

ATMOSNIFFER DESKTOP APPLICATION

By

Miguel Castrejon

A project report

Submitted to the faculty of the

MSCS Graduate Program of Weber State University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

October 10, 2021

Ogden, Utah

Approved:

Date:

---

Hugo Valle, PhD. Committee chair.

---

Yong Zhang, PhD. Committee member

---

Abdulmalek Al-Gahmi, PhD. Committee member

## TABLE OF CONTENTS

	Page
<b>LIST OF FIGURES</b> . . . . .	<b>iii</b>
 <b>1 Atmosniffer Overview</b> . . . . .	 <b>1</b>
1.1 Motivation . . . . .	1
1.1.1 Why build a device that can "Sniff" the atmosphere? . . . . .	1
1.1.2 AtmoSniffer Project . . . . .	1
1.1.3 AtmoSniffer Device Pictures . . . . .	2
1.1.4 Project Interaces . . . . .	3
 <b>2 MSCS Project Proposal</b> . . . . .	 <b>5</b>
2.1 Purpose . . . . .	5
2.2 Approach . . . . .	5
2.3 Research . . . . .	5
2.4 Criteria . . . . .	6
 <b>3 AtmoSniffer Desktop Application</b> . . . . .	 <b>7</b>
3.1 Old Desktop Application . . . . .	7
3.2 New Desktop Application . . . . .	7
 <b>4 AtmoSniffer User Manual</b> . . . . .	 <b>8</b>
4.1 Users . . . . .	8
4.1.1 Installation . . . . .	8
4.1.2 Opening a file . . . . .	8
4.1.3 Adding a graph to an open file . . . . .	10
4.1.4 Closing graphs . . . . .	11
4.1.5 Connecting to an AtmoSniffer Device . . . . .	11
4.1.6 Sending commands to connected device . . . . .	13
4.1.7 Closing a connected device . . . . .	15
4.1.8 Find communication port . . . . .	15
4.2 Developers . . . . .	19
4.2.1 Installation . . . . .	19
4.2.2 Creating the installer . . . . .	20
4.2.3 Adding more commands to device command dropdown . . . . .	26
4.2.4 Add more headers for upcoming versions . . . . .	28
4.2.5 Run Tests . . . . .	28
 <b>Appendices</b> . . . . .	 <b>30</b>
 <b>A Python Libraries used to develop the application</b> . . . . .	 <b>31</b>

## LIST OF FIGURES

Figure	Page
1.1 AtmoSniffer Device Encased, this is how the whole device looks when its assembled. [1]	2
1.2 Inside AtmoSniffer Device [1]) . . . . .	2
1.3 This is the Gas Sensors Module, this can be taken on/off like any other module.[1] . . . .	3
1.4 Device Modules [1]) . . . . .	3
4.1 Main desktop window . . . . .	8
4.2 Open file dialog . . . . .	9
4.3 Choosing ATM02.CSV from open file dialog . . . . .	9
4.4 ATM02.CSV file opened . . . . .	10
4.5 Graph Dialog . . . . .	10
4.6 Graph Dialog Selections . . . . .	11
4.7 Opened graph on a static device, csv file . . . . .	11
4.8 Close a graph . . . . .	11
4.9 Communication Ports Dialog . . . . .	12
4.10 Communication port device opened . . . . .	12
4.11 Error dialog for communication port . . . . .	13
4.12 Manual command dialog . . . . .	14
4.13 Turning off the ac module through a manual command . . . . .	14
4.14 Dynamic device, dropdown menu . . . . .	15
4.15 Dropdown menu command options . . . . .	15
4.16 Close dynamic device . . . . .	15
4.17 Bluetooth settings . . . . .	16
4.18 More bluetooth settings . . . . .	16
4.19 Com ports list . . . . .	17
4.20 Device Manager . . . . .	17
4.21 Device Manager Com Ports . . . . .	18
4.22 Installing requirements with pip . . . . .	19
4.23 Requirements satisfied . . . . .	19
4.24 Pip freeze . . . . .	19
4.25 Setup command . . . . .	20
4.26 Build folder . . . . .	20
4.27 Required files for executable to work . . . . .	20
4.28 Inside build folder with required files . . . . .	21
4.29 Using Inno Setup to make an installer . . . . .	21
4.30 Information concerning our application . . . . .	22
4.31 Adding files to the installer . . . . .	22
4.32 Selected files in Inno Setup . . . . .	23
4.33 Adding folders to inno setup . . . . .	23
4.34 Selecting the exe file . . . . .	24
4.35 Choosing compiler inno setup . . . . .	24
4.36 Inno Setup Script Example . . . . .	25
4.37 Giving users the required permissions to modify files in system folders . . . . .	25
4.38 Setting up placeholders for icons. . . . .	25
4.39 Editing script line to correctly use icons. . . . .	26
4.40 Atmosniffer setup installer. . . . .	26
4.41 Submenu cmd list. . . . .	26
4.42 First if block in the command making process. . . . .	27
4.43 Battery volts command example. . . . .	28
4.44 Version parsing. . . . .	28
4.45 vscode Atmosniffer. . . . .	29

4.46	vscode Atmosniffer run tests. . . . .	29
------	---------------------------------------	----

## CHAPTER 1

### Atmosniffer Overview

This Chapter provides an overview of the device the research team at Weber State University, led by Dr. Sohl and Dr. Valle started working on, the AtmoSniffer.

#### 1.1 Motivation

##### 1.1.1 Why build a device that can "Sniff" the atmosphere?

- "An estimated 4.2 million premature deaths globally are linked to ambient air pollution" according to the World Health Organization [2]. In other words, the equivalent to the entire population of Utah every 9 months dies prematurely because of bad air. WHO estimates that 91% of the world's population is living in places where air quality guidelines are not met.[2] Measuring this pollution is the first step in understanding and solving the problem.

- **Problem:**

- Current air monitoring systems can be very large, heavy, and/or expensive
- They possess a limited range of sensors typically measuring only one gas type
- Are only usable for a specific task or deployment
- Require significant AC power.

- **Solution:**

- The AtmoSniffer is small and light.
- Detects a variety of gas types.
- Multiple user interfaces that can be easily accessed.

##### 1.1.2 AtmoSniffer Project

- The research team at Weber State University, led by Dr. Sohl and Dr. Valle started working on a device which is capable of moving air particles through gas sensors and transmit its data through radio and Bluetooth signals to monitor air quality and measure pollution. [1]
- The current AtmoSniffer prototype, version 0.3, detects CO, CO<sub>2</sub>, NO, NO<sub>2</sub>, SO<sub>2</sub>, NH<sub>3</sub>, O<sub>3</sub>, VOCs, fine particulate material (in six size ranges covering 0.3 μm to 10 μm diameter aerosols), temperature,

pressure, relative humidity, and inertial/true position. As a complete environmental monitor, turbulence/vibration is detected through a 9-axis inertial sensor. [1]

- The AtmoSniffer can be deployed as a portable emissions locator or as a long duration air quality monitor staged on a work bench or mounted to pretty much anything.

### 1.1.3 AtmoSniffer Device Pictures

Here are a couple of pictures showing you how the physical device looks like:

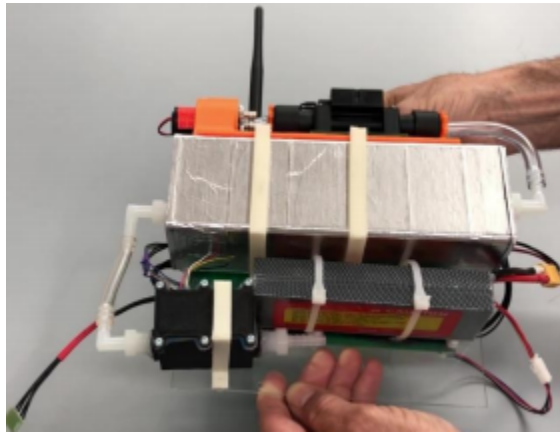


Figure 1.1: AtmoSniffer Device Encased, this is how the whole device looks when its assembled. [1]



Figure 1.2: Inside AtmoSniffer Device [1])



Figure 1.3: This is the Gas Sensors Module, this can be taken on/off like any other module.[1]

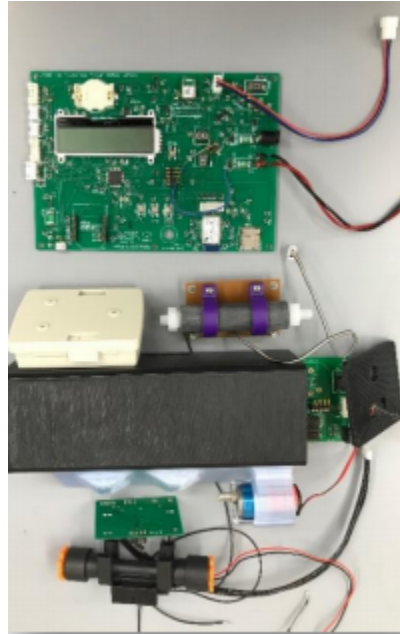


Figure 1.4: Device Modules [1])

#### 1.1.4 Project Interaces

- The device transmits a stream of data via the following communication protocols:
  - Xbee radio via UART channel; Fully functional
  - Bluetooth via UART channel: Fully functional
  - WIFI via UART Channel: To be develop
- Supports two Atmosniffer interfaces
  - Android App
    - \* Quick Diagnostic
    - \* Data Calibration
  - Desktop Application

- \* Full Data Analysis
- \* Data Calibration



## **CHAPTER 2**

### **MSCS Project Proposal**

This chapter provides the purpose, approach, research, and criteria on the approved project proposal.

