control Library** `PS2000B.py`

The code necessary to control the power supply units was adapted from S. Sprößig [1] with only minor modifications. The library provides the class `PS2000B` and some supporting functions. Each object of the class represents a physical PSU. These objects can be used to access status information and command the device, for example to set currents and voltages. More information can be found in the readme file provided by the original author.

The main modification to this library done as part of this thesis, was to implement independent commanding of both PSU channels. For this an additional parameter `channel` (integer type, 0 or 1) was added to all methods of the `PS2000B` class that interact with the device. Additionally the properties `PS2000B.output1`, `PS2000B.voltage1` and `PS2000B.current1` were duplicated to support the second channel (e.g. `PS2000B.output2`).

#### **Arduino Command API** `arduino.py`

To control the Arduino microcontroller the online library from [2] was integrated without major modifications. To use it, the provided `prototype.ino` file needs to be transferred to the Arduino. An object of class `Arduino` can then be used to control the board (connected via USB) with methods that closely resemble the ones used in the standard Arduino programming language. More details can be found in the readme file provided by the original author [2].

## 1.3. Conversion to Executable

The program is compiled to a `.exe` executable file to enable simple use without a Python installation. For this, the "Auto Py to Exe" tool developed by B. Vollebregt [4] is used. The resulting files are distributed in a separate repository on the IRS git server.<sup>2</sup> When a new software version is ready for publication, the following procedure should be used to release it:

1. Run "auto-py-to-exe.exe" (provided in main development repository)
2. In the Auto Py to Exe UI, select `main.py` file as "Script Location"
3. Select options "One File" and "Window Based"
4. Select file "Helmholtz.ico" as the "Icon"
5. Select "Releases" in the development repository as the output folder
6. Execute conversion
7. Rename resulting "main.exe" file to "Helmholtz Cage Control.exe"
8. Verify correct program operation from created executable
9. Copy new executable to the release repository<sup>2</sup> and replace previous version
10. Commit, merge and create a new release in the git web interface
11. Record changes in release notes and changelog and update the documentation

---

<sup>2</sup>[https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz\\_Test\\_Bench\\_Releases](https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz_Test_Bench_Releases)

## 2. Operating Instructions

### 2.1. Test Bench Assembly Instructions

Because of the limited space available in the IRS cleanroom, the test bench may need to be disassembled and reassembled in the future. Instructions for this are provided here. Mentioned position numbers relate to those shown in Figure 2.1.

