Tool development and characterisation of quantum fluids of light in hot atomic vapors

Thomas PICOT

Resume : This report is the conclusion of the work I did during my internship at the Kastler Brossel laboratory of Sorbonne University, between my under-graduated degree and my master's degree. I have worked on quite a few projects that are compiled inside. The main research topic is the Quantum fluids of light. These fluids are analogous to Bose-Einstein condensates (BECs). We are able to manipulate it with fewer constraints that cold atoms gas and this is what makes their research so interesting. In order to advance research on the subject, the team had to develop various tools in parallel, to which I contributed a lot. I was also able to familiarise myself with various notions of non-linear quantum optics which you will find in this report.

Key words : - *Quantum fluids of light* - *Optical Non-Linear Medium* - *Hot Rubidium vapor* - *Software development* -

Internship supervised by :
Quentin Glorieux
quentin.glorieux@lkb.upmc.fr / office number. (+33) 1 44 27 41 89
Laboratoire Kastler Brossel
4 Place Jussieu
75005
http://www.lkb.upmc.fr

Acknowledgements:
I would like to thank Quentin for the welcome he gave me. He quickly gave me responsibilities and let me lead projects. I would also like to thank the interns, PhD and post-doctoral students of the rubidium team for helping me in optical physics and programming. Thanks to Alberto Bramati and the polariton team who were always present and with whom I had fun. Special thanks to the mechanical and electrical workshops for helping me build a very time consuming oven. I am very grateful to you all for taking part in this adventure.

Contents

Introduction

Quantum fluids of light are a non-linear quantum optical phenomenon that occur due to the Kerr effect. This one produces photon-photon interactions. Its propagation is described by the nonlinear Schrödinger equation. (NLSE) such that :

$$i\partial_z \mathcal{E}_0(\mathbf{r}_\perp, z) = \left[-\frac{1}{2k}\boldsymbol{\nabla}_\perp^2 - \frac{i\alpha}{2} - \frac{3}{8}\frac{k}{n_0^2}\chi^{(3)}(\omega) \mid \mathcal{E}_0(\mathbf{r}_\perp, z) \mid^2\right] \mathcal{E}_0(\mathbf{r}_\perp, z) \tag{0.1}$$

Where $\mathcal{E}_0$ is the envelop of the electric field in the void, propagating across the z-direction, α the absorption coefficient such that $\alpha = k_0 n_0 Im(\chi^{(1)})$, with $\chi^{(1)}$ and $\chi^{(3)}$ the first and the third coefficients of susceptibility. We need to be in the approximation of a paraxial light. It means that we need a slowly-varying function of z, with $\boldsymbol{\nabla}^2\mathcal{E}_0/k^2 \sim |\partial_z \mathcal{E}_0|/k \ll 1$. We can then consider a local perturbation δn of n, the linear refractive index. Finally, we get :

$$i\partial_z \mathcal{E}_0(\mathbf{r}_\perp, z) = \left[-\frac{1}{2k}\boldsymbol{\nabla}_\perp^2 - \frac{i\alpha}{2} - k\frac{\delta n(\mathbf{r}_\perp, z)}{n_0}\frac{3}{8}\frac{k}{n_0^2}\chi^{(3)}(\omega) \mid \mathcal{E}_0(\mathbf{r}_\perp, z) \mid^2\right] \mathcal{E}_0(\mathbf{r}_\perp, z) \tag{0.2}$$

Now we can compare equation (0.2) with the Gross-Pitaevskii equation (GPE) :

$$i\hbar\partial_t \Psi(\mathbf{r}_\perp, t) = \left[-\frac{\hbar^2}{2m}\boldsymbol{\nabla}^2 + \mathcal{V}(\mathbf{r}_\perp) + g \mid \boldsymbol{\Psi}(\mathbf{r}_\perp, t) \mid^2\right] \boldsymbol{\Psi}(\mathbf{r}_\perp, t) \tag{0.3}$$

With m the boson mass, $\mathcal{V}$ an external potential and $g = 4\pi\hbar^2 a_s/m$ the coupling constant. This equation describes the time-evolution of a Bose-Einstein condensate (BEC) in the Hartree-Fock approximation.

It is in this context that the Quantum Fluids of Light team led by Alberto Bramati and Quentin Glorieux is evolving. They seek to characterise the nonlinear medium that gives birth to this effect and its various quantum and optical properties. They are also looking for new applications in artificial intelligence with polaritons and hydrodynamic instabilities. For my part, I joined the hot alkaline vapors team which uses natural isotopes of rubidium as a nonlinear medium of interactions.

We can see that (0.2) propagates along z-direction while equation (0.3) propagates along the time t. It is not really a problem experimentaly because we still can see non-linear effects of the light. See [8] to see how to get the final equation in vapour cells :

$$i\frac{\partial\psi}{\partial\tau} = \left(-\frac{1}{2}\boldsymbol{\nabla}_\perp^2 + \mid \psi \mid^2\right)\psi \tag{0.4}$$

With L the length cell, $\psi = \frac{\mathcal{E}_0}{\sqrt{\rho(0,L)}}$, $\tau = z/z_{NL}$ where $z_{NL} = \frac{1}{g\rho(0,L)}$ is the non-linear axial length and $\rho(0,L) = \rho(0,0)e^{-\alpha L}$ the decaying density. One may ask what are the advantages of using light quantum fluids instead of cold atoms. The different properties of these fluids have allowed the emergence of new fields of research, thanks to the simplicity of the experimental setup, the repeatability of the experiment and its low energy cost. The main advantage of light quantum fluids is that the measurement does not destroy the Bose-Einstein condensate. In this report, I will present the different projects I participated in during my internship. The first part will be about the application I developed to control the power and frequency of a laser. This was the main project of my internship and although it is not directly related to fundamental physics, it taught me to master a lot of programming tools that are very useful for modelling, simulations, experiment setup and results analysis. The second part will be on the modelling of the transmission spectrum of the D-line transition of natural rubidium isotopes. This part makes me make a transition between programming and physics. It is a basic project but very useful for the team. I will then talk about a project of realisation of a rubidium cell furnace: from the conception to the realisation of the part. I will finish by presenting two experiments that I carried out: The first one on the measurement of the dispersion of bogolons, quasi-particles that propagate in the non-linear medium. The second one on the measurement of the nonlinear refractive index. Then I will conclude on what the internship brought me and what are my new perspectives.

1 Application development to control a laser

1.1 challenges

The main objective was to control a μquans laser delivered as a prototype. In the beginning, the only way to control the laser was by placing commands in a console. It was really laborious and not very elegant. I had to develop a desktop application in python to control the laser with buttons. We chose python rather than C++ or JAVA as a language because it was easy to learn and user friendly. The trick is that we can pass the commands of the laser through python.

We need to establish a connection between the laser and the computer through a network protocol, which has already been interfaced in python module. There are several network protocols such as VISA and Telnet. It is the latter that we use. We can separate the project in two parts: the interfacing and the GUI. To do these two parts we use object oriented programming (OOP).

First, we need to understand how the laser works and what are the different parts we can interact with. We can see figure-1 that we have a diode and an EDFA in our laser. The gas cell is our rubidium cell that we have to setup (see figure-sas). Applying a current to the EDFA will generate laser power. We can do that by passing commands. And applying tension to the diode will change the laser frequency. To do this we need to generate a voltage across the diode. We use for that an AFG or a Red Pitaya that we interface, you will see in next section how we interface devices with python.

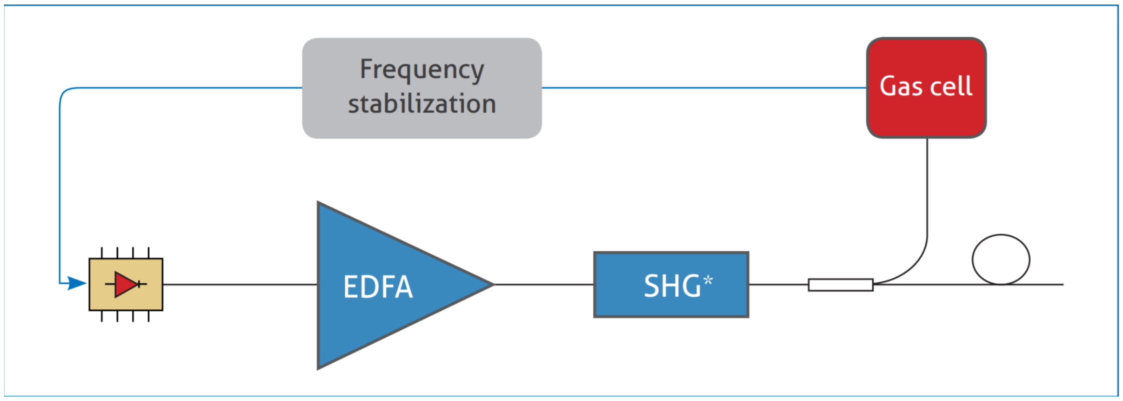


FIGURE 1

In this section, I will first introduce what the OOP language is and why it is useful for interfacing. Then I will present the structure of my application, how I built it and how I manage the different views. Finally I will present you my code.