#### **2.1 Purpose**

The main purpose of this application was to make a more user friendly AtmoSniffer Device Interface that could be easily accessed by both, developers and users. Moreover, the main audience for the application is the research team at WSU. For them to be able to visualize data sent by a device, in a graph format, and to communicate and control the device for calibration purposes.

Other than building an application that people can easily use, I wanted to learn python, a new framework, such as qt, and how to professionally document a project.

#### **2.2 Approach**

To develop the application, I went through each step of the Software Development Life Cycle. With the help and permission of Dr. Valle, I created a GitHub repository to hold a copy of the old desktop application and start building from there. At first, I tried continuing and building up from the old desktop application, but I rather quickly found out that the old project lacked proper documentation and it was a much better idea to start from scratch rather than spend a whole month trying to decyphering it. I still kept the old project there under a legacy folder as a reference.

I was able to create a GUI and test it only with the help of python libraries. I met with users throughout and scheduled sprints to develop features.

#### **2.3 Research**

Research was focused on learning a QT, a new framework, and python. I used pyqt5 library which is the binding between qt and python.

The application needed to be able to handle a lot of data quickly. Python came with a big number of libraries used for data visualization, analysis, and parsing. This is the reason we chose it, but also because I have always wanted to learn it.

One more reason I found to do this project was for me to learn and experience a professional approach to software development. Including clean code, documentation, life cycle, testing, and a lot of messing up and debugging.

The optimal goal of this project is to visualize a large amount of data.

## 2.4 Criteria

Application goals and milestones

- Read data from SD card
- Visualize data from SD files
- Support bluetooth and radio connectivity.
  - Connec to device
  - Receive data from a device through bluetooth and/or radio signals
- Visualize data from broadcast records.
  - Create and close graphs
  - Give the user the option to choose which type of graph they would like to see.
- Send commands to the connected device through bluetooth and/or radio signals
  - Turn ON/OFF device components
  - Enable extra debugging
  - Reset sensors
- Documentation
  - Full documentation of code, following best standards.
  - Documentation on how to use the application
- Unit Testing
  - This includes code and gui components of the application
- Support Multiple OS
  - Create executables/installers for various OS

## CHAPTER 3

### AtmoSniffer Desktop Application

This chapter provides an overview on the main differences between the old and new desktop application

#### 3.1 Old Desktop Application

- The old "Desktop app" was very basic application, only used for debugging purposes.
  - It connected to a single device
  - Parsed data stream from radio signals
  - Basic plotting of data members
  - Basic I/O with device
  - It was only able to be ran through a set of options from a terminal, no gui interface.

#### 3.2 New Desktop Application

- The proposed "Desktop app" was very basic application, only used for debugging purposes.
  - Its able to connect and listen to multiple devices
  - Parses data stream from both radio and bluetooth signals
  - More advance plotting of data members, with the option to run different types of graphs.
  - More advance and user friendly I/O with connected devices, using dropdown menu and textboxes.
  - GUI interface for easier user navigation.

## CHAPTER 4

### AtmoSniffer User Manual

Hello and welcome to the Desktop application for the AtmoSniffer! Here you will find all you need to know to install, run, and even develop the application.

With this application you are able to connect to an atmosniffer device, open static report files, and compare data with the help of graphs. You are also able to communicate with the connected devices through commands. This is a python project, so please familiarize yourself with python first.

#### 4.1 Users

Let's first start with a quick guide for every day users!

##### 4.1.1 Installation

For the every day user we have a very simple and easy to use installer, here is the link [\[1\]](#), download and follow all instructions inside.

##### 4.1.2 Opening a file

So you want to look at old data and compare it with other device's or even the same device's data? Well you can do it by first making sure you have the desktop application running.

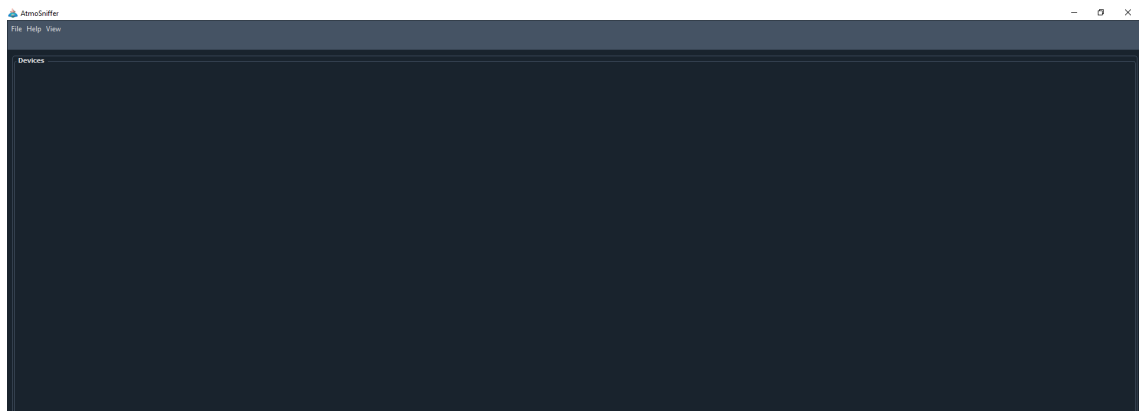


Figure 4.1: Main desktop window

Use the menu item File on the left-top corner of the screen and then click Open. This should bring up a dialog for you to find whichever data file you would like to look at.

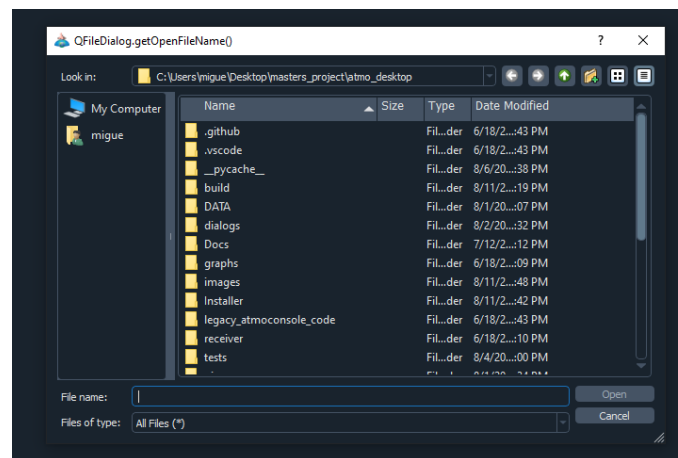


Figure 4.2: Open file dialog

!You can also use the shortcut CTRL+O!

After the dialog has popped up, navigate into the DATA folder and select your desired file, click okay and you should hopefully have a new device added into the application for analysis.

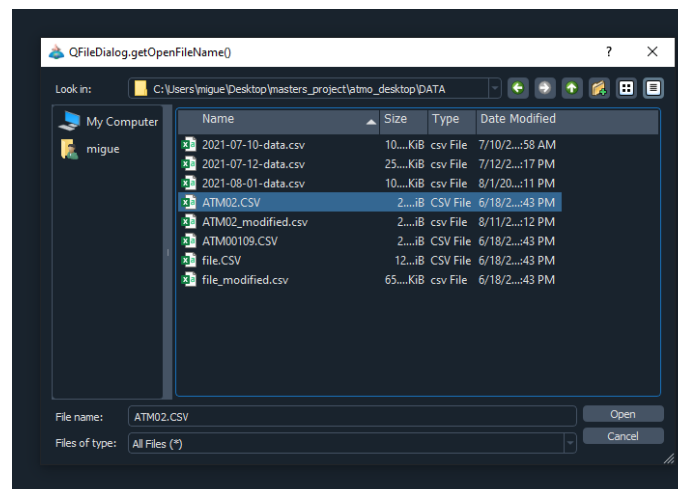


Figure 4.3: Choosing ATM02.CSV from open file dialog

In this case we are using the ATM02.CSV file.