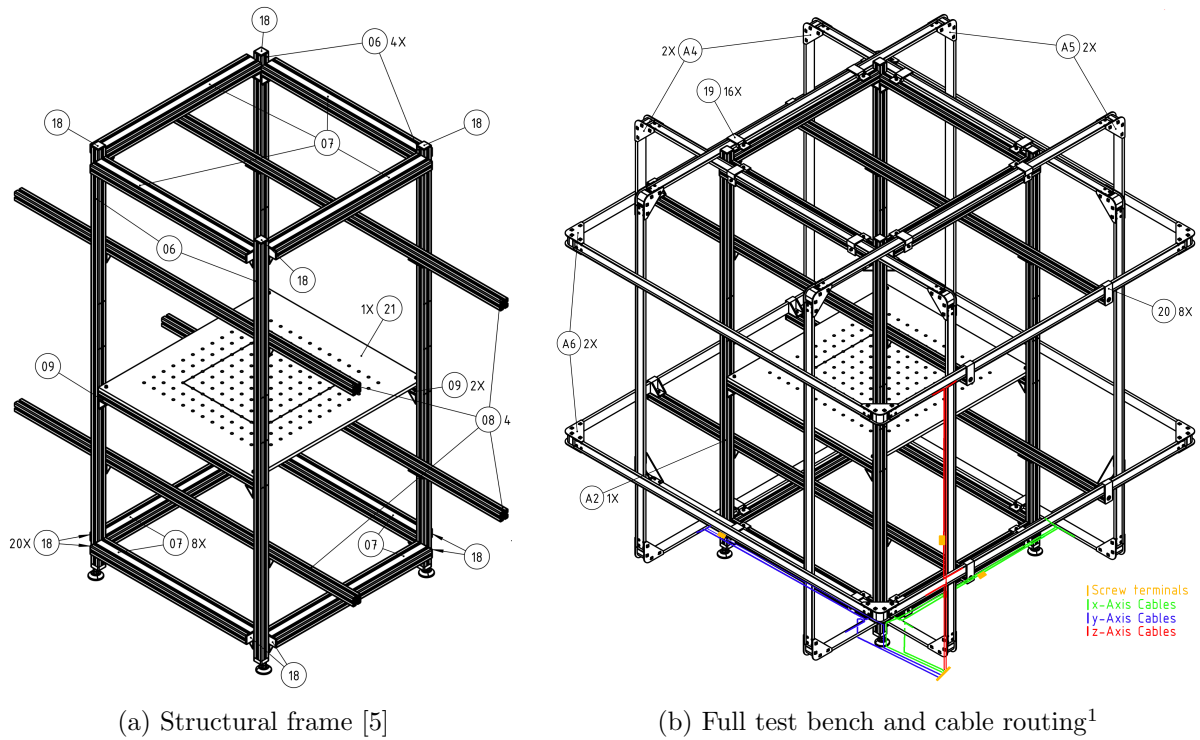


Figure 2.1.: Test bench assembly drawings

#### 2.1.1. Disassembly Procedure

##### Notes:

The coils must always be supported on two sides during handling to avoid bending.

The used slot nuts are wedged in the structural profiles and will stay in place even when their screw is removed. In most cases this is desirable, to mark the correct positions for reassembly. If one needs to be removed, loosen the attached screw and gently tap its head with a hammer.

**All removed screws must be stored orderly, as they are needed for future reassembly!**

---

<sup>1</sup>Base drawing from [5]



**Procedure:**

1. Undo cabling
  - a) Disconnect switch box from main cable and PSUs
  - b) Disconnect coil cables from central screw terminal (Fig. 2.2)
  - c) Disconnect screw terminals between coils, leave terminal on one side
  - d) Untie wires from their guiding structures
2. Remove Z-axis coils
  - a) Unscrew upper coil brackets (20) from profiles (08), but leave them on the coil
  - b) Lift off upper coil (A6)
  - c) Remove upper coil support profiles (08), leave angle pieces on main structure
  - d) Repeat procedure for lower coil
3. Remove Y and X-axis coils
  - a) Unscrew coil brackets (19) for one Y-axis coil (A5) from frame profiles (07), but leave them on the coil
  - b) Lift off coil (A5)
  - c) Repeat procedure for second coil (A5) and X-axis coils (A4)
4. Remove mounting plate (21)
5. Lay remaining frame on its side, mounting plate profiles (09) should be on top and bottom (orientation shown in Figure 2.3)
6. Separate "upper" rectangular frame section (highlighted yellow in Figure 2.3)
  - a) Remove bolts of angles connecting rectangle to cross-member profiles (07), angle pieces should remain on "upper" rectangle
  - b) Lift off entire rectangle section
7. Remove cross-member profiles (07) from "lower" rectangle section, angles stay attached

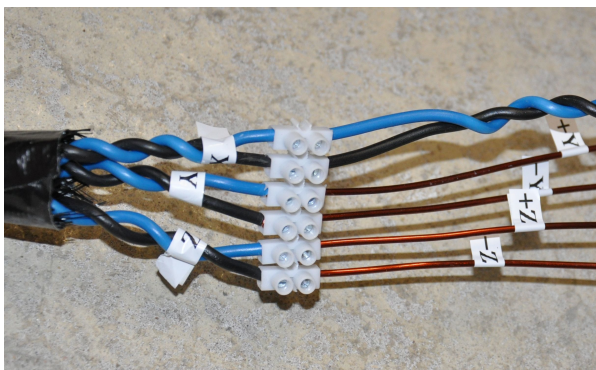


Figure 2.2.: Main connector (X-cables temp.)

