

Física Moderna

Difração de Múltiplas Fendas

Luiz Fernando Gomes de Oliveira- 10/46969

Lucas Severo Alves- 10/0034772

Leonardo- xx/yyyyy

Resumo

Este experimento tem como objetivo apresentar o experimento de Thomas Young - experimento da fenda dupla. Este experimento demonstra como a luz em específico pode se comportar como partícula e onda ao mesmo tempo, o que gera uma grande dificuldade de compreensão de sua composição. Este experimento tem implicações profundas, determinando maior parte da física do século XIX e resultando em várias tentativas para descobrir o éter, ou do meio de propagação da luz. Embora a experiência seja mais notável com a luz, o facto é que este tipo de experiência pode ser realizada com qualquer tipo de onda, tal como a água. Para esse experimento, no entanto, vamos focar no comportamento da luz.

This experiment aims to present the experiment of Thomas Young - double slit experiment. This experiment demonstrates how light can behave as specific particle and wave at the same time, which creates a great difficulty to understand its composition. This experiment had profound implications, determining most of nineteenth century physics and resulting in several attempts to discover the ether, or the medium of light propagation. Though the experiment is most notable with light, the fact is that this sort of experiment can be performed with any type of wave, such as water. For this experiment, however, we'll focus on the behavior of light.

Keywords

Física Moderna, Arduino, Difração de Múltiplas Fendas, Thomas Young, Double-slit experiment, ondas, partículas

SUMÁRIO

1	Introdução	1
1.1	Dualidade	1
2	Criação do Circuito	2
2.1	Ponte de Wheatstone	2
2.2	Abstração da ponte para o circuito final	2
3	Colhendo dados reais - Arduino	3
3.1	Levantando o programa	3
3.1.1	Shield Comum	3
3.1.2	Utilizando o Eclipse para o Arduino	4
3.2	Criação do Código	4
4	Coletando os dados no computador - Python	5
4.1	Python - o início	5
4.2	Lendo os dados	6
4.3	Plotando os dados	6
5	Amostras - laser vermelho	6
6	Amostras - laser verde	8
	Apêndice A: Códigos Fontes	9

Apêndice B: Anexos

12

LISTA DE FIGURAS

1	Difração em fenda simples	1
2	Difração em fenda dupla (com interferência)	1
3	Efeito fotoelétrico	2
4	Ponte de Wheatstone	2
5	Circuito do LDR	2
6	Circuito proposto do LDR	3
7	IDE do Arduino	4
8	Fluxograma do microcontrolador	4
9	Idealização do circuito do LDR	12
10	Fenda dupla	13
11	Difração em fenda simples	14
12	Difração em fenda dupla	15

LISTA DE TABELAS**LISTINGS**

1	Exemplo da função main()	4
2	setup()	5
3	loop()	5
4	Bibliotecas do Python	5
5	Acessando a porta serial	5
6	Lendo dados	6
7	Plotando os dados	6
8	main.c	9
9	uart.py	10

1 INTRODUÇÃO

Se a luz consistisse estritamente de partículas normais ou clássicas, e estas partículas fossem disparadas em linha reta através de uma fenda e deixa-se chegar a uma superfície, do outro lado, seria de esperar um padrão correspondente ao tamanho e à forma da fenda. No entanto, quando esta "experiência de única fenda" é efectivamente realizado, o padrão na superfície (parede) é um padrão de difração em que a luz se espalha. Quanto menor a fenda, maior o ângulo de dissipação.

Da mesma forma, se a luz consistisse de partículas estritamente clássicas e iluminado duas fendas paralelas, o padrão esperado na superfície seria simplesmente a soma das duas fendas de um único padrão. Na realidade, porém, o padrão muda para uma série de faixas claras e escuras. Quando Thomas Young demonstrou pela primeira vez este fenômeno, indicou que a luz consiste de ondas, já que a distribuição de brilho pode ser explicada pela interferência construtiva e interferência destrutiva de frentes de onda. A experiência de Young, realizada no início de 1800, desempenhou um papel vital na aceitação da teoria ondulatória da luz, vencendo a teoria corpuscular da luz, proposta por Isaac Newton, que tinha sido o modelo aceito de propagação da luz nos séculos 17 e 18. No entanto, a descoberta posterior do efeito fotoelétrico demonstrou que, em função das circunstâncias, a luz pode comportar-se como se ela fosse composta de partículas discretas. Estas descobertas aparentemente contraditórias tornou necessário ir além da física clássica e tomar a natureza quântica da luz em conta.

1.1 Dualidade

A teoria quântica nos diz que tanto a luz quanto a matéria consistem de pequenas partículas que apresentam propriedades semelhantes à das ondas. A luz é composta de partículas chamadas de fótons e a matéria é composta de partículas chamadas de elétrons, prótons, nêutrons. É só quando a massa de uma partícula fica pequena o suficiente que suas propriedades ondulatórias aparecem.

Com o experimento feito por Thomas Young percebemos propriedades de onda na luz, sendo essas a difração e interferência:

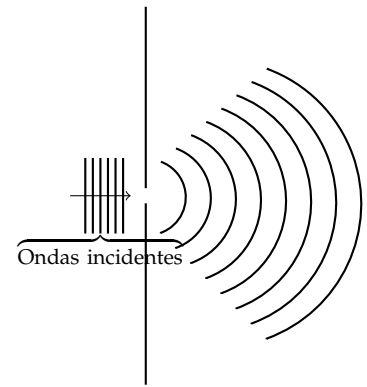


Figura 1: Difração em fenda simples

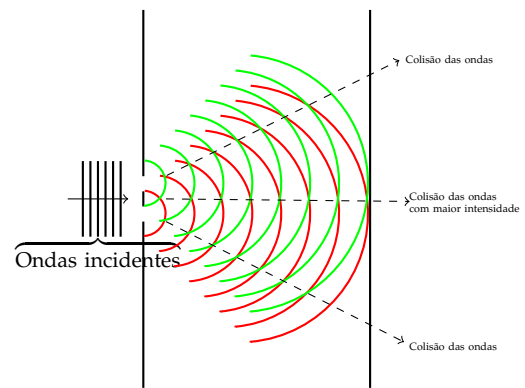


Figura 2: Difração em fenda dupla (com interferência)

Mesmo que a luz seja composta de partículas chamadas fótons, pode-se facilmente mostrar que a luz pode ser considerada uma onda eletromagnética que se propaga com a velocidade da luz. A frequência da luz está relacionada com o seu comprimento de onda de acordo com

$$f = \frac{C}{\lambda} \quad (1)$$

Sendo C a velocidade da luz, λ o seu comprimento de onda, e f a frequência.