Figure 4.4: ATM02.CSV file opened

### 4.1.3 Adding a graph to an open file

Now that you have successfully opened a device, let's try to add a graph to the main window and start analysing. First you will need to click the Add Graph button on the right side of the device and this next dialog should pop up.

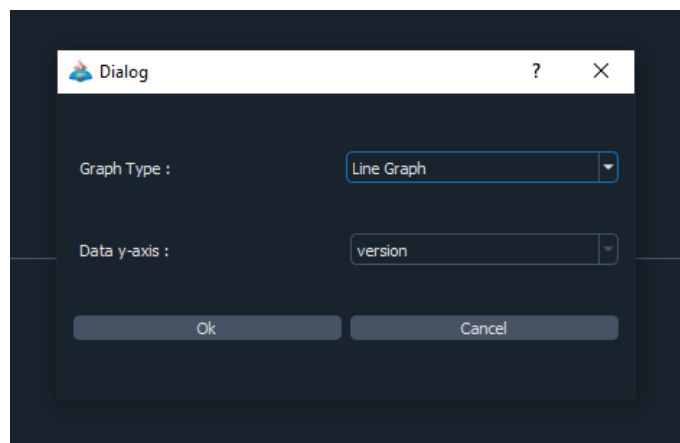


Figure 4.5: Graph Dialog

From here, go ahead and select which type of graph you would like to see and then continue to select which x-axis/y-axis data item you would like to see. Please keep in mind that line graph and bar graph only need the x - axis item, but the scatter plot needs both. Once you have selected your desired graph type and items go ahead and click 'Ok'

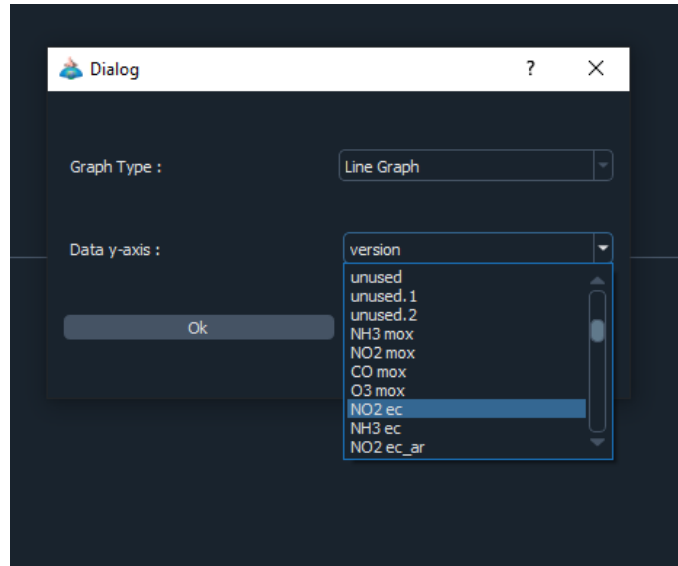


Figure 4.6: Graph Dialog Selections

This should add a new graph tab to our device window.

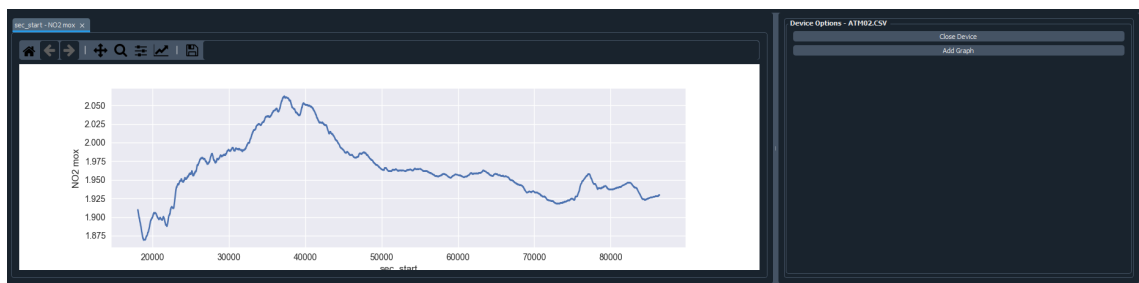


Figure 4.7: Opened graph on a static device, csv file

#### 4.1.4 Closing graphs

If you would like to close an opened graph, you just need to click on the small 'x' by the selected graph's tab.

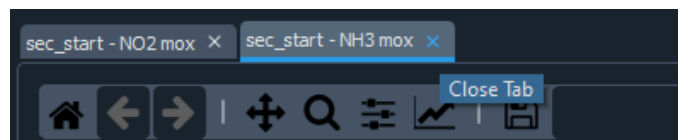


Figure 4.8: Close a graph

#### 4.1.5 Connecting to an AtmoSniffer Device

If you would like to watch the data being transmitted from an AtmoSniffer device then all you have to do is go to the main window, go to the top left menu 'File', and then click Connect Device. This should bring up

a little dialog window asking you to choose from the currently connected devices in your computer. This can be done through bluetooth or radio for now. You can also use the shortcut CTRL+O!

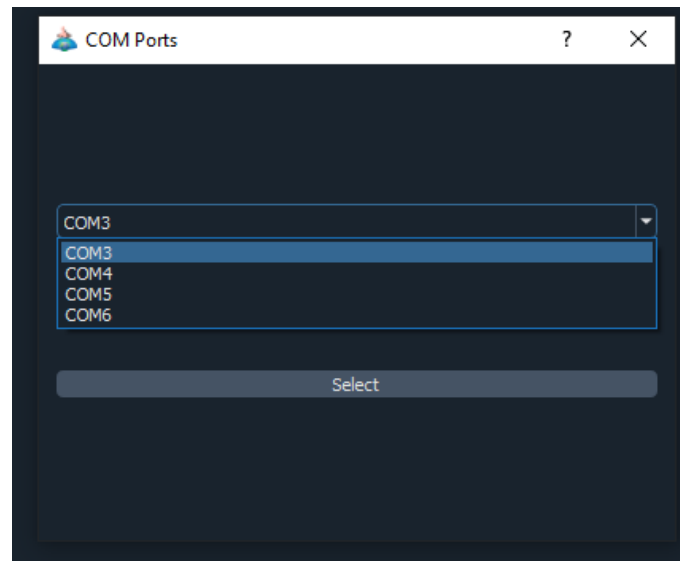


Figure 4.9: Communication Ports Dialog

The image above shows the current connected devices on my computer, by going into the communication ports settings on my computer I know that I should be connecting to COM6 since that's what corresponds to the actual AtmoSniffer Device. I'll choose it from the dropdown menu and then proceed to click Select. If everything goes right we should get an additional outline added to our main window with the name of its corresponding communication port.



Figure 4.10: Communication port device opened

Caution! Might sometimes not work as expected because of signals being sent through the 'air'



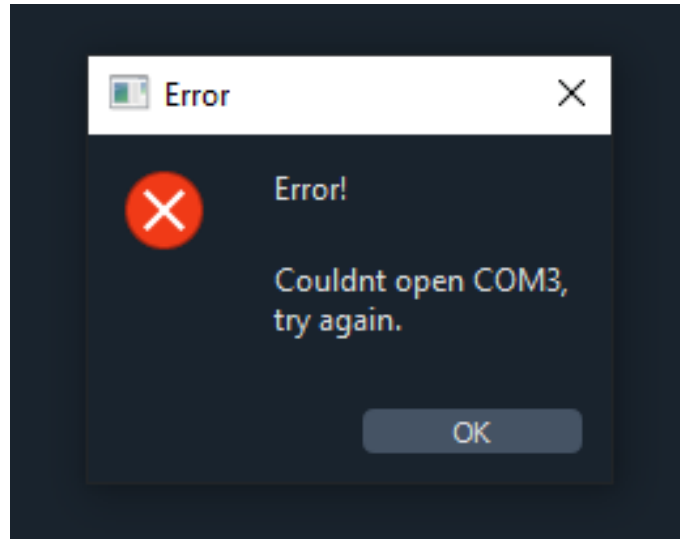


Figure 4.11: Error dialog for communication port

If this happens, it can mean one of two things. It could be that you are trying to connect to a communication port that does not correspond to an AtmoSniffer Device or it just had a small hiccup and you should just try again.