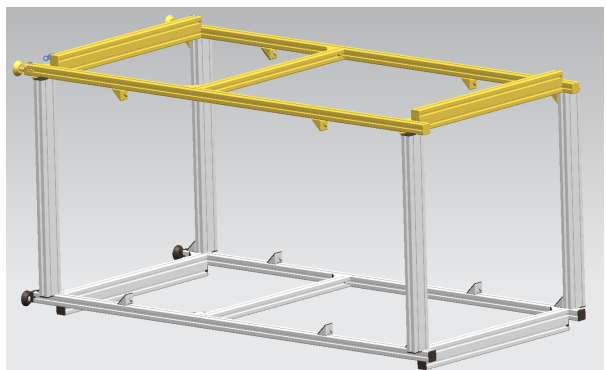


Figure 2.3.: Intermediate (dis-)assembly step

### 2.1.2. Reassembly Procedure

**Notes:**

These instructions assume a disassembly according to Section 2.1.1. Most slot nuts should still be in place and simplify positioning the parts. However, the angle pieces do allow some tolerance, so care should still be taken to get the correct alignments.

**Procedure:**

1. Lay one main frame rectangle on the floor as shown in Figure 2.3
2. Attach cross-member profiles (07)
3. Attach second frame rectangle on top, state should now be as shown in Figure 2.3.
4. Lift frame upright and adjust foot heights for solid stand
5. Attach mounting plate (21)
6. Install X and Y-axis coils
  - a) Lift +X coil (A4) onto cross-member profiles (07); mind wire exit position (Fig. 2.1b)
  - b) If needed, adjust cross-member profile height; coil should be supported equally on upper and lower profiles without bending
  - c) Centre coil and secure with 3D-printed brackets (19)
  - d) Repeat for -X (A4) and Y-axis coils (A5)
7. Install Z-axis coils (A6)
  - a) Attach lower coil support profiles (08)
  - b) Lift -Z coil over X/Y coils onto support profiles; mind wire exit position (Fig. 2.1b)
  - c) Centre coil and secure with 3D-printed brackets (20)
  - d) Repeat for upper support profiles and +Z coil
8. Make electrical connections
  - a) Connect screw terminals between coils (unmarked wire ends)
  - b) Route and tie down wires as shown in Figure 2.1b
  - c) Check that there are no short circuits between coil wires and frames
  - d) Connect main screw terminal as shown in Figure 2.2
  - e) Connect main cable to switch box
  - f) Verify contact on all axes (multimeter at switch box inputs), resistances approx.  $3.1\ \Omega$
  - g) Connect switch box to PSUs: X-axis to PSU 1 channel 1, Y-axis to PSU 1 channel 2, Z-axis to PSU 2 channel 1
9. Setup laptop and initialize control program as described in Section 2.2.3
10. Verify correct polarity on all axes through magnetic field measurements at different currents
11. Calibrate test bench (exact procedure to be developed)

## 2.2. Software Users Guide

### 2.2.1. Installation

1. Download latest release: [https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz\\_Test\\_Bench\\_Releases/releases](https://egit.irs.uni-stuttgart.de/zietzm/Helmholtz_Test_Bench_Releases/releases)
2. Unpack ZIP-folder and run "Helmholtz Cage Control.exe"
3. Setup hardware and program according to Section 2.2.3

### 2.2.2. User Interface Elements

The general UI layout is shown in Figure 1.2. The upper area contains the interactive elements of each operating mode and settings page. Switching between these pages is done using the top "Menu" bar. The lower half of the UI contains the status display and a console output.

#### Status Display

The status display shows the current state of the test bench devices. Explanations of the different values are given in Table 2.1.

