Física Moderna
Difração de Multiplas 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 especifico 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.

Index Terms

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

SUMÁRIO

Revisado em 13 de janeiro de 2013.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTINGS

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 angulo 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 interferencia 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 fotoeléctrico 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:

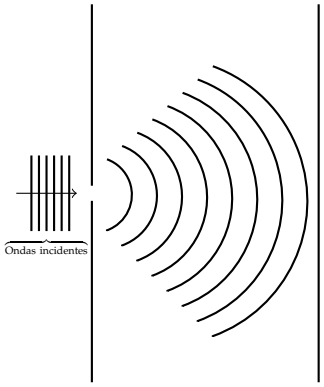


Figura 1: Difração em fenda simples

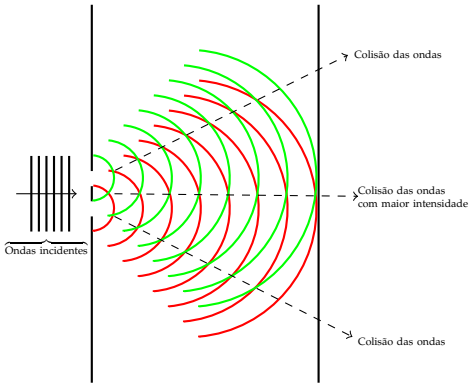


Figura 2: Difração em fenda dupla (com interferência)

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

$$f = \frac{C}{\lambda} \tag{1}$$

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.

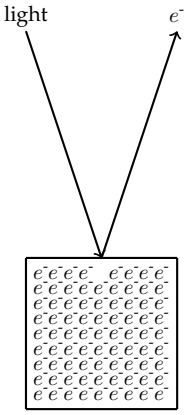


Figura 3: Efeito fotoelétrico

Uma característica importante deste trabalho é que o eletron é emitido a partir do metal com uma energia cinética específica (isto é, uma velocidade específica). Agora qualquer um que esteja familiarizado com o comportamento das ondas sabe que a energia associada com uma onda está relacionada com a sua amplitude ou intensidade. Assim, todos os que pensavam que a luz é apenas uma onda estavam realmente confusos quando a intensidade da luz foi aumentada (luz brilhante) e a energia cinética do elétron emitido não mudou. O que

acontece é que, ao aumentar o brilho da luz mais elétrons são emitidos, mas todos têm a mesma energia cinética.

A energia cinética do elétron emitido tem que depender de alguma coisa. Foi descoberto então que variar a frequência da luz modifica a energia cinética do eletron emitido.

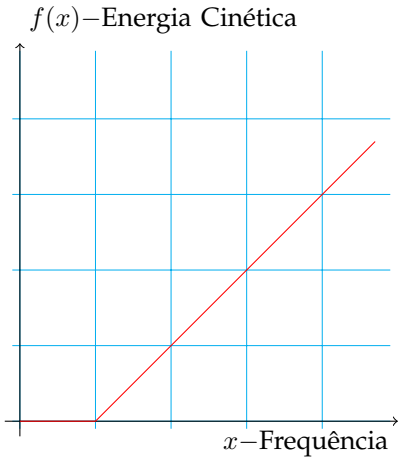


Figura 4: Gráfico Energia cinética vs. Frequência

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 5, é utilizada quando queremos balancear um circuito com uma resistência desconhecida. A ponte é comumente caracterizada pelo desenho na figura 5, onde pode-se obter uma corrente no amperímetro igual a zero quando a seguinte equação é respeitada:

$$R_1 \cdot R_x = R_3 \cdot R_2 \tag{2}$$

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

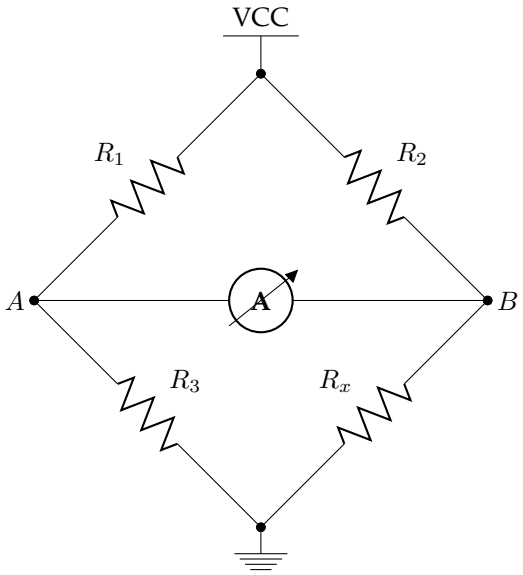


Figura 5: Ponte de Wheatstone

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 6 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.