#### **4.1.6 Sending commands to connected device**

Before you are able to send any commands, please make sure you have successfully connected to a device first and then follow either one of the next list items. By clicking on 'View' -> 'Terminal' a terminal window will be added to the bottom of the screen and you will be able to see the commands being sent to the connected device.

- **Manual:**

To be able to send a command manually, click on the device's 'Manual Command' button, this should pop up a window.

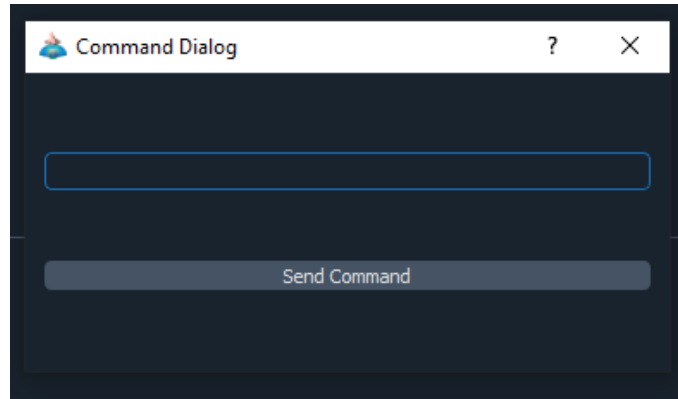


Figure 4.12: Manual command dialog

Now go ahead and type in your command and then proceed to press 'Send Command', in this case I will be sending 'ac 0', which should turn it off.

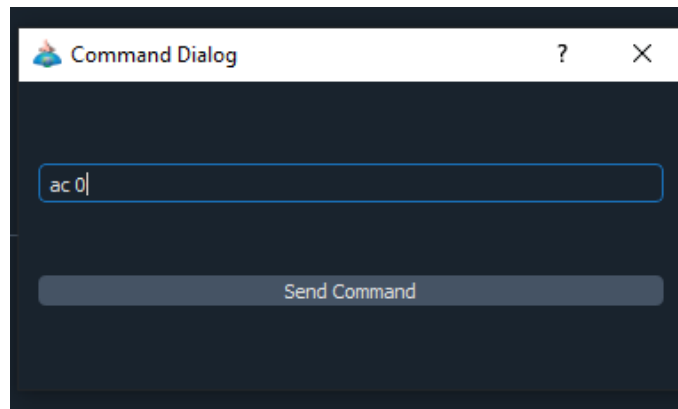


Figure 4.13: Turning off the ac module through a manual command

- Dropdown Menu:

Using the dropdown menu can sometimes speed things up since you won't be needing to type anything down.

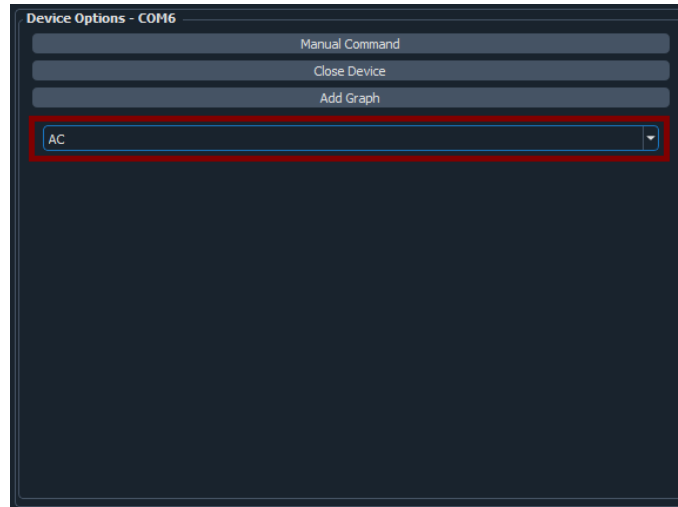


Figure 4.14: Dynamic device, dropdown menu

Just click on the name of the module you would like to interact with and click on one of it's pre configured options.

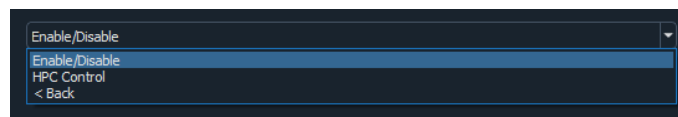


Figure 4.15: Dropdown menu command options

#### 4.1.7 Closing a connected device

Now you are all set up and are able to enjoy the application! To close a device, just click on the 'Close Device' button and you are set!

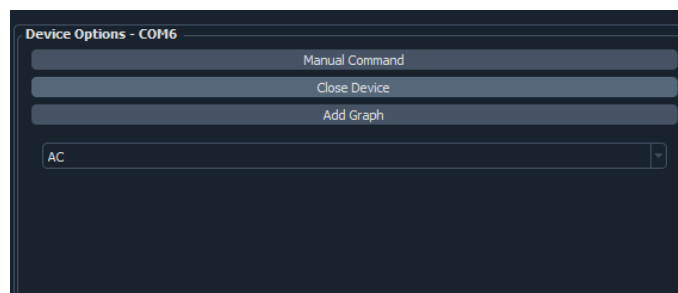


Figure 4.16: Close dynamic device

#### 4.1.8 Find communication port

This guid only works for windows sytems. For bluetooth connections, we will bring up our bluetooth settings and find the correct one. At first lets open our notifications in the task bar, right click on bluetooth, and then

“go to settings”.

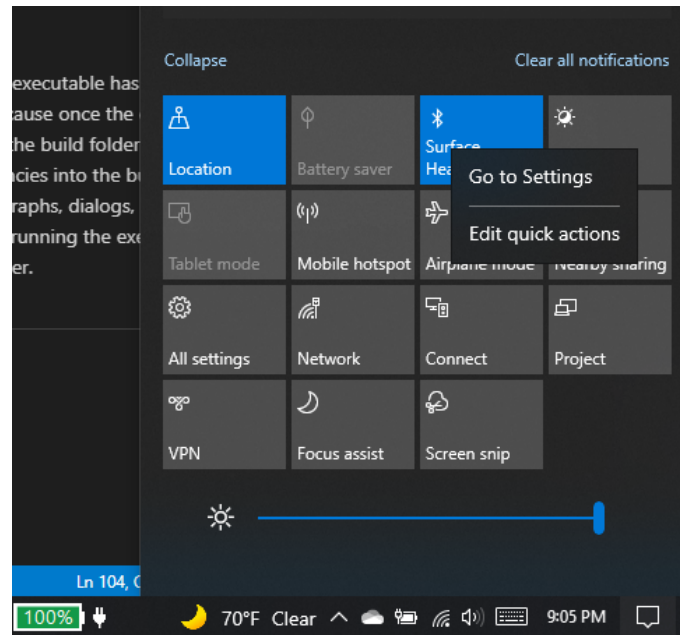


Figure 4.17: Bluetooth settings

Now, after the settings window is up, lets choose More Bluetooth options.

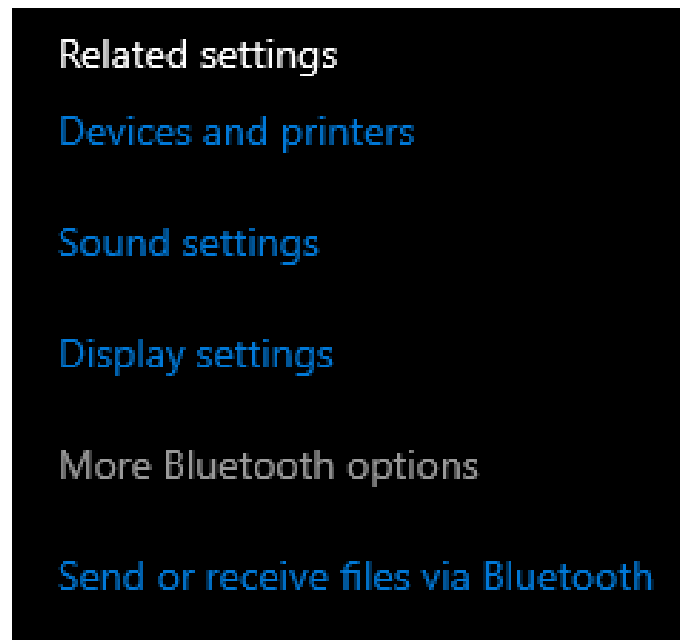


Figure 4.18: More bluetooth settings

After we have done this, we should get another little window, where we have the tab “COM ports”. If we click there we will see our connected and available bluetooth com ports. For example COM6 and COM8.