Table 2.1.: Status display entries

Entry	Explanation
PSU Serial Port:	Serial COM port used to connect to the PSU for this axis.
PSU Channel:	PSU output channel for this axis (0→channel 1, 1→channel 2).
PSU Status:	Connection status of PSU for this axis. Possible states: <ul style="list-style-type: none"> <li>• <i>Connected</i>: communication nominal</li> <li>• <i>Not Connected</i>: PSU not found during initialization</li> <li>• <i>Connection Error</i>: PSU was connected but then an error occurred, e.g. it was disconnected</li> </ul>
Arduino Status:	Connection status of switch box Arduino (identical for all axes). Possible states: <ul style="list-style-type: none"> <li>• <i>Connected</i>: communication nominal</li> <li>• <i>Not Connected</i>: Arduino not found during initialization</li> <li>• <i>Connection Error</i>: Arduino was connected but then an error occurred, e.g. it was disconnected</li> </ul>
Output:	Status of PSU output channel. Possible states: <ul style="list-style-type: none"> <li>• <i>Active</i>: set current or voltage is applied to the output jacks</li> <li>• <i>Inactive</i>: output jacks are unpowered</li> <li>• <i>Unknown</i>: no connection to PSU</li> </ul>
Remote Control:	Status of PSU channel remote interface. Possible states: <ul style="list-style-type: none"> <li>• <i>Active</i>: channel can be controlled remotely</li> <li>• <i>Inactive</i>: channel can not be controlled remotely</li> <li>• <i>Unknown</i>: no connection to PSU</li> </ul>
Voltage Setpoint:	Maximum voltage PSU is set to supply.

*Continued on next page*

Table 2.1 – Continued from previous page

Entry	Explanation
Actual Voltage:	Voltage across PSU channel output jacks. Usually lower than "Voltage Setpoint", as voltage is throttled to achieve the desired current.
Current Setpoint:	Current the PSU output channel is set to supply
Actual Current:	Current flowing through the PSU output channel.
Target Field:	Desired magnetic flux density in the measurement area.
Trgt. Field Raw:	Flux density used to calculate needed current (after ambient field compensation).
Target Current:	Desired current to flow through the coils. Negative values mean reversed polarity.
Inverted:	Status of polarity change relay for this axis inside the switch box. Possible states: <ul style="list-style-type: none"> <li>• <i>True</i>: (Arduino pin "HIGH"→relay switched→polarity inverted, status light-emitting diode (LED) should be illuminated)</li> <li>• <i>False</i>: pin "LOW"→opposite of "True" state</li> <li>• <i>Unknown</i>: no connection to Arduino</li> </ul>

## Manual Mode

The manual input mode is used to set static currents or magnetic fields on the test bench. Its layout is shown in Figure 2.4. The main UI elements are listed below.

Figure 2.4.: Manual input mode user interface

- **"Select Input Mode" drop-down:** Switches between setting currents or magnetic fields
- **Value entry fields:** Enter values to be set on the test bench here
  - Entries must be numeric (decimal point)
  - Entries must be in indicated safe range (may be changed in the settings page)

- **"Compensate ambient field" checkbox:** When ticked, the ambient magnetic field (set in settings page) is subtracted from entered values before commanding the test bench (inactive for "Current" input mode)
- **"Execute" button:** Implements values from the entry fields
  - Commands currents on PSUs and polarity on switch box
  - If a device is not connected, the remaining ones are still commanded
  - Values beyond safe limits (indicated to the right of entry fields) are rejected
  - Device status and any errors are displayed in status display and console
- **"Power Down All" button:** "Panic button", sets currents on both PSUs to 0, deactivates outputs and sets switch box relay pins to inactive state
- **"Reinitialize" button:** Reruns program initialization
  - Reinitializes connection to PSUs and switch box Arduino
  - Press after (re)connecting a device to let program establish communications

**To command a field vector or currents on the test bench:**

1. Select needed input mode, using the drop-down menu
2. Enter desired values in entry fields
3. For magnetic fields, choose whether ambient field should be compensated by (un)tickling the checkbox
4. Press "Execute" button, devices will now implement set values
5. Check console output to see if any errors occurred
6. Monitor behaviour in status display and on devices
7. When finished, press "Power Down All" button to remove currents from the test bench