Pode parecer bastante óbvio que a luz se comporta como uma onda, mas a prova de que a luz é realmente composta de partículas vem da experiência do efeito fotoelétrico.

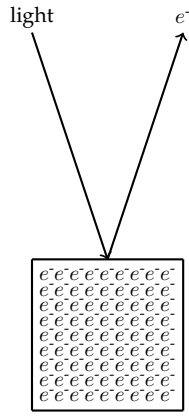


Figura 3: Efeito fotoelétrico

2 CRIAÇÃO DO CIRCUITO

PARA a criação do circuito, precisamos inicialmente entender como funciona uma ponte de *Wheatstone*.

2.1 Ponte de Wheatstone

A ponte de Wheatstone, representada na figura 4, é utilizada quando queremos balancear um circuito com uma resistência desconhecida. A ponte é comumente caracterizada pelo desenho na figura 4, onde pode-se obter uma corrente no amperímetro igual a zero quando a seguinte equação é respeitada:

$$R_1 \cdot R_x = R_3 \cdot R_2 \quad (2)$$

Quando a equação 2 é respeitada, as tensões entre os pontos **A** e **B** é a mesma, de forma que não existe corrente nestes pontos

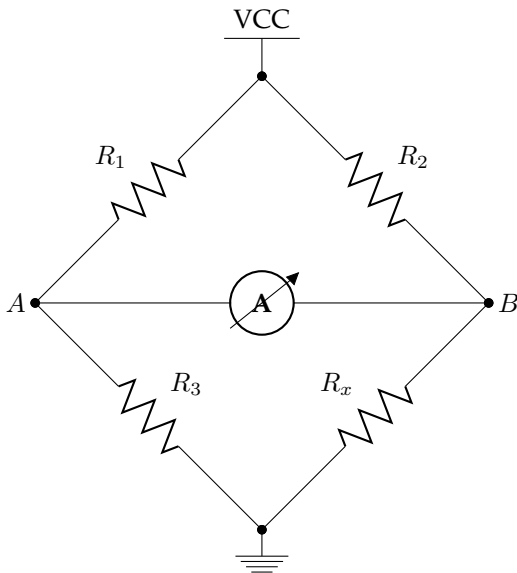


Figura 4: Ponte de Wheatstone

2.2 Abstração da ponte para o circuito final

Com base no conhecimento obtido na sessão 2.1, podemos inferir que no circuito da figura 5 teremos equilíbrio se:

$$R_1 \cdot R_x = R_3 \cdot R_2, \text{ sendo}$$

$$R_x = \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4}$$

Para uma maior facilidade em montar o circuito, podemos considerar que $R_1 = R_2 = R_3 = R$. Outra fator importante é considerar o range da resistência que o *LDR* adota de acordo com a intensidade de luz.

Normalmente, um *LDR* possui resistência em torno de 100Ω quando exposto a luz e $10^6\Omega$ quando em completa escuridão.

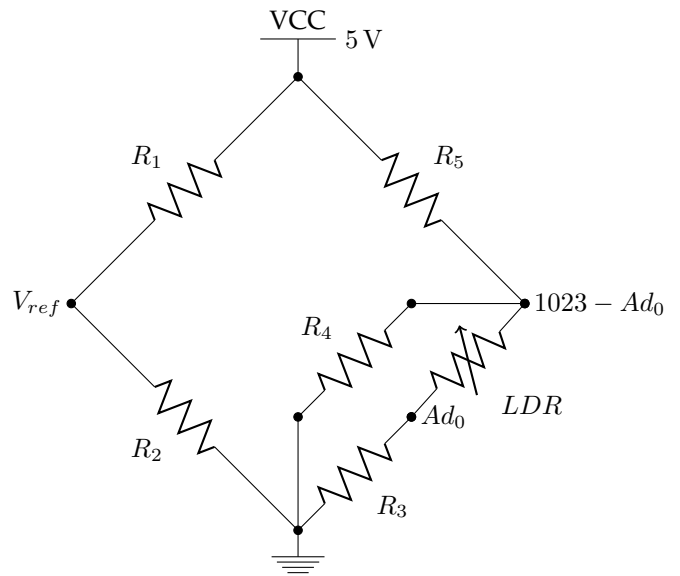


Figura 5: Circuito do LDR

Com base neste range e, adotando um valor para $r = 10K\Omega$, podemos fazer alguns calculos na tentativa de obter alguns valores razoaveis para os resistores R_4 e R_3 que solucionem o sistema proposto:

$$R \cdot R_x = R \cdot R$$

$$R_x = R$$

$$R_x = \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4}$$

$$R_x = \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4} = 10^3$$

$$10^3 = \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4}$$

Tal que: $10^2\Omega < LDR < 10^6\Omega$

Podemos observar que assumindo $R_3 = 8.2 \cdot 10^3 \Omega$, temos o valor de R_4 muito próximo para ambos os casos (10^3 e $1.2 \cdot 10^3$). Assim, uma possibilidade para o circuito final que abranja quase todo o range de resistência do *LDR* é o circuito apresentado na figura 6.

Observe que este circuito permite a leitura de dados de forma distinta. Caso o pino do ADC seja conectado no ponto 1023 – Ad_0 teremos os dados expressos de forma inversa, ou seja, quanto maior a intensidade de luz, mais baixo serão os níveis de tensão coletados pelo microcontrolador. Já no ponto Ad_0 temos a leitura como esperada, ou seja, a amplitude do sinal coletado será diretamente proporcional à intensidade de luz observada pelo sensor.