You need to use the ones with 'RNI-SPP'

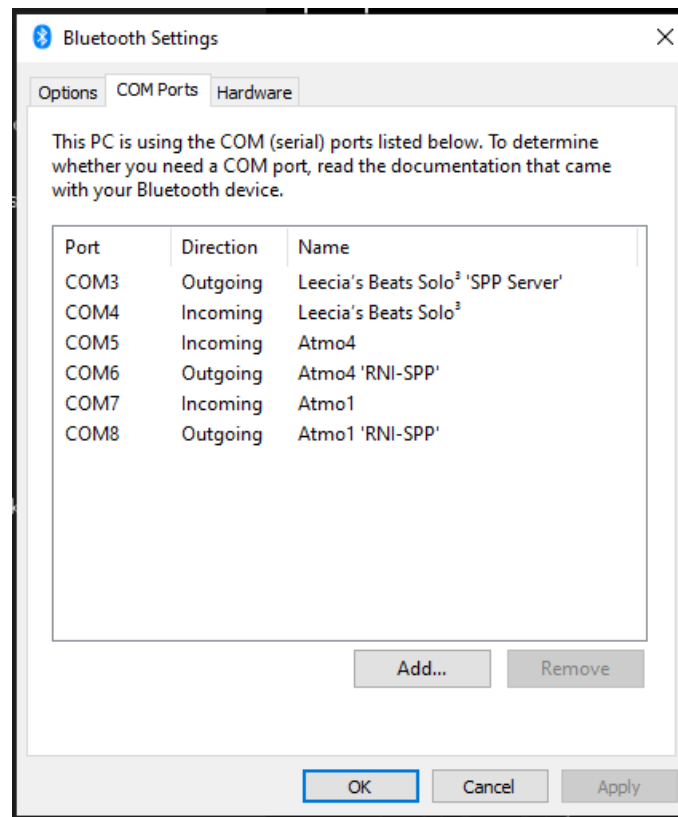


Figure 4.19: Com ports list

For physically connected devices, its a bit easier. We go to our search menu in Windows and type in “Device Manager”, press enter, and wait for the Device Manager window to pop up.

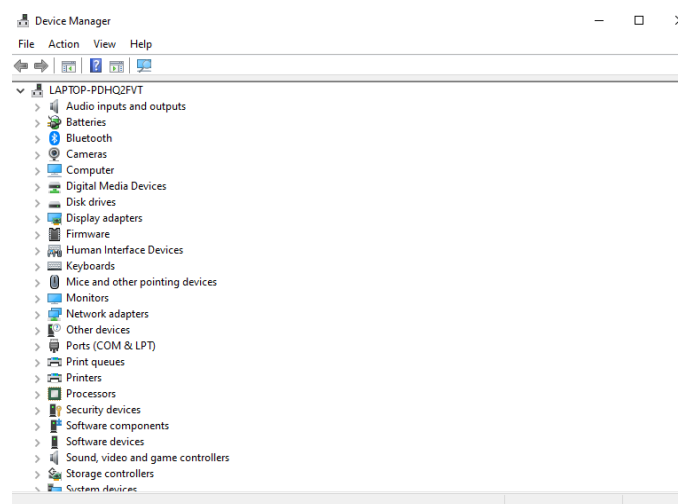


Figure 4.20: Device Manager

We now look for the “Ports (COM & LPT)” list, click on it, and voila! Here we have a list of all

communication ports, the only problem is that it doesn't really tell us the name of each of them. So to be able to find our correct communication port, we will have to connect and disconnect our radio receiver and check which of those communication ports disappears/appears.

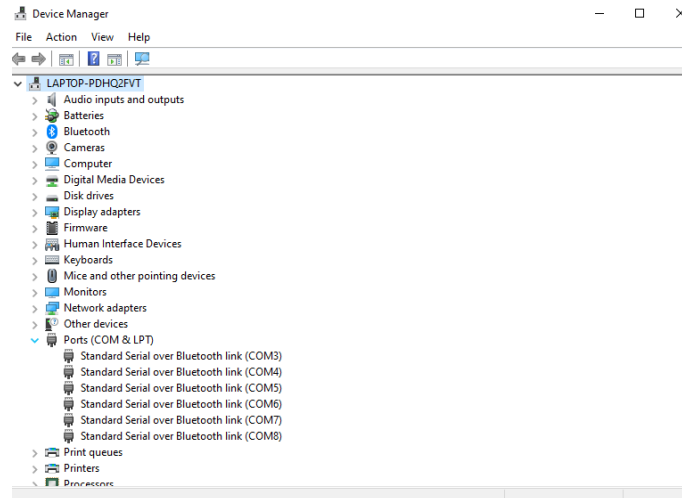


Figure 4.21: Device Manager Com Ports

## 4.2 Developers

### 4.2.1 Installation

- Installer

As a developer I think the Installer is not your best option, but why not, right? Just follow the installer instructions as you download it here [link...](#)

- Required libraries

First, you will need to clone this repository into your local branches. Then, inside the Docs folder there is a file with the name 'requirements.txt' this file contains the name of all libraries used in this application. This is very useful because we can use the command 'pip install -r requirements.txt' to download and install them all.

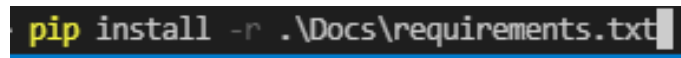


Figure 4.22: Installing requirements with pip

In my case, I have all those libraries installed already so I get a bunch of 'Requirement already satisfied' messages.



Figure 4.23: Requirements satisfied

If you have added a new library into the application, use the command 'pip freeze > ./Docs/requirements.txt' in the main folder and it should create a new 'requirements.txt' file that can be later used to install all required libraries.

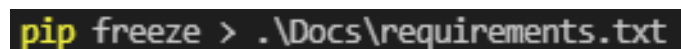


Figure 4.24: Pip freeze

#### 4.2.2 Creating the installer

For this project, I use cx.Freeze and InnoSetup to make an executable and the installer, it is fairly easy to make. I have already made a python file called setup.py which can be called with the command 'py setup.py build', this will run it and create an executable inside the folder build.

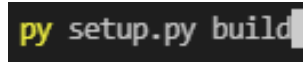


Figure 4.25: Setup command

lib	8/17/2021 8:50 PM	File folder	
atmoSniffer	7/25/2021 9:06 PM	Application	13 KB
python3.dll	5/3/2021 5:35 PM	Application exten...	59 KB
python39.dll	5/3/2021 5:35 PM	Application exten...	4,359 KB

Figure 4.26: Build folder

Once the executable has been created you will notice that it won't really work, this is because once the executable is created it does not move all required files into the build folder. Here is where we come in, we have to move in all dependencies into the build folder. All folders and files such as; receiver, ui, images, graphs, dialogs, DATA, com\_popup.py, Device.py, etc. Once this is done, try running the executable, if it works then we can move on to making the installer.

.git	8/16/2021 10:27 PM	File folder	
.github	6/18/2021 9:43 PM	File folder	
.vscode	6/18/2021 9:43 PM	File folder	
__pycache__	8/6/2021 10:38 PM	File folder	
build	8/17/2021 8:50 PM	File folder	
DATA	8/12/2021 7:11 PM	File folder	
dialogs	8/2/2021 6:32 PM	File folder	
Docs	7/12/2021 9:12 PM	File folder	
graphs	6/18/2021 10:09 PM	File folder	
images	8/17/2021 8:52 PM	File folder	
Installer	8/11/2021 10:42 PM	File folder	
legacy_atmoconsole_code	6/18/2021 9:43 PM	File folder	
receiver	6/18/2021 10:10 PM	File folder	
tests	8/4/2021 10:00 PM	File folder	
ui	8/1/2021 6:34 PM	File folder	
.gitignore	6/18/2021 9:43 PM	Text Document	2 KB
_init_.py	6/18/2021 9:43 PM	Python File	0 KB
atmoSniffer.py	8/11/2021 10:14 PM	Python File	7 KB
com_popup.py	6/18/2021 9:43 PM	Python File	1 KB
Device.py	8/1/2021 8:41 PM	Python File	7 KB
dynamicDevice.py	8/2/2021 7:34 PM	Python File	14 KB
FormatCSV.py	6/18/2021 9:43 PM	Python File	4 KB
loadingScreen.py	7/27/2021 10:38 PM	Python File	1 KB
README.md	8/17/2021 8:52 PM	MD File	10 KB
setup.py	8/11/2021 10:19 PM	Python File	1 KB
versionParsing.py	6/30/2021 8:15 PM	Python File	3 KB