1.2 Object-Oriented Programming in Python

Using OOP is to create programming objects that can be implemented in the main code. For this we need to create Classes and define some of their properties.

1.2.1 Python Classes

To create a class in Python we need to write the following syntax :

```
class MyClass:
```

LISTING 1: Declare a Python Class

Now you can write a random script in you class and call it in your main script by basically typing :

```
class MyClass:
    print("Hello World!")


""" Main script"""
MyClass()
>>>Hello World!
```

LISTING 2: Use a class

For the moment, this class does the same thing that a typical script. Indeed, we could just write print("Hello World!") and get the same result. The point is that the class has an initialisation function called init() that is initialised when the class is called in the main script. This init() function can take arguments and we can make inherit all those arguments in the entire class by adding the prefix self. to the class variables. :

```
class MyClass:
    def __init__(self, string_var: str):
        self.string_var = string_var

    def firstMethod(self):
```

```
 6          print(self.string_var)
 7
 8
 9 """Main script :"""
10 myclass = MyClass(string_var="Hello World!")
11 myclass.firstMethod()
12 >>>Hello Wolrd!
```

Listing 3: initialising MyClass and creating a method

A method is the name given to a class function. As the variables, it is an attribute from MyClass. to call it, we need to use the object myclass (l-10) and add the attribute with a dot (l-11).

The last thing I wanted to present you in this section is the class inheritance. It allows us to make inherit attributes from a parent class to a children class.

```


class MyClass:
    def __init__(self, string_var: str):
        self.string_var = string_var

    def firstMethod(self):
        print(self.string_var)


class ChildrenClass(MyClass):
    pass


"""Main script :"""
my_children_class = ChildrenClass(string_var="Hello World!")
my_children_class.firstMethod()
>>> Hello World!
```

Listing 4: Class inheritance

I introduced you the basics of what a python class can do. We will see in the next section what we can do with this in our case to control a laser.

1.3 interfacing

We can define in experimental physics the interfacing as the way to pass instructions from the computer to the device (in our case the Laser). Indeed, it is possible to communicate by network protocol to those devices. In our case we are using telnet, and by chance, it exists a python package that allows us to directly connect the computer to the device by a simple script :

```
import telnetlib
TelnetClient = telnetlib.Telnet(host:str, timeout:int=1)
TelnetClient.open(host:str, port:int)
```

Listing 5: establishing a connection to laser devices

Where host is the laser IP address, timeout is an argument to avoid bugs and port is the network port by default equal to 23.

Finally we can send instructions to the laser by writing

```
TelnetClient.write("INSTRUCTION".encode('ascii') + b"\n")
```

Listing 6: Basic instruction

Now we just have to write a class Laser that establish in the init() method the connection to the device and create methods for each instructions. We finally get :

```
class Laser:
    def __init__(self, host: str = "XXX.XXX.X.XXX", port: int = 23):
        """
        initiate a connection to the laser Muquans
        :param host: IP addr in the network lab of the laser, by defatult it is kapaz
        :param port: value needed to establish a connection to the good port, should not be changed.
        """
        # needed to open a connection to Laser with his telnet Protocol
        self.host = host
        self.port = port
        self.TelnetClient = telnetlib.Telnet(self.host, timeout=1)
        self.TelnetClient.open(self.host, self.port)

        if self.host == "XXX.XXX.X.XXX":
            # power calibre to get edfa setting from power
            x_array = np.array([0.0, 5.3, 18, 35.3, 53.6, 71.2, 91.4, 111, 130, 149, 167, 185, 203,
    222, 240, 259, 277, 300, 325, 347, 371, 401, 429, 464, 500, 544, 598, 662, 736, 826, 916])
            y_array = np.arange(0, 3.1, 0.1)
            self.f = interpolate.interp1d(x_array, y_array)
        else:
            x_array = np.array([0.0, 10.3, 34.5, 46, 84, 127, 174, 222, 272, 322, 375, 426, 478, 531,
    585, 638, 692, 747, 802, 857, 922, 975, 1035, 1095, 1145, 1195, 1220, 1295, 1337, 1380, 1417])
            y_array = np.arange(0, 3.1, 0.1)
            self.f = interpolate.interp1d(x_array, y_array)

    def diode_on(self):
        self.TelnetClient.write("sml780_tool Enable_Current_Laser_Diode on".encode('ascii') + b"\n")

    def diode_off(self):
        self.TelnetClient.write("sml780_tool Enable_Current_Laser_Diode off".encode('ascii') + b"\n")

    def shutdown(self):
        self.TelnetClient.write("sml780_tool edfa_shutdown".encode('ascii') + b"\n")

    def exit(self):
        self.TelnetClient.write("sml780_tool edfa_shutdown".encode('ascii') + b"\n")
        self.TelnetClient.write("exit".encode('ascii') + b"\n")

    def command(self, string):
        self.TelnetClient.write(f"sml780_tool {string}".encode('ascii') + b"\n")

    def set_power(self, power):
        self.TelnetClient.write(f"sml780_tool edfa_set {power}".encode('ascii') + b"\n")
```

Listing 7: Class Laser

We apply this strategy for every devices we need. Now we need to connect all the devices to the main code.

1.4 Application Structure

In the previous section, I have exposed how and why to interface the devices. Now let's talk about how we want to build this app. For that we need to think about user experience. Indeed, an application can be functional but unused because of its complexity to handle. A good way to see if it's easy or not to use is by drawing the different views and their utilities, as figure-2.