**CSV Sequence Execution Mode**

This mode is used to run timed sequences of magnetic fields. These have to be defined in a CSV file of the following form:

- *Column separator:* Semicolon (;)
- *Decimal:* Comma (,)
- *Line terminator:* Tested with Windows standard (`\r\n`), other options may work as well
- *Columns:* Time in seconds; X-axis, Y-axis and Z-axis flux density in Tesla

An example for the CSV file structure is given below:

```
Time (s);xField (T);yField (T);zField (T)
0,5;0,000015;0,000025;0,00002
1;0,0000155;0,0000245;0,0000205
```

When loading such a file, the application will check that there are no values exceeding the safe test bench limits in the sequence. If any are found, a warning message is shown and the limits displayed in the sequence graph. It is still possible to execute such a sequence, however the excessive values will not be commanded. For those time stamps, 0 A is set instead. The limits may be adjusted in the program settings page, but care should be taken to avoid equipment damage.

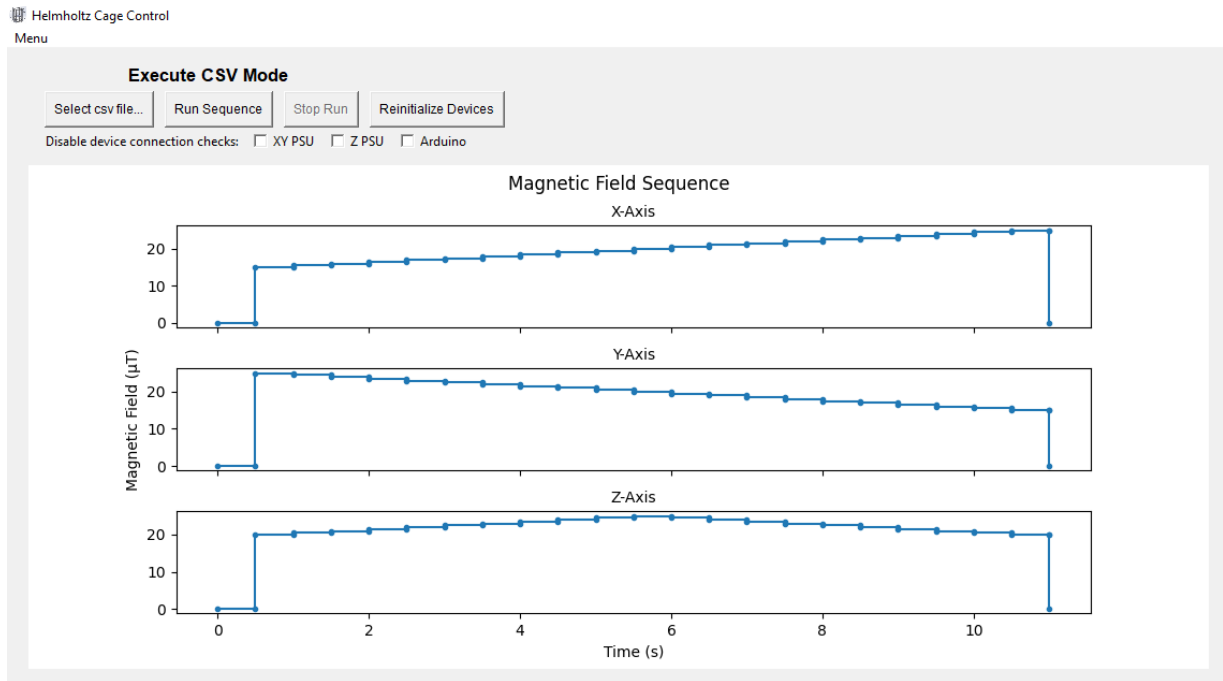


Figure 2.5.: CSV sequence mode user interface

The UI layout is shown in Figure 2.5, its main elements are listed below.