Figure 4.27: Required files for executable to work



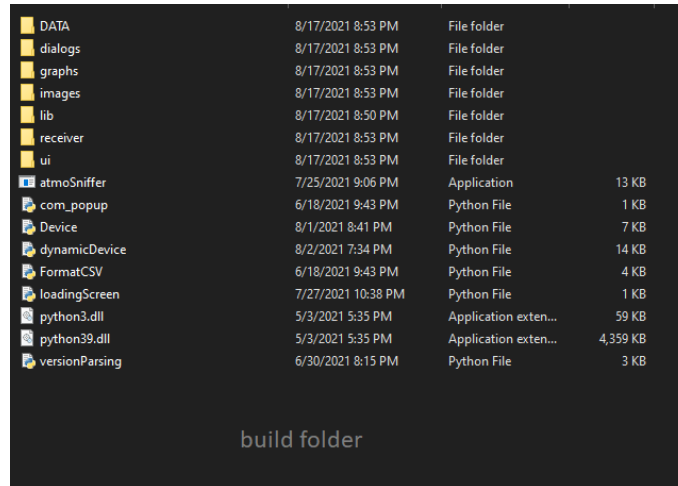


Figure 4.28: Inside build folder with required files

So I like I had said before, I use InnoSetup, it is fairly easy to use. I'm currently using their 6.2.0 version. You can download it from THIS LINK (<https://jrsoftware.org/isdl.php>). After running InnoSetup, you will get this Welcome dialog. Choose "Create a new script file using the Script Wizard" and click "Ok".

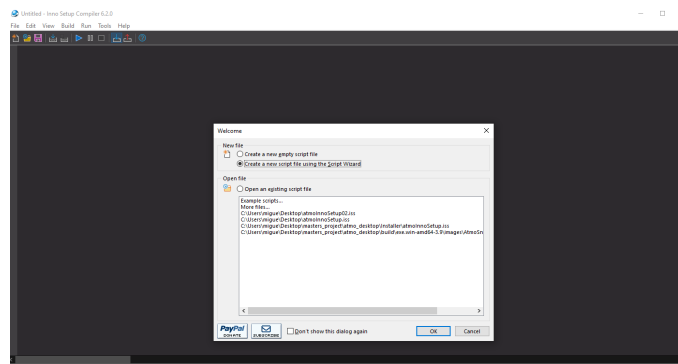


Figure 4.29: Using Inno Setup to make an installer

Click "Next" on the next popup window, until you get to this window where we need to fill in our application information.

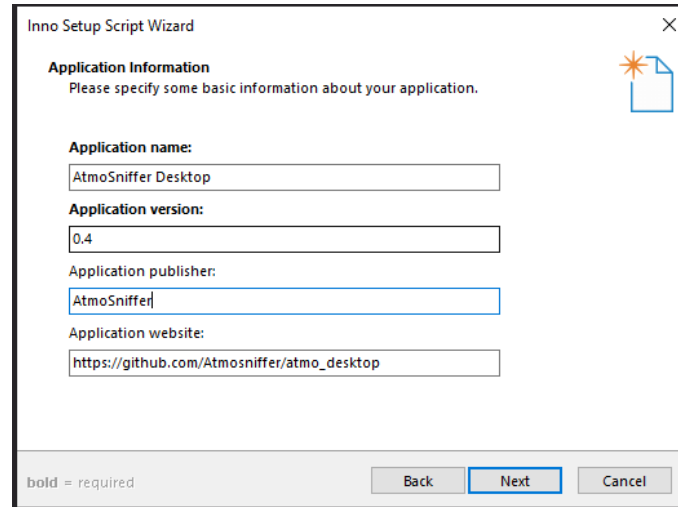


Figure 4.30: Information concerning our application

Next we will continue to press “Next” until we find ourselves in the Application Files Window, where we need to add all of the files in our “build” folder by clicking “Add File(s)” and/or “Add Folder”

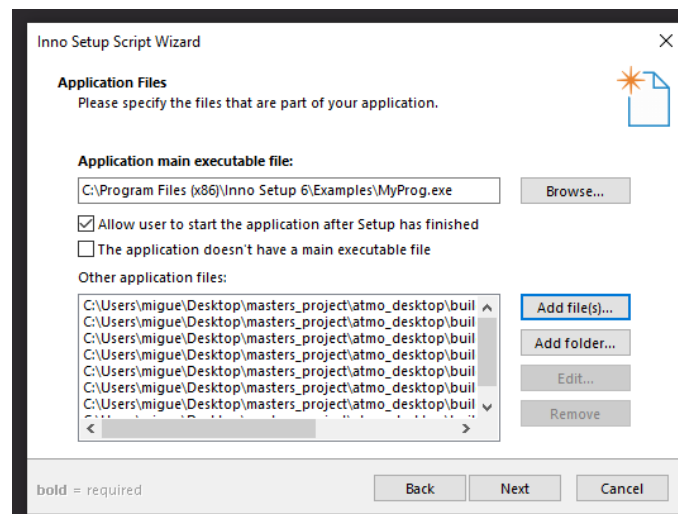


Figure 4.31: Adding files to the installer

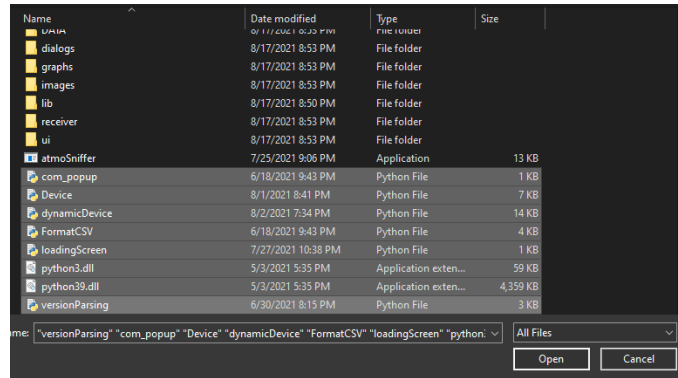


Figure 4.32: Selected files in Inno Setup

It is a little different for adding folders, first we choose which folder we would like to add and then we select it and click on the “Edit” button to make sure our folder contents are added to the right folder inside our application’s installation folder

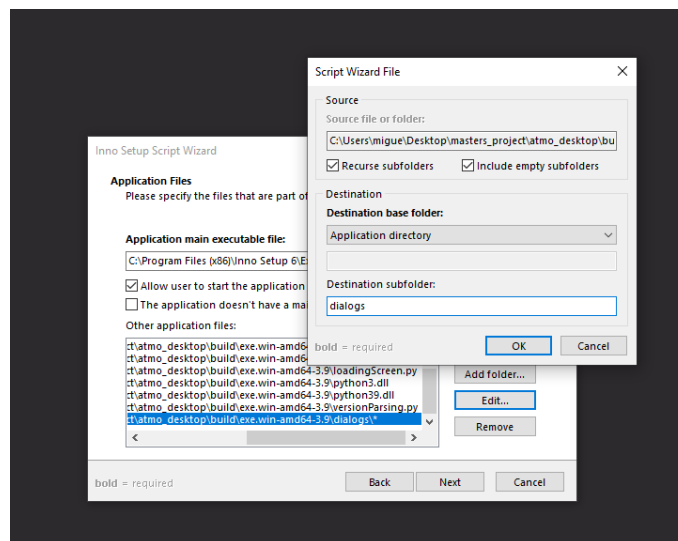


Figure 4.33: Adding folders to inno setup

In the image above, you can see that for the folder dialogs, you need to click on it, edit it and add the string “dialogs” under “Destination subfolder”, and then click “Ok”. Do this for all folders inside the build folder.

Do not include the .exe under “Other application files”

At last, we should select our executable file with the “Browse” button in the “Application Files” window.

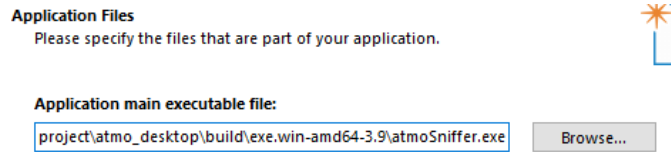


Figure 4.34: Selecting the exe file

Next, we continue clicking “Next” until we get to the “Compiler Settings” window where we are going to set our output file name, and choose the icon file from the build/images/ folder called logo\_icon.ico as the application’s icon image.