1.4.1 schema

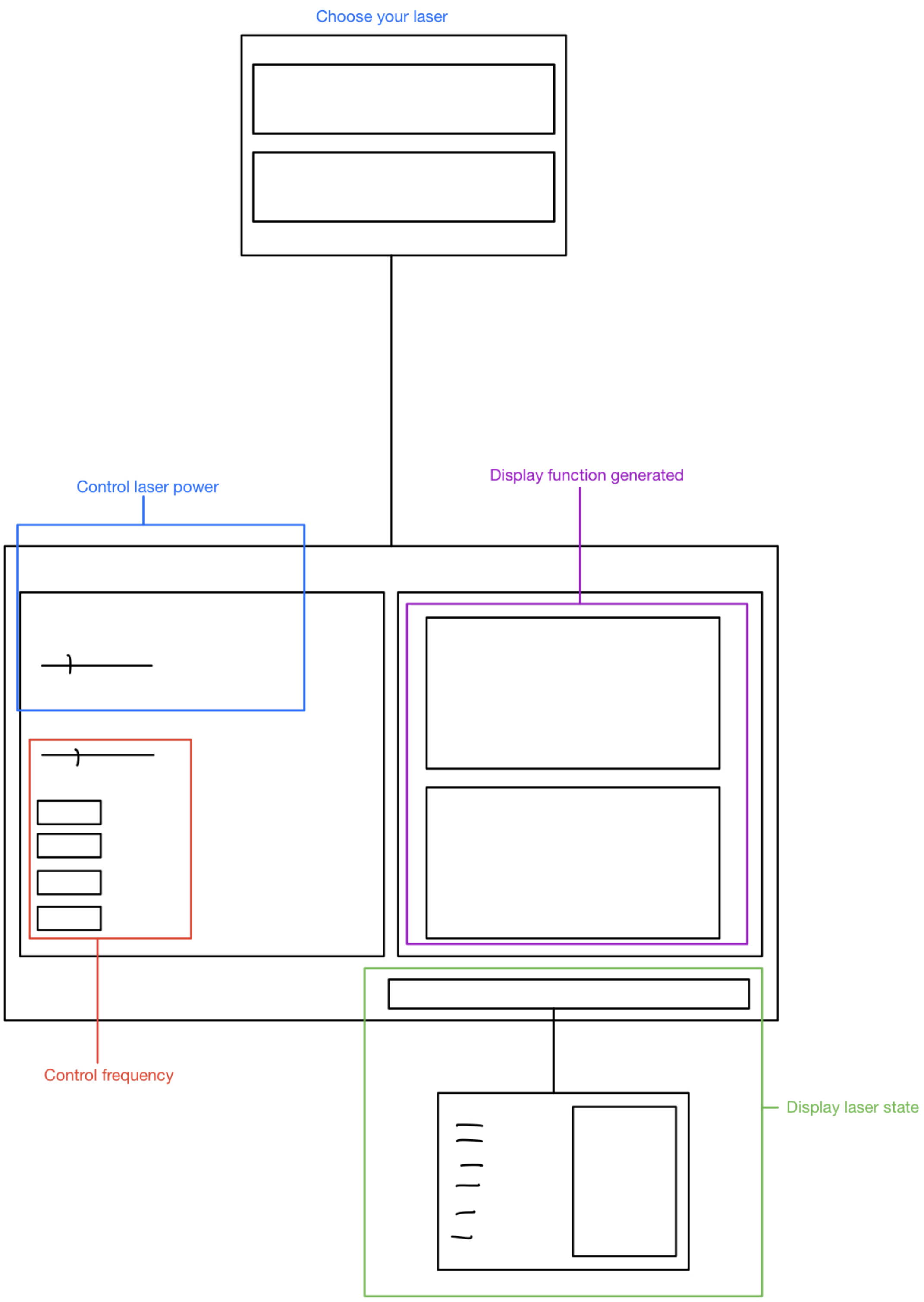


Figure 2

We can segment every part of the code as presented in figure - 2. Starting from the top, we open the app on a view that asks you to choose a Laser (figure-3).

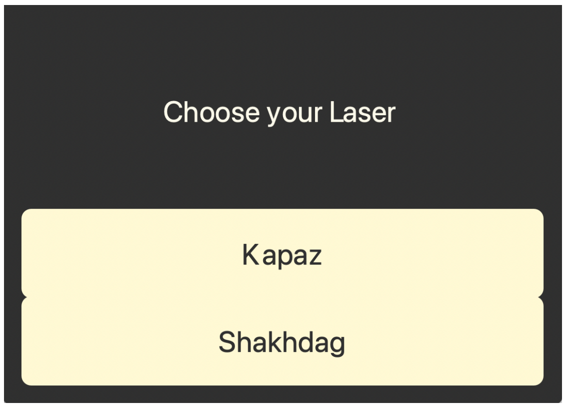
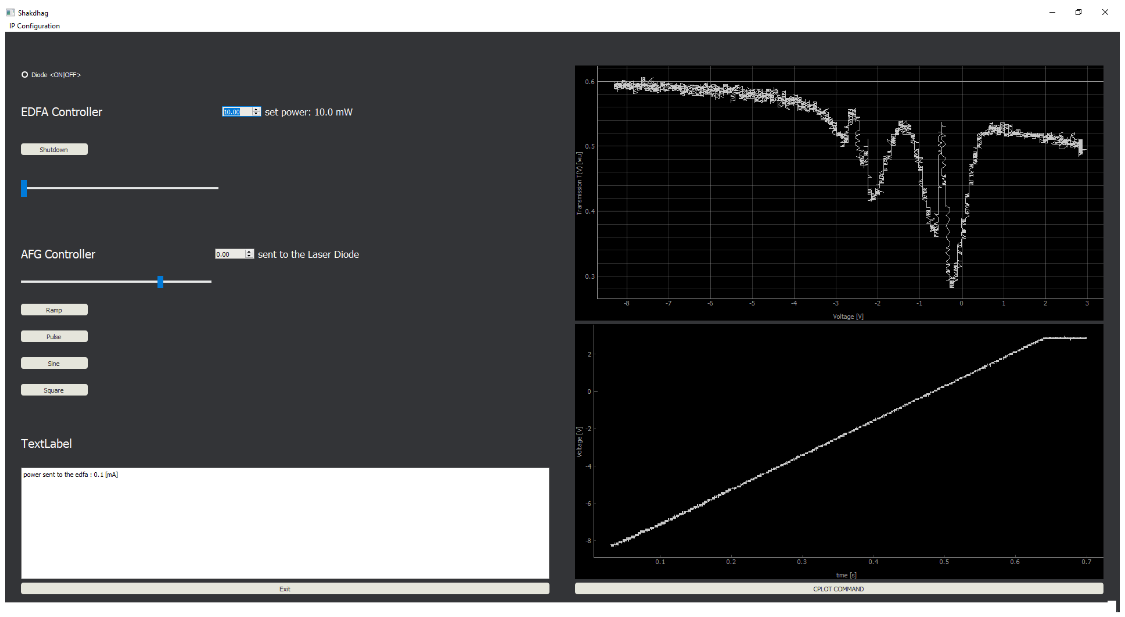


FIGURE 3

Depending on the laser you click on, it will open the main view and initiate the code with the good arguments. (see figure-code) (figure-4).



10.00 set power: 10.0 mW

0.00 sent to the Laser Diode

power sent to the edfa : 0.1 [mA]

FIGURE 4: Final GUI that controls the laser in power and frequency. On the left side we have the control widgets. On the top we can control the power. On the AFG controller part we have buttons to select the function we want to generate to control the laser frequency. On the right side, we display the laser frequency and the function generated. We can see in our case that we generate a ramp function. You can compare this figure with figure-2

I present in this report only the main functions of the GUI but I added other properties like the possibility to have, in time the laser state like temperature or current. In the next section I will present you my code structure.

1.4.2 code

I decided to instantiate every devices when the main view open. So I had to declare every device class in the init() function of my main window as bellow :

```
class UiMainWindow(object):
    def __init__(self, host: str = "XXX.XXX.X.XXX", channel: float = 2):
        """
        init the main view of the GUI.
        :param str host: IP address of the laser, depending of the choice you made during the
    selection. by default he
        will initiate Shakdhag
        :param int channel: red Pitaya channel, 1 for shakdhag and 2 for kapaz
```

```
 8         """
 9         self.host: str = host
10         self.channel: int = channel
11         self.addr_scope: str = 'TCPIP::XXX.XXX.X.XXX::INSTR'
12         self.laser = Laser(host=self.host)
13         self.red = MyRedpitaya()
14         self.worker = WorkerThread(self.plotAbsSat)
15         self.timer = QtCore.QTimer()
16         self.timer.setInterval(5000)
17         self.timer.timeout.connect(self.get_data_with_thread)
18         self.timer.start()
```

Listing 8: init mainWindow class. It is the class of the main view.I will talk later about the lines 14 to 18.

I pass in attribute host and channel because I use them in my code later. The setupUi function has been generated by a PyQt5 tool. This function generates the window with the UI, where UI means "User interface". PyQt5 is the python version of Qt5, an UI creator software that creates application desktops, widely use in companies. So I just had to manage widgets an layouts and generate my code. Then I had to connect every widgets to functions and make it work. a simple example of how to do that is connect a button to a laser command.

```
def exitButton(self):
        """
        exit the software without danger
        :return: None
        """
        self.laser.shutdown()
        time.sleep(4)
        self.laser.diode_off()
        self.laser.exit()
        sys.exit()
```

Listing 9: exitButton method that disconnect the laser and quit the app.

We can see that we call laser attribute methods, where laser has been initiated in the init(). to connect this function to its button we just need to write :

```
self.exit_button.clicked.connect(self.exitButton)
```

Listing 10: Connect button to method.

We need to do that for every buttons and we finally have an operational app. The only problem remaining is the RAM threading. Indeed we want to execute at least two tasks at the same time. The first one is control the laser and the second one is display something like frequency or laser state. But in programming it is not allowed to execute two tasks at the same time on the same thread. every program runs on a unique thread by default but you need to manage them if you want to run multi-tasks. This is what I do in the next section.