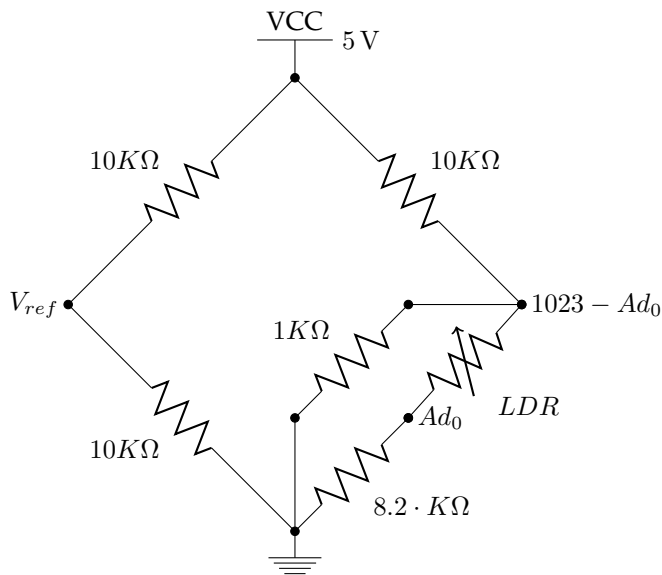


Figura 6: Circuito proposto do LDR

3 COLHENDO DADOS REAIS - ARDUINO

Para coletar os dados expostos será necessário o uso de um microcontrolador que tenha um conversor analógico-digital para mensurar os níveis de intensidade da luz projetados pelo laser após a fenda. Para tal atividade, será utilizado um kit UNO Arduino e os dados serão transmitidos através de uma comunicação serial RS232.

3.1 Levantando o programa

Inicialmente, é importante frisar que o kit do Arduino nada mais é que uma camada de abstração criada acima do microcontrolador AVR. Desta forma, podemos programar nos kits do Arduino de duas formas sob o sistema Linux:

3.1.1 Shield Comum

A forma mais simples de utilizar um kit do Arduino é utilizando a própria IDE oferecida para o kit. A sua instalação é bem simples e fácil de ser feita. Basta entrar com o seguinte comando no terminal:

```
user@DESKTOP:~$ sudo apt-get install arduino
```

Desta forma, podemos acessar a IDE do Arduino, que pode ser observada na figura 7 e escrever o programa tranquilamente, utilizando apenas as funções *loop* e *setup*.

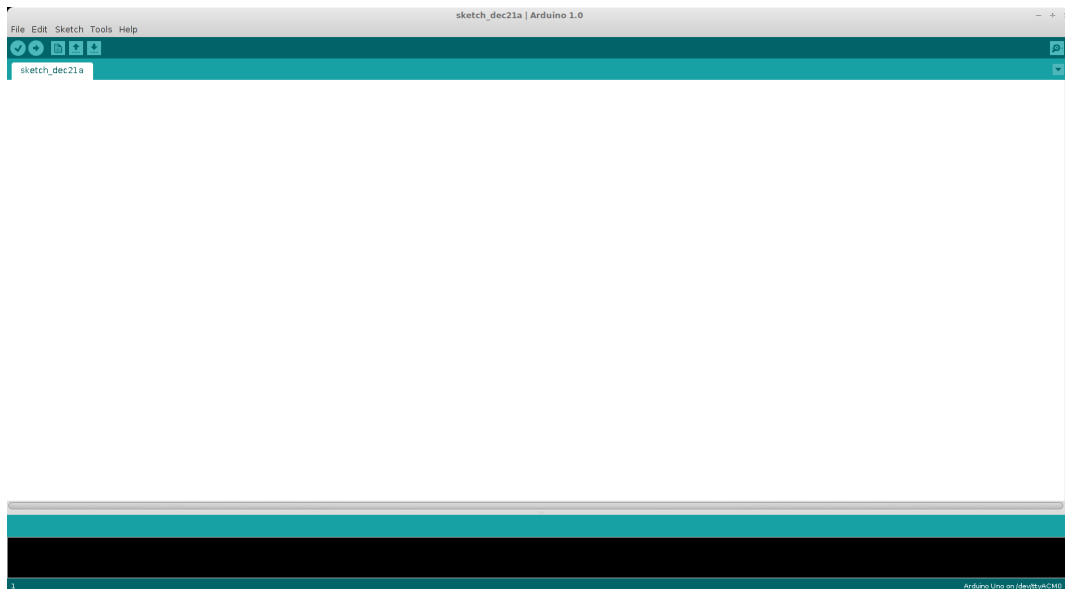


Figura 7: IDE do Arduino

3.1 Utilizando o Eclipse para o Arduino

Porém há alguns grandes inconvenientes ao utilizar a IDE do Arduino.

- Todas as bibliotecas serão compiladas sempre
- O editor de texto deixa muito a desejar
- Não há uma função de auto-completar para facilitar o uso das funções já criadas

Desta forma, uma opção interessante é utilizar o Eclipse para programar o Arduino. Isso requer alguns conceitos mais avançados de programação que não serão descritos neste relatório, que são eles:

- Instalação do *avr-gcc* e *avrdude*
- Criação de uma biblioteca estática com os arquivos em */usr/shared/arduino*.
- Inclusão do diretório */usr/shared/arduino* nas dependências da compilação.
- Configuração da compilação com as flags *-Os, -O3* e *-g*.
- Instalação do plugin de AVR e do AVRDUDE no Eclipse.

Um passo a passo de como proceder com a configuração do Eclipse para uso do Arduino pode ser observado em [?].

Outro fator interessante de se usar o Eclipse é a necessidade de fazer uma função *main*, já que esta é descartada pela IDE do Arduino. Abaixo segue um exemplo de como a função *main* poderia ser escrita para ter um funcionamento semelhante ao da IDE do Arduino, facilitando assim a compatibilidade entre os códigos.

Listing 1: Código de exemplo da função main()

```
#include <Arduino.h>
```

```
int main(void)
{
    init();
    setup();

    while(true)
        loop();

    return 0;
}
```

Observe que este código mantém toda a estrutura básica do Arduino, assim como a inclusão do header *Arduino.h*, necessário para a inclusão das rotinas básicas do Arduino, assim como sua configuração.

3.2 Criação do Código

Com o ambiente de trabalho já configurado, o próximo passo é levantar o programa que funcione de acordo com o diagrama, apresentado na figura 8.

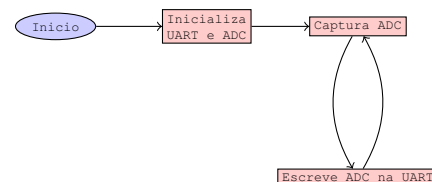


Figura 8: Fluxograma do microcontrolador

Como descrito no diagrama, é necessário inicializar tanto a UART (Comunicação Serial) quanto o ADC (Conversor Analógico-Digital) antes de prender o microcontrolador no laço infinito de captura e exportação dos valores do ADC. Seguindo o contexto de manter a máxima semelhança com o modelo proposto de programação no Arduino, inicializaremos a UART e o ADC na função *setup* da seguinte forma:

Listing 2: Código da função *setup()*

```
void setup()
{
    Serial.begin(9600);
    analogReference(EXTERNAL);
}
```

Observe que a UART é inicializada com rate de 9600, enquanto o conversor é configurado para usar a referência de tensão externa. A velocidade de 9600 é uma faixa de velocidade bem comum em comunicações seriais. Já a referência externa é necessária devido ao circuito que foi proposto na sessão 2.2, onde a figura 6 apresenta uma tensão de referência.