- **”Select csv file...” button:** Opens a file dialog to let user choose a CSV file to execute
- **”Run Sequence” button** Starts executing the sequence from the chosen CSV file
- **”Stop Run” button:** Aborts sequence execution
- **”Reinitialize devices” button:** Reruns program initialization
  - Reinitializes connection to PSUs and switch box Arduino
  - Press after (re)connecting a device to let program establish communications
- **”Disable device connection checks” checkboxes:** Disable connection check for specific devices, allowing sequence execution with some equipment missing
- **Graph display:** Displays graph of field sequence, as it will be executed by the test bench

**To execute a field sequence:**

1. Ensure correct configuration of program and test bench in the settings page
2. Prepare sequence in CSV file with correct format (external program)
3. Press "Select csv file..." button, select prepared file and open
4. Check plots to confirm sequence was loaded correctly and no values exceed safe limits
5. If not all devices are used, check boxes of unused devices to disable connection check
6. Press "Run Sequence" button
7. Monitor execution in console, status display and on devices

**Data Logging Configuration Page**

The application has the ability to log test bench data to a CSV file. The data is temporarily stored internally and must be saved to an external file by user request.

An example of a log file is given in Appendix A. The first three columns are time stamps: date, system time and time since the start of logging in seconds. The other columns contain the data, as selected by the user. All dynamic values from the status display can be logged, see Table 2.1 for explanations. Each type of data is logged for all three axes. The units for numerical values are Volt, Ampere and Tesla.

There are two options as for when and how often data is logged. The first option logs a row of data in a regular time interval specified by the user. The second option logs, whenever a significant command is sent to the test bench, for example when a new field vector is commanded. Both options can be used simultaneously.

The logging configuration UI is shown in Figure 2.6. Its elements are listed below.

- **"Start Logging" button:** Starts the logging of data, as configured in the other elements
- **"Stop Logging" button:** Stops the logging of data
- **"Write data to file" button:** Lets user save the logged data to a file
  - Opens dialogue to let user select a file path (chosen file name must be \*.csv)
  - Saves logged data to the selected file
- **"Clear logged data" button:** Deletes all data logged internally by the program, user will be asked if data should be saved to a file first
- **"Datapoints logged" counter:** Displays the current number of logged data rows
- **"Log in regular intervals" controls:** Enable checkbox to periodically log data, set interval (in seconds) in the entry field to the right
- **"Log whenever test bench is commanded" checkbox:** Enable, to log data on significant changes to the test bench (e.g. a new field vector is commanded)
- **Data selection checkboxes:** Select what data to log, explanations are given in Table 2.1

Helmholtz Cage Control  
Menu

**Configure Data Logging**

Start Logging Stop Logging Write data to file Clear logged data

Datapoints logged: 8

☒ Log in regular intervals Interval (s): 1

☒ Log whenever test bench is commanded

Select which data to log:

☒ PSU Status

☒ Voltage Setpoint

☒ Actual Voltage

☒ Current Setpoint

☒ Actual Current

☒ Target Field

☒ Trgt. Field Raw

☒ Target Current

☒ Inverted

Figure 2.6.: Data logging configuration page

**To collect and save log data:**

1. Select when to log data (both options can be used simultaneously)
  - For regular logging, enable "Log in regular intervals" checkbox and set desired interval
  - For logging on test bench command, enable second checkbox
2. Select what data to log, using the lower checkboxes
3. Press "Start Logging" button
4. When finished, press "Stop Logging" button
5. Press "Write data to file" button
6. Choose file location and name (must be \*.csv, e.g. my\_log.csv), and press save
7. Press "Clear logged data" button (optional)



## Settings Page

This is the main page for configuring the program. The settings can be stored to and loaded from \*.ini configuration files.