1.4.3 multi-threading

In our case, to execute two tasks at the same time we need to have the following properties in our code :

1. To execute a task on anther thread we need to build a class thread with a method "run".

```
    def run(self):
        """
        run the function. must never be called
        :return: None
        """
        try:
            self.fun()
            self.output.emit(self.fun)
        except:
            self.dialog_browser.setText('problem with the thread, grow the period time : self.time.setInterval(ms)')

```

Listing 11: method run from WorkerThread class.

2. This class is a children of the main class.

```python
    class WorkerThread(QThread, UiMainWindow):
        output = QtCore.pyqtSignal(np.ndarray)

    def __init__(self, fun, parent=None):
        """

        :param fun: enter the function you want to let run in another thread, in our case it is
    the method that plots
        in time the saturated absorption cell.
        :param parent: None
        """
        QThread.__init__(self, parent)
        self.fun = fun

```

Listing 12: UiMainWindow is the main class

3. The task we want to execute in parallel has to be a method from the main class.

```python
        def get_data_with_thread(self):
            """
            call the method that executes the thread.
            :return: None
            """
            self.worker.get_data()

```

Listing 13: Method from UiMainWindow class.

Then we want to execute the two tasks at the start of the main window. this is why we have, Listing 9 lines 14 to 18.

To conclude, we are able to change the frequency of the laser by generating a voltage across the diode and we can generate the laser power by passing instructions with the GUI buttons. We used the OOP to group the different classes that interface the devices into a single project. The disadvantage of this setup is that we need two devices: a Red Pitaya or AFG and a Scope and a saturated absorption (SAS) to control our lasers.

2 Simulation of the D2 transition line of Rubidium

2.1 Interest of this project

Within the team, We were creating photon-photon interactions mainly in two ways: by creating polaritons in micro-cavity (exciton type : photon, electron / gap), or by hot alkaline atomic vapors (here rubidium). It is on the last one I have been affiliated. The interests of using hot rubidium vapors are multiple. The first one is that they provide strong non-linear effects for a "low cost". We only need a macroscopic rubidium cell (length from 1 to 10 cm) and a laser to create see interactions and create a BEC. Another reason is that we know how to caracterize our medium and what are the parameters of our interactions.

In my case, I had to model the transmission spectrum of the Rubidium cell, depending on its parameters : cell length, temperature and mixture. Indeed we will see that we have two kinds of cells. The first one is composed by 28% ^{87}Rb and 72% of ^{85}Rb, and the second one is only composed by ^{87}Rb. An interest of doing this is knowing the temperature during the measurement. Indeed it is a important parameter to know and the more accurate way to get the temperature is by fitting the experimental spectrum with the theoretical. We will see section 5 that temperature can influence non-linear refractive index.

In this section, I will first expose the transition D-line of the rubidium isotopes and why we can use them. Then I will present the model.

2.2 Fine and hyperfine Rubidium structures

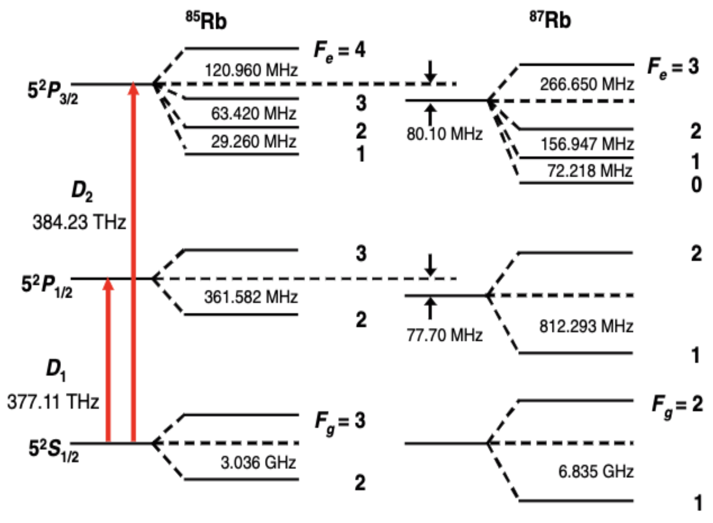


FIGURE 5: hyperfine structures of isotopes ^{87}Rb and ^{85}Rb for transitions D1 and D2 from [9]. As you can see, for the two levels up and ground, there is different layers according to its angular momentum. We will see later what we can do with this configuration and some hypothesis. See [4] for more details.

we can define the transmission $\mathcal{T}$, depending on the detunning $\Delta = \omega - \omega_i$ where ω_i a the transition frequency associated to the values in figure-5.
We know with Beer-Lambert law that the intensity beam is given by $I(z) = I_0 exp[-\alpha(\nu, T)z]$ where $\alpha(\nu, T)$ is the absorption coefficient. we can transform this equation easily as :

$$\frac{I(z)}{I_0} = exp[-\alpha(\nu, T)z] \tag{2.1}$$

where z is the propagation direction, I_0 the input intensity, $\nu = \frac{2\pi}{\omega}$ the frequency, T the cell temperature. We can see that for each transition frequency we have an absorption coefficient α_i so we have to sum them to get the transmission :

$$\mathcal{T} = exp((-\sum \alpha_i)L) \tag{2.2}$$

where L is the length cell. We can see [9] that $\alpha = kIm[\chi(\Delta)]$ where $k = n\frac{2\pi}{\lambda}$ is the wave number and χ is the dielectric susceptibility equal to :

$$\chi(\Delta) = \frac{\sqrt{\pi} d^2 C_f^2 \mathcal{N}}{\sigma} \frac{1}{2(2\mathcal{I}+1)} \frac{1}{\hbar\epsilon_0} V(\frac{\gamma - i\Delta}{\sigma}) \tag{2.3}$$

You can find all the values in [9].

2.2.1 definition of the different values from $\chi(\Delta)$

d takes two values depending on the D line $d = 5.177ea0$ for D_1 and $d = 5.177ea0$ for D_2, wherea_0 is the Bohr radius.
C_f^2 is the transition coefficient and depends on the type of transition line and its angular momentum. See Appendix B from [9] with table 1 to get the values.
$\gamma = 0.5 * 3.81138e7Hz$ is the atomic decay
$\mathcal{N}$ is the atomic density number that is also explained in [9], Appendix A.
$\sigma = ku$ where $u = \sqrt{2k_BT/M}$ with M the atomic mass.
$\mathcal{I}$ is the nuclear spin value = 3/2 or 5/2 for respectively ^{87}Rb and ^{85}Rb $\hbar$ is the Plank-Einstein constant and ϵ_0 is the void permitivity.

The last function I want to talk about in this section is the V function named Voigt function. this is a function that fits with the velocity distribution. We have to add it to explain the Doppler broadening. What happens in the cell is that the Laser encounter atoms with a velocity that can be, in 1D in the same direction that the laser or in the opposite direction. We can traduce this by changing the detunning $\Delta = \omega - \omega_i$ by $\Delta - \sigma$

The related frequency is then a bit more or less than the absolute transition frequency and we get a gaussian profile for the velocity distribution that can be written with the Voigt function. The mainly reason why I use this function is because for the numerical simulations there is a function that does this so it is easier to model. We will see it in the section code.

This is why we setup saturated absorption (SAS) to measure laser frequency. If the laser encounter an atom in the two directions z and $-z$ then the related frequency measured is $\Delta - \sigma + \sigma$.