Com o chip já configurado, o próximo passo é configurar a função *loop* para que o processo de captura do valor do ADC e impressão deste valor seja mantido enquanto o chip estiver alimentado.

Listing 3: Código da função *loop()*

```
void loop()
{
    static int i=0;
    char buf[20];

    sprintf(buf, "%d %d\n",
            i++, analogRead(0));
    Serial.print(buf);
}
```

Desta forma, os valores serão impressos na porta serial com a formatação "x y", estando assim prontos para serem plotados no GNUPLOT. No apêndice A pode-se observar na íntegra este código no Listing 8, onde haverá a inclusão de alguns headers e alguns tempos no decorrer do programa para garantir uma maior estabilidade do código.

4 COLETANDO OS DADOS NO COMPUTADOR - PYTHON

Com o processo realizado na sessão 3, qualquer equipamento capacitado para a leitura de comunicação serial poderá realizar a leitura dos dados. Trabalhar com os dados no computador é incrivelmente mais fácil que em um microcontrolador. A taxa de processamento e a memória disponível são extremamente maiores, sem dizer que as camadas de abstração existentes facilitam

- e muito - o nosso trabalho. Isso poderá ser observado através do método que será utilizado para esta etapa. O objetivo é criar um programa em Python que faça a leitura dos dados na porta serial e exporte-os para o GNUPLOT, plotando o gráfico.

4.1 Python - o início

Python é uma linguagem interpretada orientada à objetos que, normalmente, já vem pré-instalada em grande parte das distribuições Linux. É uma linguagem consideravelmente fácil de ser aprendida e que poder ser incrivelmente útil e prática. Para este nosso experimento, utilizaremos uma biblioteca do Python chamada *[?]pyserial*, uma biblioteca voltada para a recepção e envio de dados através de comunicações seriais.

Inicialmente, assim como em quase toda linguagem de programação, iniciamos incluindo as bibliotecas que serão necessárias no decorrer do programa:

Listing 4: Inclusão das bibliotecas do Python

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

import serial
import os
import subprocess
import sys
import curses
```

o trecho `#!/usr/bin/python` é muito comum é scripts em ambiente Linux. Quando colocados na primeira linha, este trecho possibilita que ao darmos direitos de execução ao script (torna-lo executável), ele irá procurar pelo binário, em `/usr/bin/python` no caso, para ser executado. Essa linha nos permite não ter que dar o trabalho de invocar o programa python, já que o script se encarregaria disto. A linha `# -*- coding: utf-8 -*-` por outro lado permite que palavras com acentuação sejam inseridas no programa sem que haja falha de codificação. Devido aos avisos e interação com o usuário estarem todos em português, é interessante a inserção desta linha para uma maior compreensão e qualidade do programa.

Em seguida, devemos tentar abrir a porta na qual desejamos escutar. Para uma maior portabilidade do código, o programa foi montado de forma que o parâmetro enviado junto com a inicialização do script seria o endereço desejado, como pode-se ver logo abaixo:

Listing 5: Acessando a porta serial

```
try:
    ser = serial.Serial(sys.argv[1], 9600)
    ser.open()
except IndexError:
    print "Passe uma porta existente"
```

```

        com o parametro!"
    sys.exit(0)

if ser.isOpen() & ser.readable():
    f=file('./saida.txt','w')
    fx=file('./trabalhada.dot','w')
    fx.write("(0,0) ")
    print "Porta aberta!"
else:
    print "Porta nao acessivel"
    sys.exit(0)

```

Com um conhecimento básico em orientação à objetos, conseguimos ver o quanto Python é intuitivo. Observe que o programa esta atrelado a uma velocidade de comunicação de 9600, assim como o código do Arduíno foi feito, na sessão 3.2. Logo em seguida é feito um simples tratamento de erro para verificar se a porta encontra-se aberta e disponível para leitura.

4.2 Lendo os dados

Caso o programa consiga abrir a porta, devemos então começar a lê-la. Porém em momento algum foi dito quantos dados seriam lidos pela porta serial. Uma forma banal de contornar isto seria arbitrar um valor qualquer de leituras, mas há uma forma mais refinada de contornar este problema.

Assim como em C, podemos trabalhar com interrupções do sistemas em Python. O que faremos então é manter o código preso em um laço infinito lendo as informações e gravando-as em um arquivo, enquanto não ocorrer uma interrupção do teclado (*KeyboardInterrupt*).

Listing 6: Lendo dados ininterruptamente

```

print "Aperte Ctrl+C para terminar a
      leitura\n\n"

while True:
    try:
        msg = ser.readline()
        print msg,
        stdscr.refresh()
    except KeyboardInterrupt:
        break
    finally:
        try:
            f.write(msg)
            vsel = eval(msg.split(" ")[0])
                    /10.0
            fx.write("-- (%.2f,%.2f) " % (
                vsel, eval(msg.split(" ")
                    [1])/100.0 ) )
        except:
            try:
                if len(msg) > 1:
                    print "\rFalha na
                        sincronização\

```

```

nFoi capturado
contudo inválido \
n" + msg

except:
    print "\rNão foi capturado
        nada!"
    ser.close()
    f.close()
    fx.close()
    sys.exit(0)

```

```

print "\r \nFim da leitura!"
ser.close()

```

Indo além de capturar exceções apenas para a interrupção do teclado, também foi feito uma verificação do conteúdo capturado. Assim garante que o conteúdo que será plotado esta na formatação desejada e não possui nenhum carácter inválido.

4.3 Plotando os dados

Observe que após este passo teremos um arquivo com todos os dados coletados e já com a formatação desejada pelo GNUPLOT, tudo o que temos que fazer é invocar o GNUPLOT e plotar o arquivo, algo que podemos fazer facilmente dentro do próprio Python:

Listing 7: Plotando os dados através do Python

```

proc = subprocess.Popen(['gnuplot', '-p'],
                        shell=True,
                        stdin=subprocess.
                            PIPE,
                        )

proc.stdin.write("plot 'saida.txt' with
                lines\n")
proc.stdin.write("pause mouse\n")
proc.stdin.write("quit\n")