The different program constants are set through a series of entry fields. All inputs must be numerical values (decimal points). Safe limits for each value are set inside the program. The constants and limits are listed in Table 2.2. If a value exceeding those boundaries is entered, a warning message is displayed and the respective entry field marked in red. **It is not recommended to ignore these warnings, as incorrect settings can damage equipment on the test bench!**

	X-Axis	Y-Axis	Z-Axis	
Coil Constants:	38.957	40.408	37.754	$\mu\text{T/A}$ Field generated per applied current
Ambient Field:	0.0	0.0	0.0	$\mu\text{T}$ Field to be compensated
Resistances:	3.131	3.107	3.129	$\Omega$ Resistance of coils + equipment
Max. Current:	5.0	5.0	5.0	A Max. allowed current
Max. Voltage:	15.0	15.0	15.0	V Max. allowed voltage, must not exceed 16V!
Arduino Pins:	15	16	17	- Should be 15, 16, 17

Figure 2.7.: Program settings page

Figure 2.7 shows a screenshot of the UI layout with the default settings. The main elements are listed below.

- **“Load config file...” button:** Imports an existing configuration file
  - Opens file dialogue for selecting configuration file
  - Loads settings from selected file
  - Checks if any settings exceed safe limits and, if so, displays warning messages
  - Reinitializes test bench devices with new settings
- **“Save current config” button:** Writes settings to the currently selected file and reinitializes test bench devices with new settings

- **”Save current config as...” button:** Writes settings to a new file
  - Opens dialogue to let user choose new file path and name (must be \*.ini)
  - Reinitializes test bench devices with current settings
- **”PSU Serial Port” entries:** Input COM ports for both PSUs here
  - Use Windows device manager to find correct port names (connect PSUs separately to differentiate between devices)
  - Test bench X- and Y-axes need to be connected to channel 1 and 2 of one PSU, Z-axis to channel 1 of the other
  - *Note: Switch box Arduino should be found automatically*
- **Program constant entry fields:** Set constants here, details are listed in Table 2.2
- **”Update and Reinitialize” button:** Implements any changed settings in the program and on test bench devices, needs to be pressed for changes to take effect
- **”Restore Defaults” button:** Restores default settings

Table 2.2.: Settable program constants

Entry	Limits	Explanation
Coil Constants:	0 to 50 $\frac{\mu\text{T}}{\text{A}}$	Magnetic field generated per applied current <ul style="list-style-type: none"> <li>• Used to calculate current needed to achieve desired field</li> <li>• Must be measured and tuned before test campaigns</li> </ul>
Ambient Field:	-200 to 200 $\mu\text{T}$	Background magnetic field in the measurement area <ul style="list-style-type: none"> <li>• Subtracted from desired field to compensate ambient field</li> <li>• Must be measured and tuned before test campaigns</li> </ul>
Resistances:	1 to 5 $\Omega$	Electrical resistance of each axis, measure from PSU connectors
Max. Current:	0 to 6 A	Current limit for each axis <ul style="list-style-type: none"> <li>• Program will block commanding of higher values on PSU</li> <li>• Maximum safe permanent current: <math>I_{max} = 5.5 \text{ A}</math> [5]</li> </ul>
Max. Voltage:	0 to 16 V	Voltage limit for each axis <ul style="list-style-type: none"> <li>• Program will block commanding of higher values on PSU</li> <li>• Maximum safe voltage, limited by diodes inside switch box: <math>U_{max} = 16 \text{ V}</math></li> </ul>
Arduino Pins:	15,16,17	Output pins on switch box Arduino for inverting polarity on each axis (hard wired inside switch box, should not need to be changed)