2.3 results

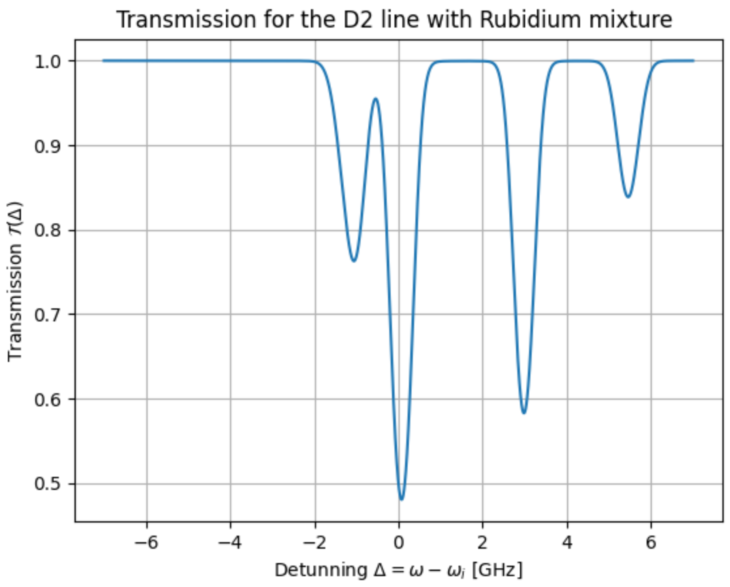


FIGURE 6: numerical Transmission plot for D2 line with length cell L = 7.5 cm, Temperature T = 393K and rubidium ^{87}Rb fraction frac = 0.28 | equation 2.2

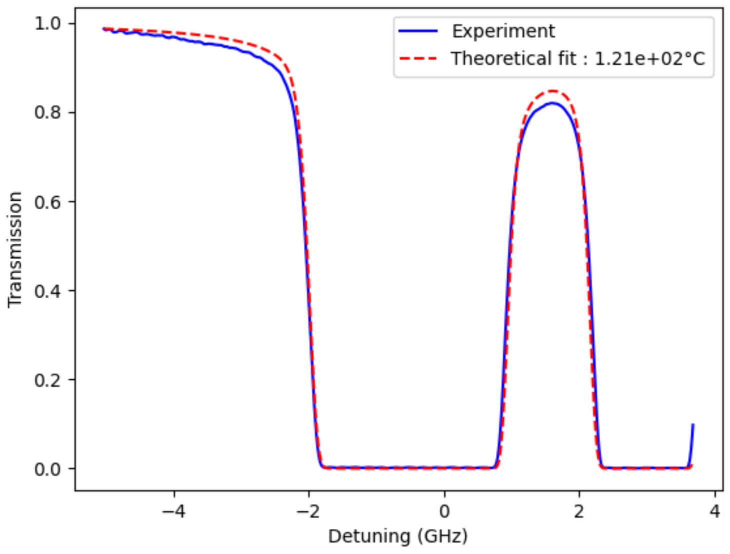


FIGURE 7: After numerical fit, we finally get the temperature for a mixture Rubidium cell with length L = 1 cm.

2.4 coding the application

The application goal is to rapidly explain how does our medium work with the setup parameters : length, temperature and rubidium fraction. We wanted to build with Mathematica an animation interactive. But then the possibility to reuse with classes the model and not all the app was pretty attractive. I defined in this sense my code structure the same as the first section. One class to model our object "transmission", that returns the transmission value computed. One class that creates a UI.

2.5 D2 line Class

2.5.1 Init() function

A good way to build our class is to define, in the init all the parameters and constants

```python
class SpectrumRubidiumD2Line:
    def __init__(self, gamma: float = 3.81138e7, k: float = 2 * np.pi / 7.807864080702083e-7,
                 d: float = 4.4003140382156403e-29, M: float = 1.4099931997e-25, beta: float = 1.03e
    -13):
        """

        Parameters
        ----------
        gamma : decay time for the stage |e> -> |g>
        k : laser wave number
        d : electric dipole
        M : atomic mass
        beta : [SI] impact self broadening
        """
        self.!!h!! = cst.hbar
        self.eps_zero = cst.epsilon_0
        self.beta = beta
        self.gamma = gamma
        self.k = k
        self.d = d
        self.M = M

        # Rb 87

        # F_g = 2 --> F_e = 1, 2, 3
        self.w_i = -2735.05e6 + 1307.87e6
        self.w_ii = -2578.11e6 + 1307.87e6
        self.w_iii = -2311.26e6 + 1307.87e6

        # F_g = 1 --> F_e = 0, 1, 2
        self.w_ib = 4027.403e6 + 1307.87e6
        self.w_iib = 4099.625e6 + 1307.87e6
        self.w_iiib = 4256.57e6 + 1307.87e6

        # Rb 85

        # F_g = 3 --> F_e = 2, 3, 4
        self.w_j = -1371.29e6 + 1307.87e6
        self.w_jj = -1307.87e6 + 1307.87e6
        self.w_jjj = -1186.91e6 + 1307.87e6

        # F_g = 2 --> F_e = 1, 2, 3
        self.w_jb = 1635.454e6 + 1307.87e6
        self.w_jjb = 1664.714e6 + 1307.87e6
        self.w_jjjb = 1728.134e6 + 1307.87e6
```

LISTING 14: Init function of the transition D2 line

2.5.2 methods

Each method is an argument. For example we have a method that returns the susceptibility χ :

```python
    def susceptibility(self, C_f: float, frac: int, temp: int, transition_frequency: float,
                       deg: int) -> np.array([float]):
        """

        :param C_f: C-f^2 transition coefficient, depending on the quantum number F and isotope
        :param frac: % of Rb 87 in the cell. frac in [0-100]
        :param temp: temperature T[K] (local var)
        :param transition_frequency: possible values are instance variables in init that depend of the
     quantum number F.
        :param deg: degeneration degree, depending of the isotope. 8 for 87 and 12 for 85
        :return: absorption coefficient
        """

        N: float = self.atomNumber(frac, temp, deg)
        delta: np.array([int]) = 2 * np.pi * (self.detuning() - transition_frequency)
        voigt_arg = (0.5*(self.gamma+2*np.pi*self.beta*self.Ndensity(temp=temp, deg=deg)) - 1j * delta
    ) / self.sigma(temp)
        suscepti = C_f * (N * (self.d ** 2) * np.sqrt(np.pi) / (self.!!h!! * self.eps_zero * self.
    sigma(temp))) * \
                   self.voigtProfile(voigt_arg).imag
        return suscepti
```

LISTING 15: susceptibility method from class SpectrumRubidiumD2Line

2.6 GUI

As the first GUI, we use the module PyQt5 and the software Qt designer to generate a python code that creates our GUI. We connect each widget to our methods and we get the figure-8.

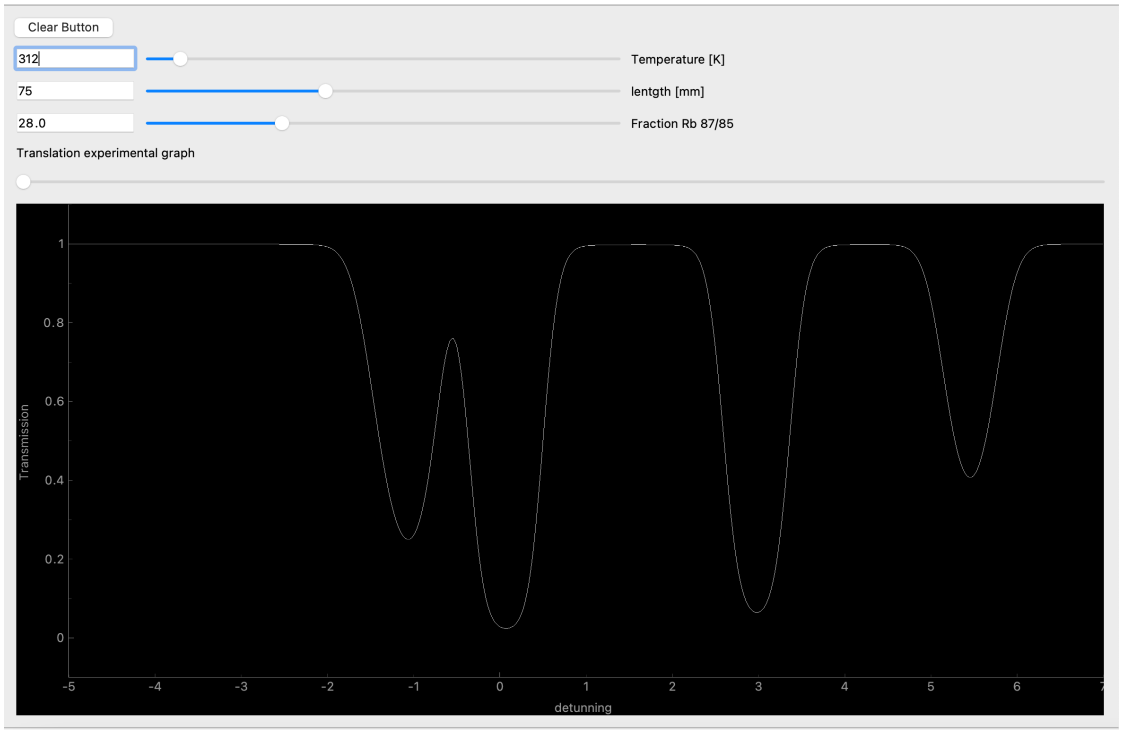


FIGURE 8

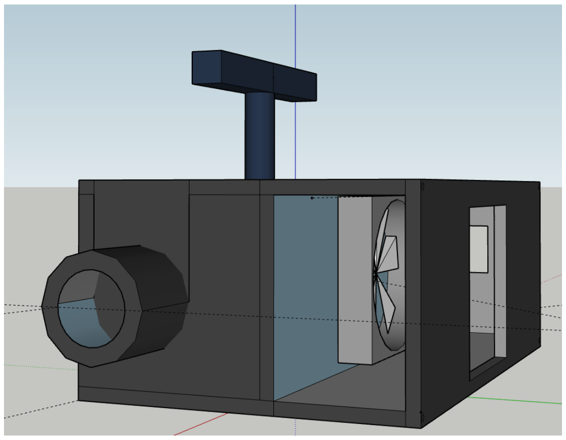
3 Cell oven

I was led to build an alkaline gas cell oven which would allow to change quickly the cell and thus the parameter of the experiment. The oven had to meet a precise set of specifications:

1. Be able to mount it on an optical table (Do not take up too much space on the optical table.
2. !heat the cell to 180 degrees.
3. Isolate from air and magnetic fluctuations the cell.
4. Warm fast the cell with homogeneous temperature in.
5. !have a leak point to condense the liquid phase of the atom.

3.1 Sketchup model

First oven model designed on Google Sketchup. We can see two tanks. the first one is composed of a cylinder in which the cell is placed. We can place the cell thanks to the handle located on the top of the oven. On the figure 9(a) we can see a fan disposed on the side of the cylinder. the idea was to make a rotating heat oven. I tried this model with the figure 10. But It was not conclusive. the oven was only heated to 90 degrees, that is clearly not enough to experiment with. subcaption


((a))

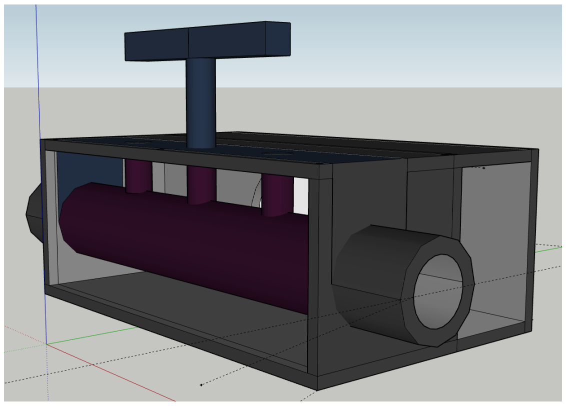

((b))

Figure 9

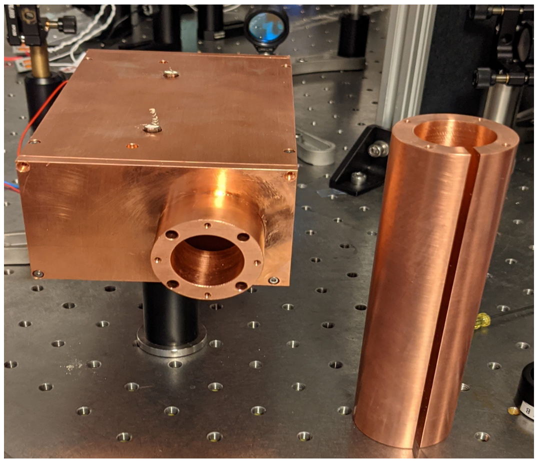
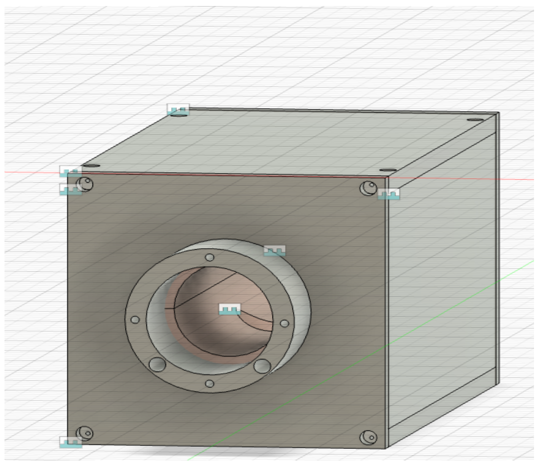


Figure 10

3.2 Fusion 360 model

Therefore, I decided to produce a new oven, thinner and more optimised for heating. I removed the fan and added insulating materials like glass wool or ceramic on the trims. I used fusion 360 as software to design it. This is basically as SOLIDWORKS but it is perhaps easier to handle. As you can see figure 12, the leak point is working. We can also easily


((A))

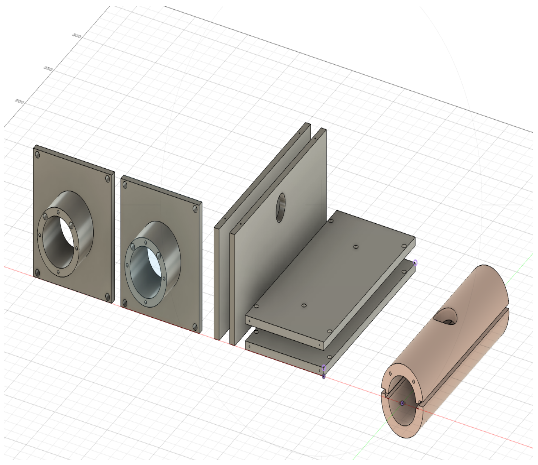

((B))

FIGURE 11

reach 150 degrees or more. Data taken for the experimental transmissions on next sections comes from this oven. There are two problems, however. The first is that the oven takes 1 hour to reach the first temperature (around 10 degrees). This is the compromise we had to make. Also, I didn't have enough heat resistant wires available which could break them because the metal was soft. However there was no danger.

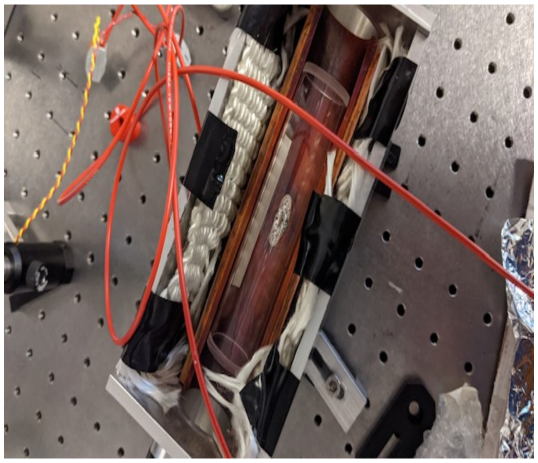


FIGURE 12: Photo taken just after heating to 133°C. You can see the leakage point located downwards as expected, which avoids fogging up the cell windows. We added glass wool to the walls of the oven to retain heat and optimise the power/temperature coupling.

3.3 Bridge

After making our oven we finished the preparations for the experiments. We have an oven that heats the cells efficiently, we have a method to determine the cell temperature and we have a control on the laser power and frequency.

4 Measure of non-linear refractive index.

4.1 Introduction

The aim of this section is to have optical control over the interactions of the medium. We have seen in section 2 that the absorption of a photon by the electron of the Rubidium atom causes a non linear interaction. We also found out which parameters were responsible for these interactions. Temperature, cell length and isotope concentration. Now we want to measure experimentally the impact of these parameters on the non-linear refractive index n_2. An immediate application of this result is, for example, to characterise the non-linear Schrödinger equation given in (0.2).