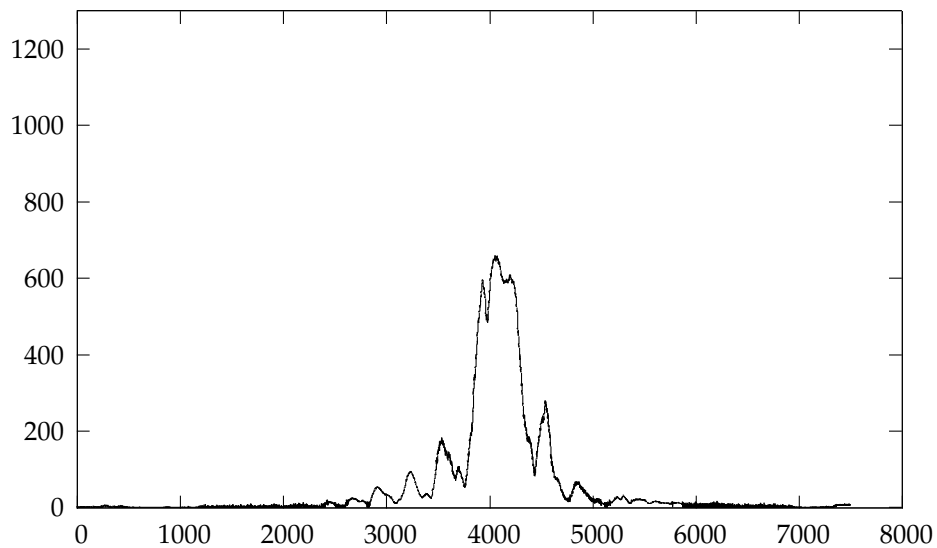
```

Com isso, teremos ao final do script um gráfico plotado na tela com os valores coletados pela porta serial. No apêndice A pode-se observar na integra este código no Listing 9, onde haverá a inclusão de alguns comentários no código e interação com o usuário. No programa final, parte do conteúdo impresso na tela é feito através da biblioteca *curses*, com o objetivo de tornar o programa mais intuitivo para o usuário e ter maior controle do fluxo de saída do programa. Assim sendo, muitos dos trechos citados nesta sessão, terão em sua versão final, em Listing 9, o uso do método *stdscr.addstr* para escrita na tela, no local de **print**.

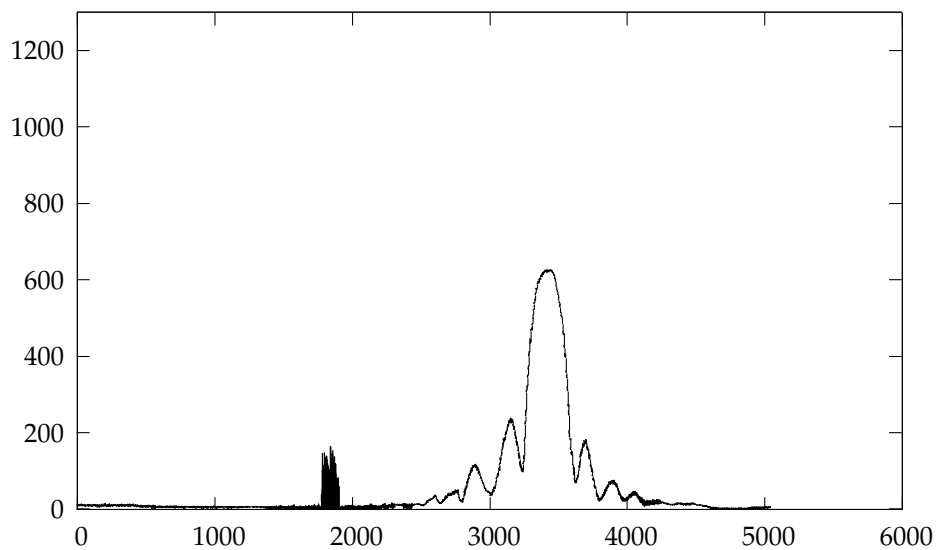
5 AMOSTRAS - LASER VERMELHO

Distancia da fresta ao objeto = 872.0 ± 1 cm
 Largura do laser no objeto = $11.35 \pm 5 \times 10^{-2}$ mm

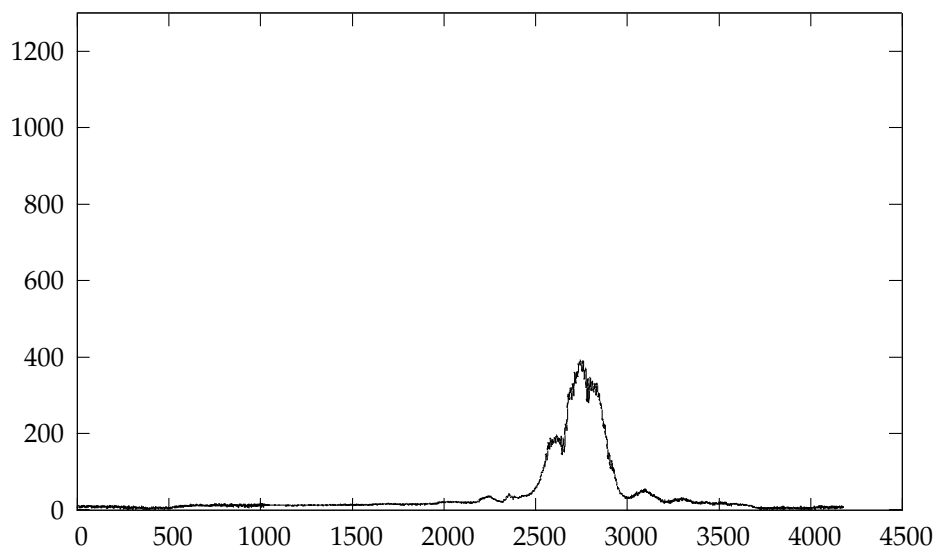
1) Verificação da colheita do sinal na fenda dupla.



2) Colheita do sinal na fenda dupla - houve uma leve falha no início devido à localização do sensor em frente ao sinal da difração

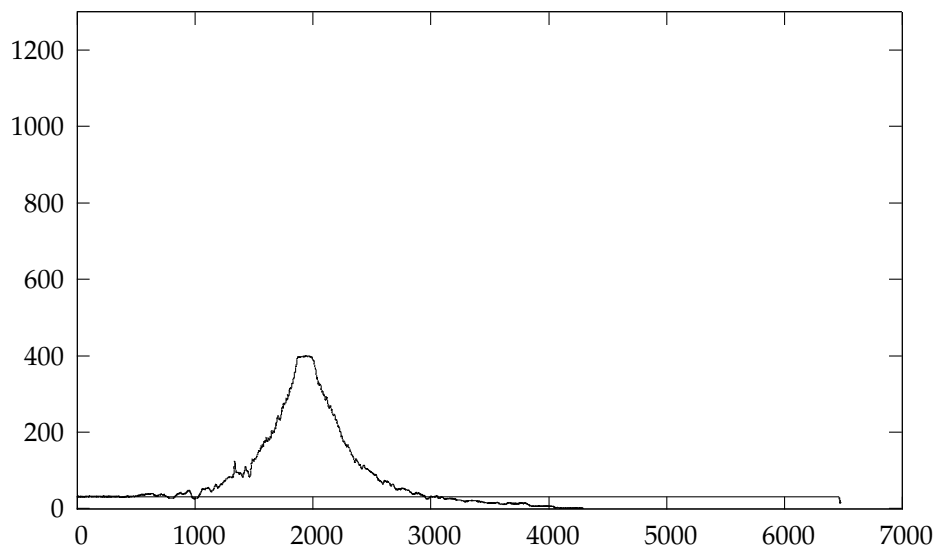


3) Colheita do sinal na fresta simples.

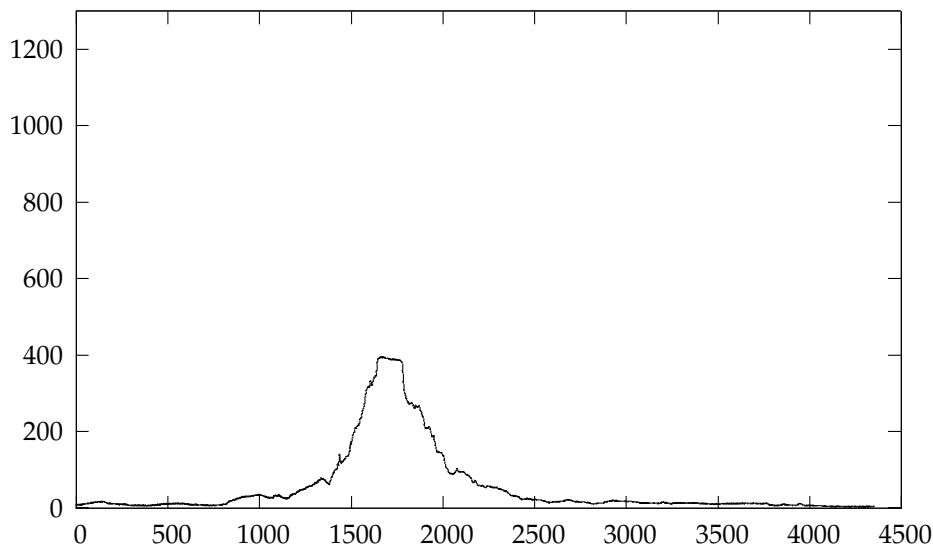


6 AMOSTRAS - LASER VERDE

1) Fenda simples, laser verde



2) Fenda dupla, laser verde



APÊNDICE A CÓDIGOS FONTES

Listing 8: Código principal do Microcontrolador