### 2.2.3. Hardware Connections and Program Setup

1. Connect hardware
  - a) Connect switch box to test bench main cable bundle (connector on the back)
  - b) Connect switch box to PSUs: **X-axis to PSU 1 channel 1, Y-axis to PSU 1 channel 2, Z-axis to PSU 2 channel 1**
  - c) Connect switch box power supply (12 V DC)
  - d) Connect switch box USB port to PC
  - e) Connect PSU USB ports to PC
2. Configure interfaces to hardware
  - a) Start program (run "Helmholtz Cage Control.exe")
  - b) Go to settings page (Menu→Settings...)
    - If program has not been configured before:
      - c) Use Windows device manager to find correct serial COM ports for PSUs (connect /disconnect in turn to differentiate between devices)
      - d) Enter COM port names in application (switch box should be found automatically)
      - e) Press "Update and Reinitialize" button
    - If a previous configuration exists:
      - c) Press "Load config file..." button
      - d) Select configuration file (e.g. "myconfig.ini") and open
      - e) Check that the settings were loaded correctly and no values violate the safe limits (fields highlighted in red)
  - f) Check console print to see if all devices were found, otherwise check physical connections and COM port settings
3. Test configuration
  - a) Go to manual mode (Menu→Static Manual Input)
  - b) Switch input mode to "Current" (see Section 2.2.2)
  - c) Set different currents and check device response:
    - Current should be activated on correct PSU channel
    - For negative currents, corresponding status LED on switch box should light up and relay actuation be audible as clicking sound
4. Go back to the settings page (Menu→Settings...)
5. Change program constants as needed (e.g. enter measured ambient field), see Section 2.2.2
6. Save configuration:
  - Press "Save current config" button to update the current configuration file
  - Press "Save current config as..." button to save to a new configuration file, set new file name in file dialogue (e.g. "myconfig.ini")

# Bibliography

- [1] SPRÖSSIG, Sören. *Python PS2000B Library* [online] [visited on 2020-11-18]. Available from: <https://github.com/ssproessig/Python-PS2000B>.
- [2] *Arduino Python 3 Command API* [online] [visited on 2020-12-09]. Available from: <https://github.com/mkals/Arduino-Python3-Command-API>.
- [3] KINSLEY, Harrison. *Object Oriented Programming Crash Course with Tkinter* [online] [visited on 2021-01-19]. Available from: <https://pythonprogramming.net/object-oriented-programming-crash-course-tkinter/>.
- [4] VOLLEBREGT, Brent. *Auto Py to Exe* [online] [visited on 2020-03-02]. Available from: <https://github.com/brentvollebregt/auto-py-to-exe>.
- [5] BLESSING, Steffen. *Design of a CubeSat Magnetic Field Cage for the Verification of CubeSats Attitude Control Systems*. Stuttgart, 2020. Institute for Space Systems (IRS), University of Stuttgart.

## A. Example Program Auxiliary Files

### Example Configuration File

```
[X-Axis]
coil_const = 3.8957e-05
ambient_field = 0.0
resistance = 3.131
max_volts = 15.0
max_amps = 5.0
relay_pin = 15

[Y-Axis]
coil_const = 4.0408000000000003e-05
ambient_field = 0.0
resistance = 3.107
max_volts = 15.0
max_amps = 5.0
relay_pin = 16

[Z-Axis]
coil_const = 3.7754e-05
ambient_field = 0.0
resistance = 3.129
max_volts = 15.0
max_amps = 5.0
relay_pin = 17

[PORTS]
xy_port = COM1
z_port = COM2
```

### Example Log File

```
Date;Time;t (s);X Target Field;Y Target Field;Z Target Field
2021-03-08;16:18:34;583986;0,0;0,0;0,0,0
2021-03-08;16:18:39;595109;5,011123;0,0;0,0,0
2021-03-08;16:18:44;217410;9,633424;-4,499999996e-05;-4,999999996e-05;0,0
2021-03-08;16:18:44;597378;10,013392;-4,499999996e-05;-4,999999996e-05;0,0
2021-03-08;16:18:49;604153;15,020167;-4,499999996e-05;-4,999999996e-05;0,0
2021-03-08;16:18:54;619011;20,035025;-4,499999996e-05;-4,999999996e-05;0,0
2021-03-08;16:18:57;713520;23,129534;-4,499999996e-05;4,999999996e-05;0,0
2021-03-08;16:18:59;635428;25,051442;-4,499999996e-05;4,999999996e-05;0,0
2021-03-08;16:19:00;759706;26,17572;0,0;0,0,0
```