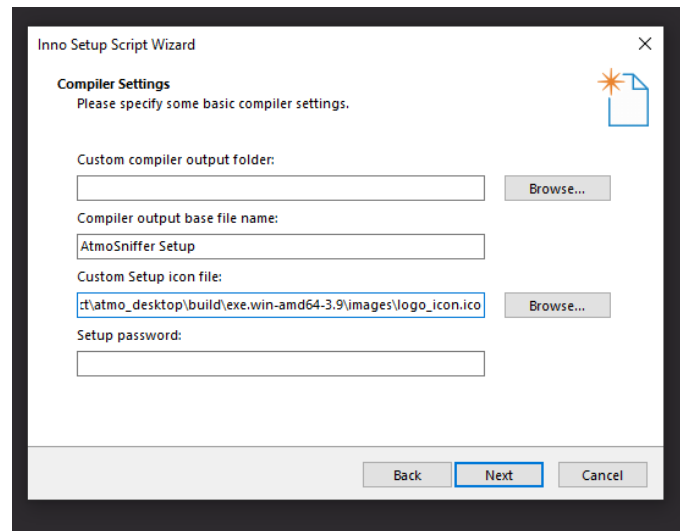


Figure 4.35: Choosing compiler inno setup

After you have something like the image above, click next until you see the finish button and click on it. Next up you will be asked if you want to run the script. Click on “No”. You will see something like this:

```

#define MyAppAssocExt ".mvp"
#define MyAppAssocKey StringChange MyAppAssocName, " ", "" + MyAppAssocExt

[Setup]
; NOTE: The value of AppId uniquely identifies this application. Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{80D0B4-46A7-46BF-955D-E21C18B47AF6}}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={autopf}\{#MyAppName}
ChangesAssociations=yes
DisableProgramGroupPage=yes
; Uncomment the following line to run in non administrative install mode (install for current user only.)
; PrivilegesRequired=lowest
OutputBaseFilename=AtmoSniffer Setup
SetupIconFile="C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\images\logo_icon.ico"
Compression=lzma
SolidCompression=yes
WizardStyle=modern

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\{#MyAppExeName}"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\com_popup.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\Device.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\dynamicDevice.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\FormatCSV.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\loadingScreen.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\python3.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\python39.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\versionParsing.py"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\dialogs\*"; DestDir: "{app}\dialogs"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\DATA\*"; DestDir: "{app}\DATA"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\graphs\*"; DestDir: "{app}\graphs"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\images\*"; DestDir: "{app}\images"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\lib\*"; DestDir: "{app}\lib"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\receiver\*"; DestDir: "{app}\receiver"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\ui\*"; DestDir: "{app}\ui"; Flags: ignoreversion recursesubdirs createallsubdirs
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

```

Figure 4.36: Inno Setup Script Example

From here, there are two things we need to ensure we do. First, we need to give our users permission to modify the DATA folder since our computer software will not allow us to modify anything inside the “Program Files” folder without permissions. For this we need to look for our “DATA” folder source and add “; Permissions: users-modify”

```

Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\python39.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\dialogs\*"; DestDir: "{app}\dialogs"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\DATA\*"; DestDir: "{app}\DATA"; Flags: ignoreversion recursesubdirs createallsubdirs; Permissions: users-modify
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\graphs\*"; DestDir: "{app}\graphs"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\migue\Desktop\masters_project\atmo_desktop\build\exe.win-amd64-3.9\images\*"; DestDir: "{app}\images"; Flags: ignoreversion recursesubdirs createallsubdirs

```

Figure 4.37: Giving users the required permissions to modify files in system folders

Now that we have that solved, we need to make sure the icon will show correctly for the setup/installer and the executable. Lets move down the script until we see the “[Icons]” Group, here we will copy both items and paste them right under the other two.

```

[Icons]
Name: "{autoprogams}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}";
Name: "{autodesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon
Name: "{autoprogams}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}";
Name: "{autodesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon

```

Figure 4.38: Setting up placeholders for icons.

And we edit both copies with IconFilename: “ {app}\images\logo\_icon.ico”

```
[Icons]
Name: "{autoprograms}\\{MyAppName}"; Filename: "{app}\\{MyAppExeName}";
Name: "{autodesktop}\\{MyAppName}"; Filename: "{app}\\{MyAppExeName}"; Tasks: desktopicon
Name: "{autoprograms}\\{MyAppName}"; Filename: "{app}\\{MyAppExeName}"; IconFilename: "{app}\\images\\logo_icon.ico"
Name: "{autodesktop}\\{MyAppName}"; Filename: "{app}\\{MyAppExeName}"; IconFilename: "{app}\\images\\logo_icon.ico"
```

Figure 4.39: Editing script line to correctly use icons.

Next, we save the script and run it. Once the script is done running you need to go to the folder where you saved your script at and look for the “Output” folder. It will contain our installation executable, which can then be used to install the AtmoSniffer Desktop application.

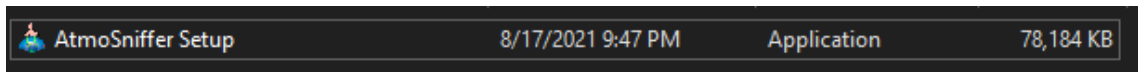


Figure 4.40: Atmosniffer setup installer.

### 4.2.3 Adding more commands to device command dropdown

We can use the AtmoSniffer Desktop application to communicate to connected “devices” this can be done by either, manually typing the command in a text box or by using a the custom dropdown embedded into a device widget. With the addition of new AtmoSniffer versions we could be getting new useful commands that would be much easier to use through the dropdown menu, rather than manually typing it in, but they haven’t been added yet.

If this happens we need to go to the dynamicDevice.py, here is where we can add new commands. At first we need to make a decision, do we want to add a new module to the whole program or do we just need to work on an existing module? If a new module is needed then we need to go to the self.cmd\_list.addItem() line and inside the function call we need to add the name of said module. Else, we just leave it how it is.

```
self.cmd_list.addItem(["AC", "GAS", "GPS", "Bluetooth", "Dongle", "OLED", "Log", "MB", "PM", "Profiler", "CO2", "External Modules"]])
```

Figure 4.41: Submenu cmd list.

Now that that’s done we go down to the on\_changed\_cmd\_list function where we will need to keep in mind which commands we will be sending to the selected module.

Here is where it gets a bit tricky but don’t worry I promise is not that hard. We are essentially building up a command inside a stack. The first “if” block checks for whether we have already started building a command or not. If not, then it will first add the module name to the command stack, “self.cmd”. and change the dropdown menu with the new options for said module.

```

if not self.cmd:
    if text == "AC":
        self.cmd.append("ac")
        self.cmd_list.addItem(["Enable/Disable", "Set ADDR", "Set Pump", "Read", "Raw", "< Back"])
        self.cmd_list.showPopup()
    elif text == "GAS":
        self.cmd.append("gas")
        self.cmd_list.addItem(["Enable/Disable", "HPC Control", "< Back"])
        self.cmd_list.showPopup()

    elif text == "GPS":
        self.cmd.append("gps")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "Bluetooth":
        self.cmd.append("bluetooth")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "Dongle":
        self.cmd.append("dongle")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()

    elif text == "OLED":
        self.cmd.append("oled")
        self.cmd_list.addItem(["msg", "Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "Log":
        self.cmd.append("log")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "MB":
        self.cmd.append("mb")
        self.cmd_list.addItem(["Info", "Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "PM":
        self.cmd.append("pm")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "Profiler":
        self.cmd.append("profiler")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()
    elif text == "CO2":
        self.cmd.append("co2")
        self.cmd_list.addItem(["Enable/Disable", "< Back"])
        self.cmd_list.showPopup()

```

Figure 4.42: First if block in the command making process.

Next, we know that the option the user clicked on cannot be the first time they clicked so we check which option they clicked on, and depending on the first index of our command stack we change the command dropdown with new options, but not without checking if the first index module is supposed to have these new options first. We always want to append something to our stack even if its meaningless, otherwise the function will not work since it could remove the wrong item once the option <back is clicked, for this we add meaningless options surrounded by <and>. Once an end argument is reached, we add the end argument and the item <end> to our command stack For example if the option they clicked on is “Battery Volts” then we first check if the first index of the command stack has the word “mbi”, if so we append a “4” and then the <end>tag.

```

if text == "Battery Volts":
    if self.cmd[0] in ["mbi"]:
        self.clear_cmd_list()

        self.cmd.append("4")
        self.cmd.append("<end>")

```

Figure 4.43: Battery volts command example.

Lastly, we check if the command stack has the <end>tag, if so we build the command by first looking for all those meaningless tags and removing them, and then joining the remaining stack items into a single string and sending it to the device itself with a helper method.