```

1 /**
2  * Test model for Arduino - UNO
3  */
4
5
6 #include <Arduino.h>
7 // #include <SD/SD.h>
8 #include <SPI.h> //inclusão da biblioteca SPI
9 #include <Ethernet/Ethernet.h> //inclusão da
    biblioteca Ethernet
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 /// The setup() method runs once, when the
    sketch starts
14 void setup()
15 {
16     // initialize the default debug output:
17     Serial.begin(9600);
18     //Serial.print("#Initializing serial
        support...if you hear me, debug is
        ready!\n");
19     // Seeding the rand
20     srand(9600);
21
22 }
23
24 /// the loop() method runs over and over again
25
26 /// 'as long as the Arduino has power
27 void loop()
28 {
29     static int i=0;

```

```

29 char buf[20];
30
31 //sprintf(buf,"%d %d\n",i++,5*analogRead
    (0)/1023 ); //! Valores convertidos
32 sprintf(buf,"%d %d\n",i++,analogRead(0) );
    //! valores brutos
33 Serial.print(buf);
34 //delay(1000); // wait
    for a second
35 }
36
37 /**
38  * This is the standard main function. On
    Arduino, it's NEVER changes.
39  * @return the standard returns that the main
    should return.
40  */
41 int main(void)
42 {
43
44     init(); //! Initialize the pins I/O and
        standard configures.
45     setup(); //! Call the user's setup
        function.
46
47     //! The eternal loop
48     while(true) {
49         loop();
50     }
51
52     return 0;
53 }

```

Listing 9: Código principal de exibição dos dados