4.2 Theory

First, we need to define the non-linear refractive index as $n = n_0 + \Delta n$ where n_0 is the refractive index in the void. A good way to approach the non-linear refractive index is first to define it as $\Delta n = n_2 I$ [5]. But this model can find its limits. Indeed it doesn't take into account the saturation of the medium at high power. To correct that we need to introduce I_{sat} as :

$$\Delta n = \frac{n_2 \mathcal{I}}{1 + \frac{\mathcal{I}}{\mathcal{I}_{sat}}} \tag{4.1}$$

where $\mathcal{I} = \frac{2\mathcal{P}}{\pi W_0^2}\left(\frac{1-e^{-\alpha L}}{\alpha L}\right)$ is the laser intensity with $\mathcal{P}$ the laser power, α the absorption coefficient and W_0 the beam waist. And $\mathcal{I}_{sat}$ is the saturation part of intensity that we measure.

$$n_2 = \frac{3}{4}\frac{Re(\chi^{(3)})}{\epsilon_0 n_0 c} \tag{4.2}$$

where $\chi^{(3)}$ is the third term of dielectric susceptibility, seen in section 2. Moreover, we have by definition :

$$\Delta n = \frac{\phi_{NL}}{k_0 L} \tag{4.3}$$

4.3 Setup and method

4.3.1 setup

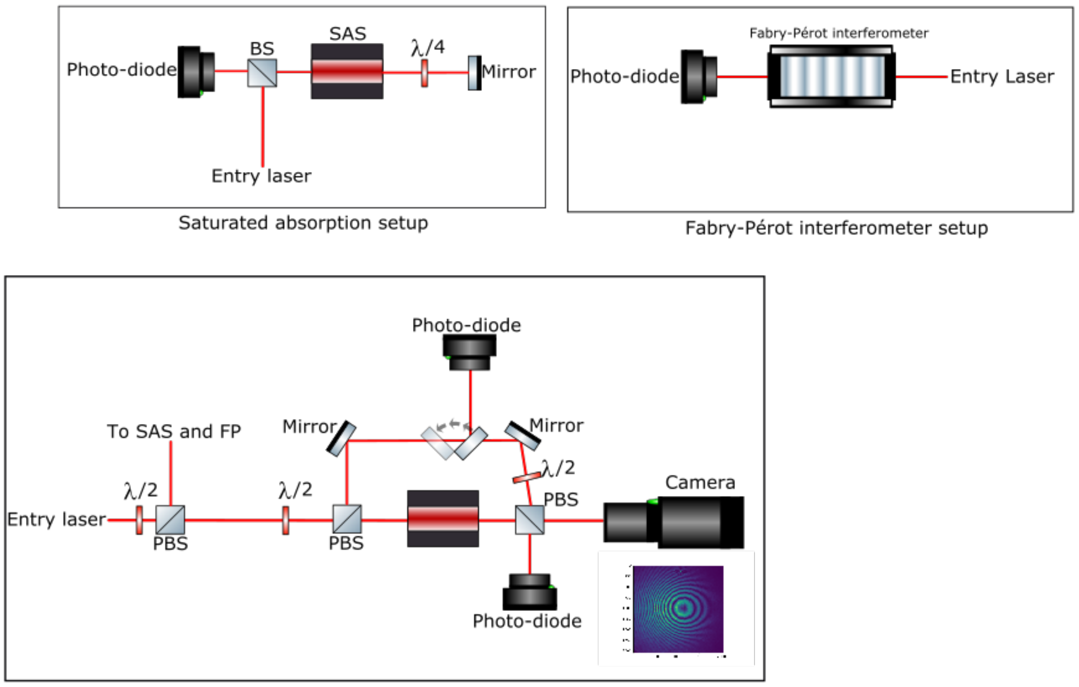


FIGURE 13: Mach-Zendher interferometer. Photodiodes are used to measure the transmission to determine the temperature. The Fabry-Perot interferometer and the saturated absorption are used to calibrate the abcissa axis for the temperature fit

4.3.2 method

The idea is to recover the intensity of the image, which is a squared sum of electric fields (superposition principle). Credits to Guillaume [6] for the calculus.

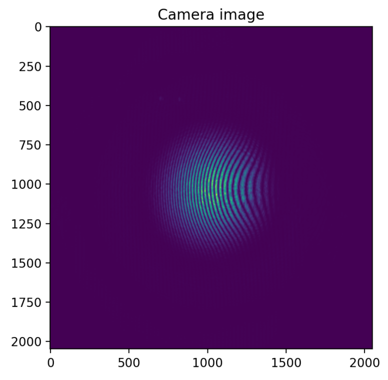
$$I_{camera}(\mathbf{r}) \propto |\mathcal{E}_S(\mathbf{r})e^{i(\mathbf{k}_S\mathbf{r}+\phi(\mathbf{r}))} + \mathcal{E}_r(\mathbf{r})e^{i\mathbf{k}_r\mathbf{r}}|^2 = |\mathcal{E}_S(\mathbf{r})e^{i\phi(\mathbf{r})} + \mathcal{E}_r(\mathbf{r})e^{ik_x x}|^2 \tag{4.4}$$

$$= \underbrace{I_S(\mathbf{r}) + I_r(\mathbf{r})}_{\text{continuous part}} + \underbrace{n_0\epsilon_0 c Re(\mathcal{E}_S(\mathbf{r})\mathcal{E}_r(\mathbf{r})e^{i\phi(\mathbf{r})+k_x x})}_{\mathcal{V}(\mathbf{r})} \tag{4.5}$$

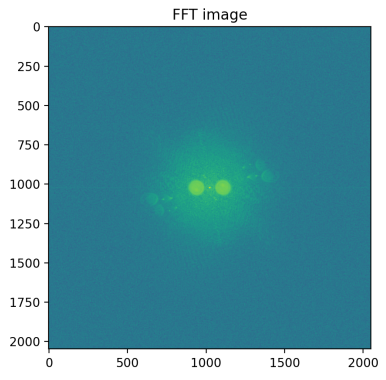
Where ϕ is the phase accumulated due to the non linearity. And this is the part we want to measure. Then $\mathcal{V}$ is the part containing the information of the phase shift. We apply a Fourier transform to this part and finally get :

$$\Phi(\mathbf{r}) = \phi(\mathbf{r}) + k_x x = \underbrace{\phi_{NL}(\mathbf{r})}_{\text{nonlinear phase}} + \overbrace{\phi_0}^{\text{geometrical phase}} + \underbrace{k_x x}_{\text{off-axis contribution}} \tag{4.6}$$

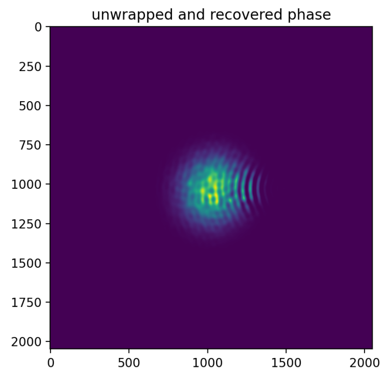
Now it is just a matter of subtracting the information that does not interest us . We just keep the non linear phase. Now we are able to measure the non linear phase and to combine it with (4.3) to get the non linear refractive index. Let's show you have does it work with images :



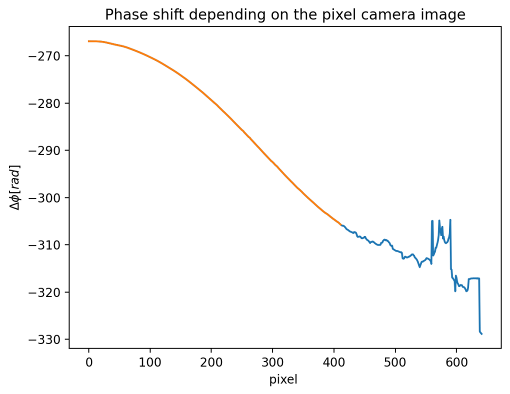
((A))



((B))



((C))



((D))

FIGURE 14: 14(a): interferogram image taken from camera. 14(b) We apply a Fourier transform to the image and we centre it. 14(c) We select the satellite peak on right and we apply an inverse FFT. Then we unwrap the image and get this figure showing the unwrapped phase. 14(d) Finally we are able to get the phase.