#### 4.2.4 Add more headers for upcoming versions

At the time this guide is being created, AtmoSniffer has 2 versions; 2.0 and 3.0. The project will continue to grow which means that more versions will be added. For this, we need to make sure we tell the application about the new versions or it will continue to crash. This can be done by going to the file versionParsing.py

```

def getHeader(version):
    if(version == 2.0):
        return ['version', 'sec_start', 'sec_end', 'gps_time', 'gps_date', 'gps_lat', 'gps_long', 'gps_alt', 'gps_lock', 'reserved', 'reserved', 'reserved', 'reserved']
    elif(version == 3.0):
        return ['version', 'sec_start', 'sec_end', 'gps_time', 'gps_date', 'gps_lat', 'gps_long', 'gps_alt', 'gps_lock', 'reserved', 'reserved', 'reserved', 'reserved']

```

Figure 4.44: Version parsing.

We just need to add a new elif statement with our new version number with the correct returned header. Keep the version number as a double/float.

#### 4.2.5 Run Tests

Now you want to run tests, right? Well its very simple. We can either run gui tests or unit tests. At first, lets run gui tests. You will need to have the project open on VSCode, making sure the file “atmoSniffer.py” is visible on the left explorer menu.



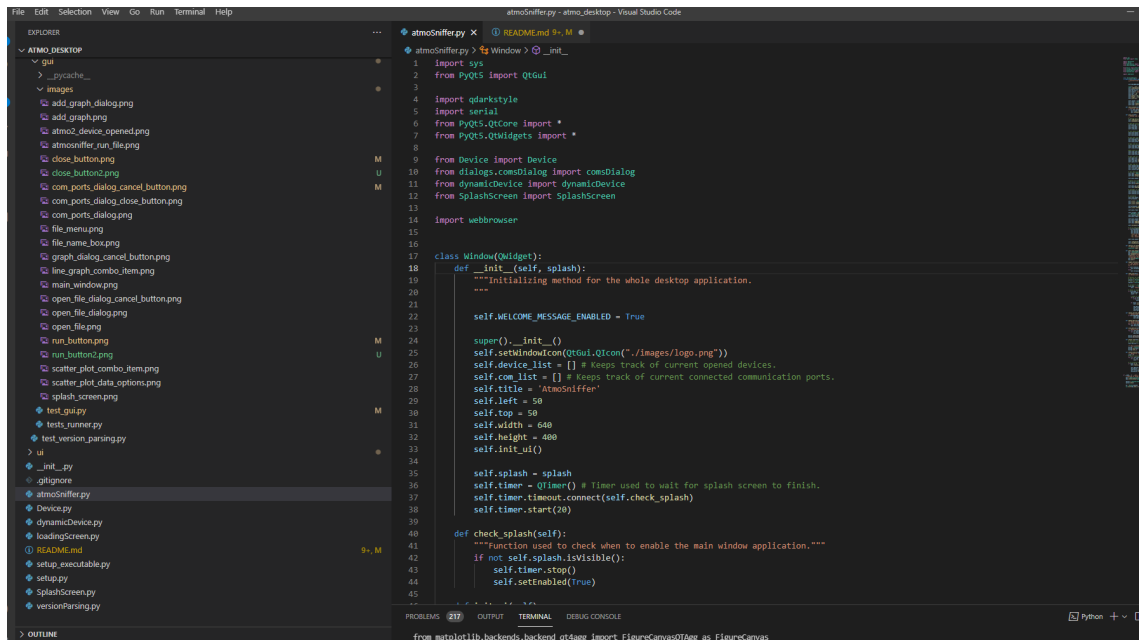


Figure 4.45: vscode Atmosniffer.

Now you need to open a command prompt terminal, navigate to the tests/gui directory and run the tests\_runner.py file making sure the that vscode is visible.

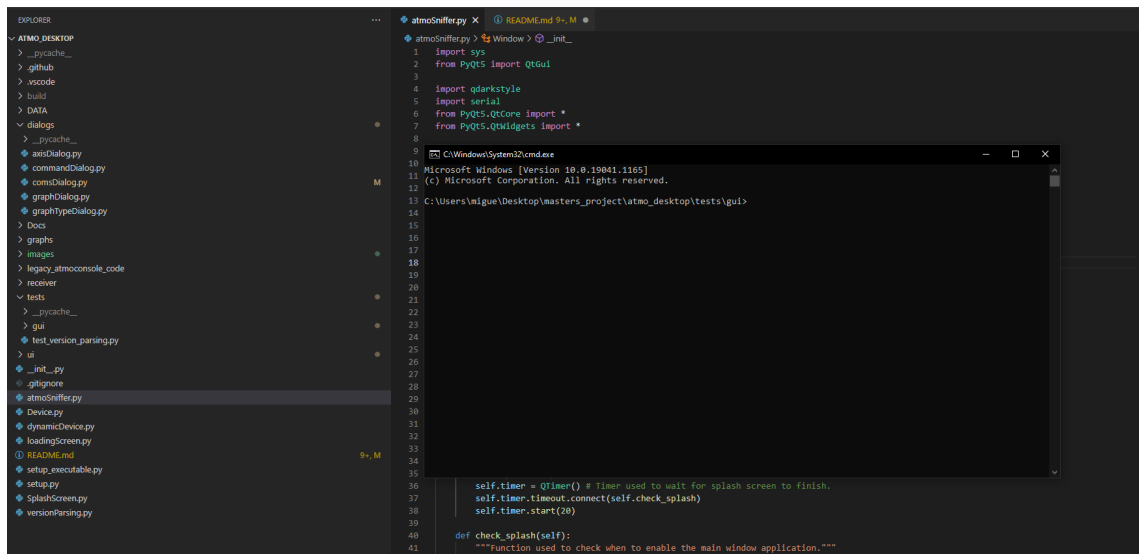


Figure 4.46: vscode Atmosniffer run tests.

Now we just let the tests run and hope to get an OK at the end. For the unit tests, we open the file test\_version.parsing.py and click on the run test icon by the test/method name.

# **Appendices**

## Appendix A

### Python Libraries used to develop the application

Here is a list of all python libraries used in the developing of the atmosniffer desktop application:

- pyqt5

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android. [3]

<https://pypi.org/project/PyQt5/>

- serial

A framework for serializing/deserializing JSON/YAML/XML into python class instances and vice versa [4]

<https://pypi.org/project/serial/>

- webbrowser

The webbrowser module provides a high-level interface to allow displaying Web-based documents to users. Under most circumstances, simply calling the open() function from this module will do the right thing. [5]

<https://docs.python.org/3/library/webbrowser.html>

- pandas

Powerful data structures for data analysis, time series, and statistics [6]

<https://pypi.org/project/pandas/>

- csv

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats. [7]

<https://docs.python.org/3/library/csv.html>

- os

This module provides a portable way of using operating system dependent functionality. [8]

<https://docs.python.org/3/library/os.html>

- `cx_freeze`

`cx.Freeze` creates standalone executables from Python scripts, with the same performance, is cross-platform and should work on any platform that Python itself works on. [9]

<https://pypi.org/project/cx-Freeze/>

- `matplotlib`

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. [10]

<https://pypi.org/project/matplotlib/>

- `io`

The `io` module provides Python's main facilities for dealing with various types of I/O. [11]

<https://docs.python.org/3/library/io.html>

- `datetime`

The `datetime` module supplies classes for manipulating dates and times. [12]

<https://docs.python.org/3/library/datetime.html>

## References

- [1] J. E. Sohl, “the atmosniffer: A broad spectrum, air & enverionmental monitoring system,” *Business Plan*, 2019.
- [2] “Ambient (outdoor) air pollution.” [Online]. Available: [https://www.who.int/news-room/fact-sheets/detail/ambient-\(outdoor\)-air-quality-and-health](https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health)
- [3] “Pyqt5.” [Online]. Available: <https://pypi.org/project/PyQt5/>
- [4] “Pyserial.” [Online]. Available: <https://pypi.org/project/pyserial/>
- [5] “Webbrowser - convenient web-browser controller.” [Online]. Available: <https://docs.python.org/3/library/webbrowser.html>
- [6] “Pandas.” [Online]. Available: <https://pypi.org/project/pandas/>
- [7] “Csv - csv file reading and writing.” [Online]. Available: <https://docs.python.org/3/library/csv.html>
- [8] “Os - miscellaneous operating system interfaces.” [Online]. Available: <https://docs.python.org/3/library/os.html>
- [9] “Cx-freeze.” [Online]. Available: <https://pypi.org/project/cx-Freeze/>
- [10] “Matplotlib.” [Online]. Available: <https://pypi.org/project/matplotlib/>
- [11] “Io - core tools for working with streams.” [Online]. Available: <https://docs.python.org/3/library/io.html>
- [12] “Datetime - basic date and time types.” [Online]. Available: <https://docs.python.org/3/library/datetime.html>