```

1#!/usr/bin/python
2# -*- coding:utf-8 -*-
3
4import serial          # metodos para a serial
5import os              # metodos para SO
6import subprocess      # para subprocessos (
    gnuplot)
7import sys             # para ter acesso aos
    parametros
8import curses          # para a biblioteca curses
9
10"""
11import signal          # pegando sinais como em C
12
13def signal_handler(signal, frame):
14    print 'You pressed Ctrl+C!'
15    sys.exit(0)
16
17signal.signal(signal.SIGINT, signal_handler)
18"""
19
20try:
21    ser = serial.Serial(sys.argv[1],9600)
22    ser.open()
23except IndexError:
24    print "Passe uma porta existente com o
        parametro!"
25    print "\teg.: python " + sys.argv[0] + " /
        dev/ttyUSB0"
26    print "\teg.: ./" + sys.argv[0] + " /dev/
        ttyUSB0"
27    sys.exit(0)
28except IOError as e:
29    print ("({})".format(e))
30    sys.exit(0)
31    raise
32
33if ser.isOpen() & ser.readable():
34    f=file('./saida.txt','w')
35    fx=file('./trabalhada.dot','w')
36    fx.write("(0,0) ")
37    print "Porta aberta!"
38
39else:
40    print "Porta não acessível"
41    sys.exit(0)
42
43vsel=0
44l_vsel=1
45teste=True
46#limpa tela e buffer
47#os.system('clear')
48ser.flushInput()
49
50stdscr = curses.initscr()
51curses.noecho()
52
53#print "Aperte Ctrl+C para terminar a leitura\
    n\n"
54stdscr.addstr (0,0,"Aperte Ctrl+C para
        terminar a leitura\n\n",curses.A_BOLD)
55stdscr.addstr (1,0,"!")
56while teste:
57    try:
58        msg = ser.readline()
59        #print msg,
60        stdscr.addstr (3,0,msg,curses.
            A_REVERSE)
61        stdscr.refresh()
62    except KeyboardInterrupt:
63        teste=False
64        break
65
66finally:
67    try:
68        f.write(msg)
69        if (l_vsel == eval(msg.split(" ")
            [0]))):
70            l_vsel=l_vsel+1
71        else:
72            raise
73        vsel = eval(msg.split(" ")[0])
74            /10.0
75        fx.write("-- (%.2f,%.2f) " % (vsel
76            , eval(msg.split(" ")[1])
77            /100.0 ))
78        if (vsel > 5):
79            stdscr.addstr (1,0," ")
80        else:
81            stdscr.addstr (1,0,"!")
82            stdscr.refresh()
83    except:
84        curses.nocbreak(); stdscr.keypad
85            (0); curses.echo()
86        curses.endwin()
87        ser.close()
88        ser.open()
89        f=open('./saida.txt','w')
90        fx=open('./trabalhada.txt','w')
91        try:
92            if len(msg) > 1:
93                print "\rFalha na
                    sincronização!\n\rFoi capturado contudo inv
                    ávido \t" + msg
94                stdscr.addstr (1,0,"!")
95                stdscr.refresh()
96        except:
97            print "\rNão foi capturado
                    nada!"
98            curses.nocbreak(); stdscr.
                keypad(0); curses.echo()
99            curses.endwin()
100            ser.close()
101            f.close()
102            fx.close()
103            sys.exit(0)
104
105stdscr.addstr (5,0, "\r   \nFim da leitura!")
106stdscr.addstr (6,0, "Fechando arquivo \"saida.
        txt\"")
107f.close()
108fx.close()

```