4.3.3 automation

We need to take a picture of the interferogram and change the beam intensity at the same time. For that we need to :

1. Measure the positions where we get p_max and p_min where the waveplate $\lambda/2$ let pass the maximum intensity I_{max} and the minimum intensity $I_{min} = 0$

2. Go to position P_{max}

3. Create two threads in the code. One that controls the waveplate and the other one that controls the camera

4. start measuring and turn the waveplate at the same time from p_{min} to p_{max}

5. check if the data set is good by summing all the frames on one column (see figure)

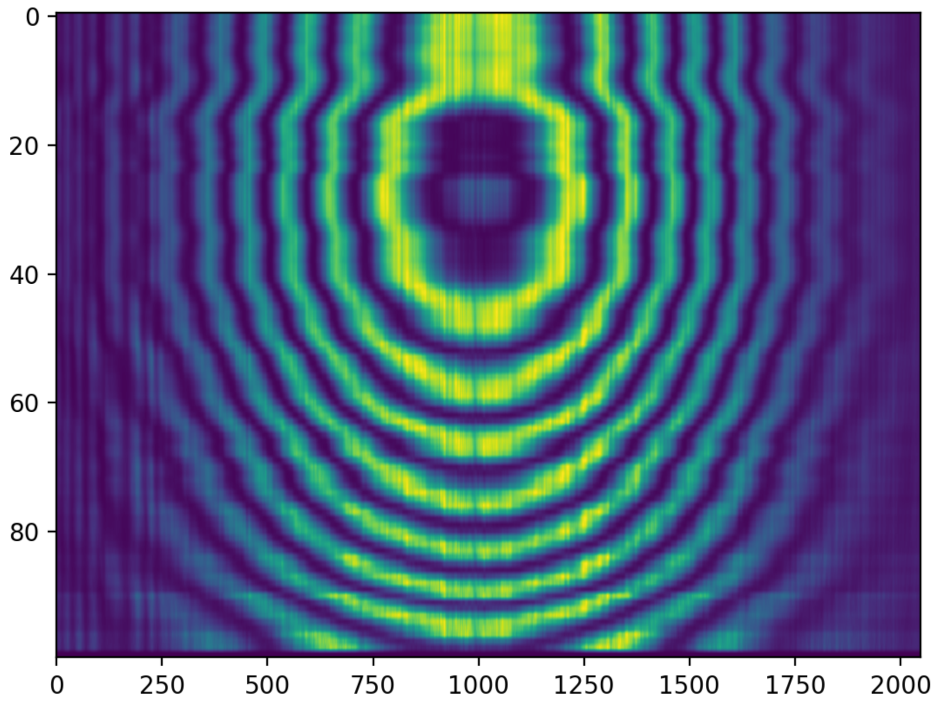


Figure 15: Sum of 100 frames. We take the column 1024 of every frame and we sum them. every peak intensity corresponds to a 2π rotation of the phase.

4.4 results

Finally we obtain successful experimental results. We can see that the data respect the fit. We can still see some gaps in some places of the curve. This is due to images that could not be analysed because of air fluctuations or reflections on the camera. We can then add digital processing to remove these points. In all cases, we still have enough points to analyse the nature of the curve.

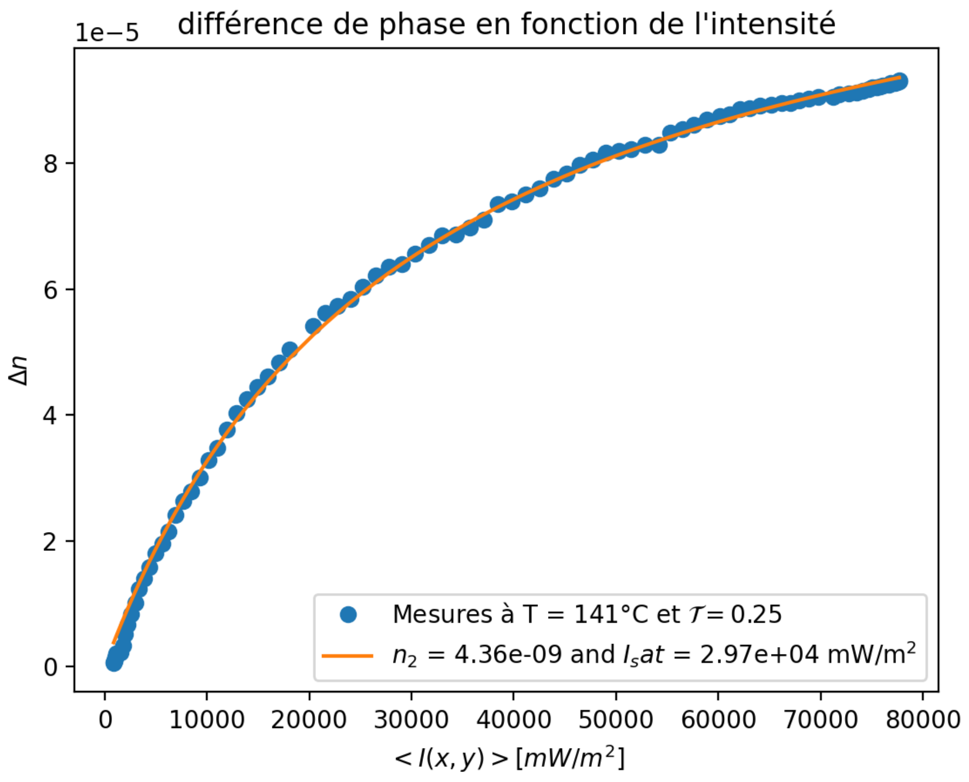


FIGURE 16: evolution of the non linear refractive index shift, as a function of the intensity. We see that our fit parameters are n_2 and I_{sat}. Data set taken with a cell at $T + 141C$, a length of 1cm. A laser Power of 453 mW, a transmission of 0.25, a waist of 0.0014 and a laser wave length of 780 nm.

4.5 Conclusion

Thus we have developed an universal experimental method for measuring the non-linear refractive index. Now we just have to change the parameters of the experiment to see the impact of these on the interactions of the medium. For this, we needed all the engineering presented in the previous sections. The temperature measurement was done using the class created in section 2 for the fit. We had control of the laser from the GUI to change the transmission of the cell. Finally the different cells were heated with the oven designed in section 3.

5 Conclusion

In conclusion, this gap year has been very enriching, both in terms of hard and soft skills. Indeed, I was able to acquire a very good level of scientific programming in Python. I first learned how to program in object-oriented programming. I was able to develop a graphical interface to control a laser model in power and frequency. On the one hand I responded to a problematic, and on the other hand, become familiar with complex programming projects. This required a lot of methodology and I was able to use the Notion app to organise myself.

But of course, I had to learn how to use Python in terms of physics. For example, I had to model the transmission of a rubidium cell according to its parameters. The aim was to show the newcomers to the team the effect of these parameters on the absorption (2.3) of the cell and therefore, the interactions linked to them, and also to provide the team with a theoretical model of the transmission to measure the temperature.

Finally, I was brought to work on the physics of quantum fluids of light. This is still a young science from an experimental point of view. We were therefore working on characterising the medium. In particular, the propagation of the fluid in the cell. As a reminder, this is governed by the Gross-Pitaevskii equation (0.3), which is analogous to the non-linear Schrödinger equation (0.1), (0.2). I was thus able to set up experiments, in particular for the non-linear refractive index, and to analyse the data by comparing them with theoretical models.

This gap year fits very well with my professional project which is to do research. Indeed, my goal is to work in quantum information, more precisely in the architecture of quantum processors, from an experimental point of view.

Many disciplines are trying to associate themselves with this, notably quantum optics, with single photon sources and multi-mode photons. And programming in Python remains essential and widely adopted in the field. I am also interested in quantum information storage, where ultra-cold atoms and BECs are being explored. Thus my skills in quantum optics acquired during this thesis can be put at the disposal of these research fields.

References

[1] Laser datasheet.

[2] J. C. E Carcolé and S. Bosch. Nonequilibrium precondensation of classical waves in two dimensions propagating through atomic vapors. 2018.

[3] Q. Fontaine. Observation of the bogoliubov dispersion relation in a fluid of light. *Physical Review Letters*, 2018.

[4] Q. Fontaine. Paraxial fluid of light in hot atomic vapors, 2019.

[5] P. S. G. E. B. A. Fontaine Q, Bienaim ´e T and G. Q. *Physical Review*, 2018.

[6] guillaume Brochier. Optical control upon quantum fluids of light, 2021.

[7] C. R. Y. McCormick C F, Solli D R and H. J. *Physical Review*, 2004.

[8] E. G. A. B. Murad Abuzarli, Tom Bienaimé and Q. Glorieux. Blast waves in a paraxial fluid of light. 2021.

[9] H. Paul Siddons. Absolute absorption on rubidium d lines : comparison between theory and experiment. *Journal of Physics*, 2008.

[10] B. H. Wr Callen and R. Pantell. optical patterns of thermally self-defocused light. *Applied Physics Letters*, 1967.