```

108 stdscr.addstr (7,0, "Serial fechada. Abrindo o GNUPLOT")
109 stdscr.refresh()
110
111 proc = subprocess.Popen(['gnuplot', '-p'],
112                          shell=True,
113                          stdin=subprocess.PIPE,
114                          )
115
116 #Margem segura para visualizar os dados
117 proc.stdin.write("set yrange [-1:1300]\n")
118 proc.stdin.write("plot 'saida.txt' with lines\n")
119
119 proc.stdin.write("pause mouse\n")
120 stdscr.addstr (8,0, " Clique na figura para
    fecha-la ", curses.A_BOLD)
121 stdscr.addstr (25,25, " ")
122 stdscr.refresh()
123
124 proc.stdin.write("set terminal latex\n")
125 proc.stdin.write("unset key\n")
126 proc.stdin.write("set output \"graph.tex\" \n")
127
127 proc.stdin.write("plot 'saida.txt' with lines\n")
128
128 proc.stdin.write("quit\n")
129 proc.wait()
130 #proc.terminate()
131 os.system('cp graph.tex ./relatorio/')
132
133 stdscr.addstr (9,0, "\n\nGráfico gerado. \nSaindo\n")
134 stdscr.refresh()
135
136 curses.nocbreak(); stdscr.keypad(0); curses.
    echo()
137 curses.endwin()
138
139 print "Programa concluído com sucesso"

```

APÊNDICE B

ANEXOS

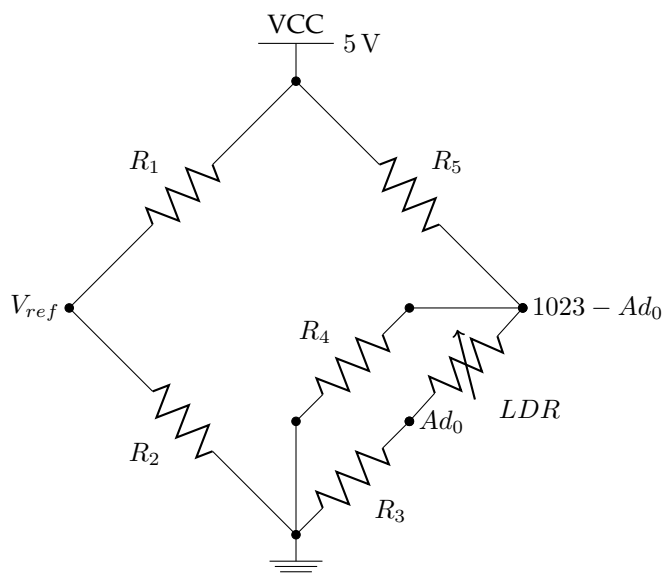
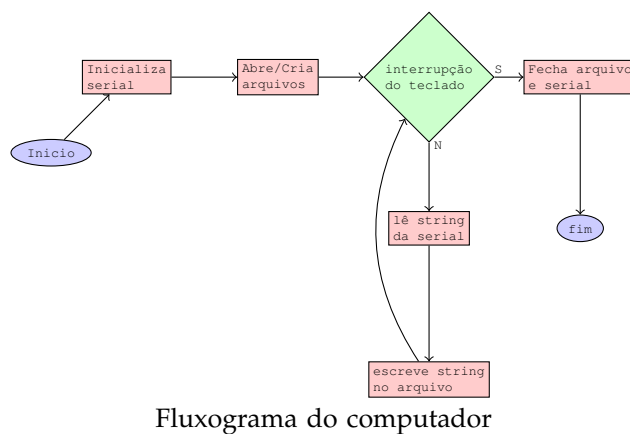
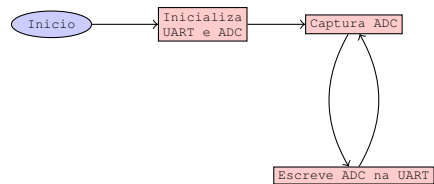


Figura 9: Idealização do circuito do LDR

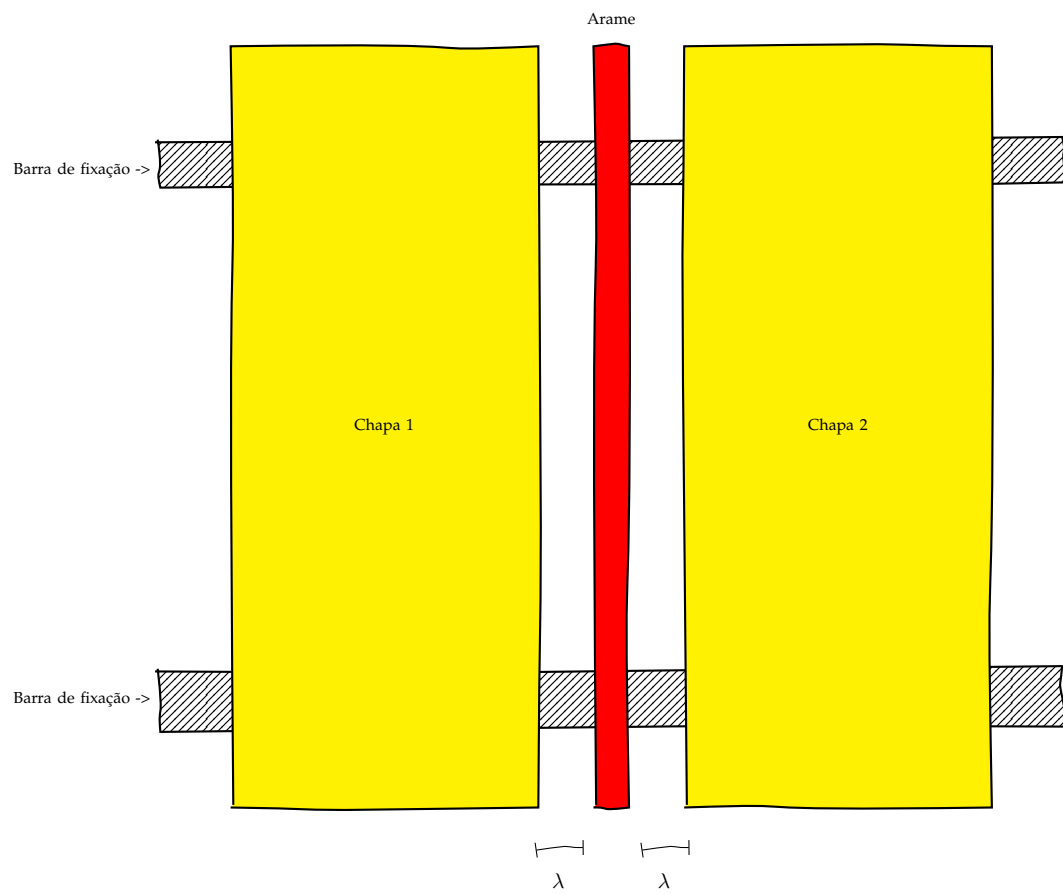


Figura 10: Fenda dupla

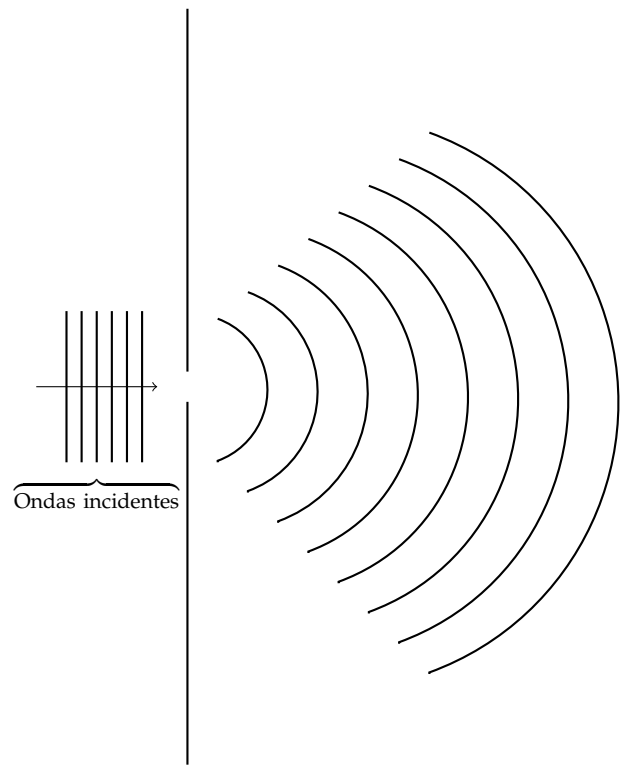


Figura 11: Difração em fenda simples

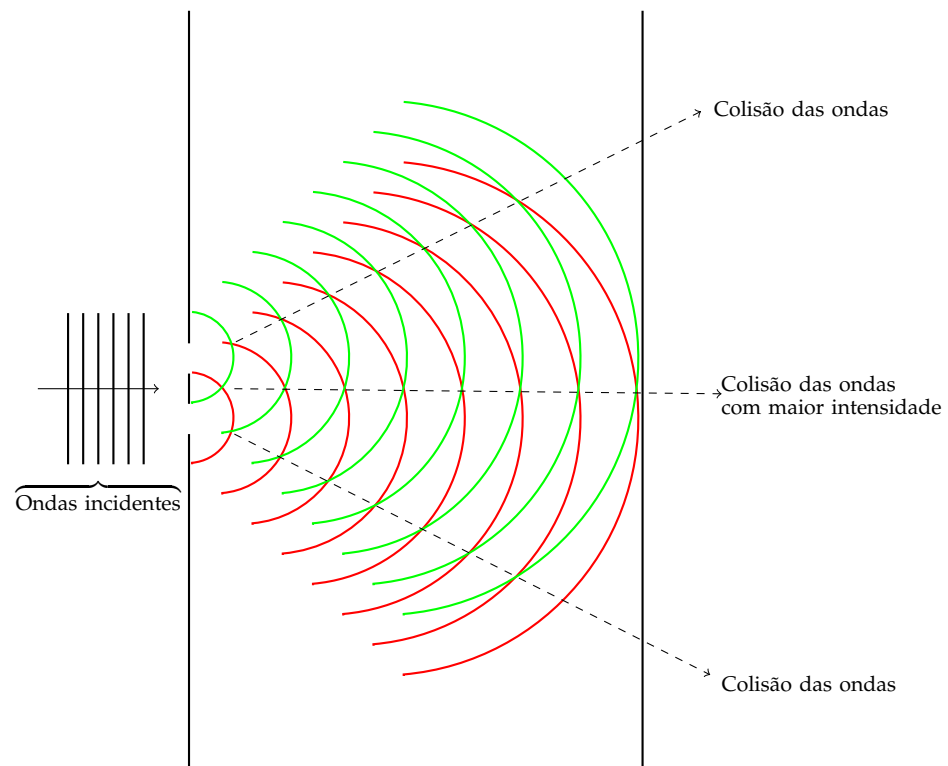


Figura 12: Difração em fenda dupla