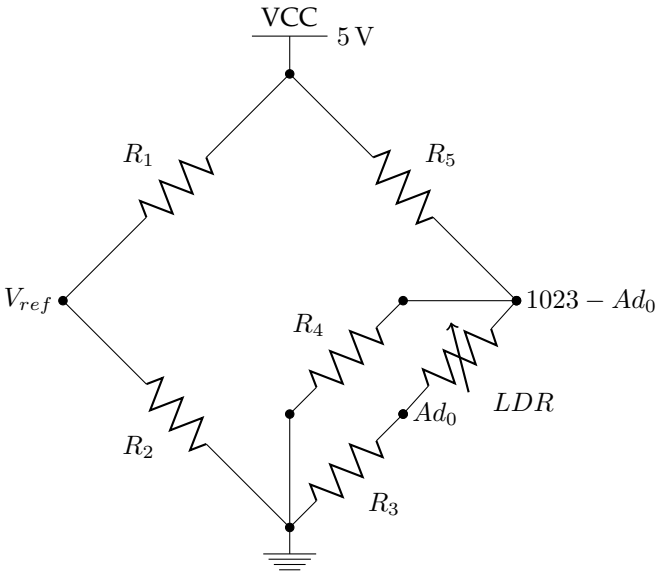


Figura 6: Circuito do LDR

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:

$$\begin{aligned} R \cdot R_x &= R \cdot R \\ R_x &= R \\ R_x &= \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4} \\ R_x &= \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4} = 10^3 \\ 10^3 &= \frac{(R_3 + LDR) \cdot R_4}{(R_3 + LDR) + R_4} \end{aligned}$$

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 7.

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.

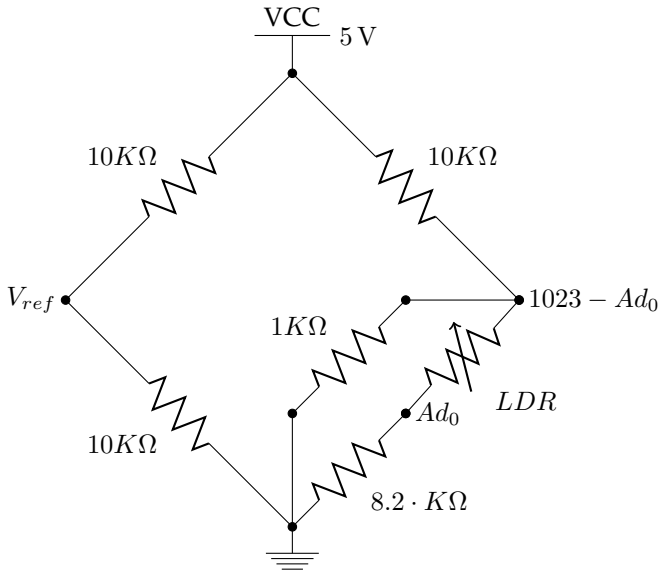


Figura 7: Circuito proposto do LDR

3 CONSTRUINDO AS FENDAS

Para a realização do experimento, era necessário a construção de duas fendas. A primeira tinha como objetivo ser uma simples lacuna entre duas laminas. Porém a segunda fenda regia a necessidade de haver um objeto suficientemente fino entre as duas laminas, com o objetivo de dividi-la em duas fendas.

3.1 Fenda Simples

A obtenção da fenda simples não apresentava grandes dificuldades, tudo o que precisávamos era seguir o modelo descrito na figura 8. Na construção deste ponto do trabalho, utilizamos laminas de barbear como sendo as chapas, e uma lamina extra foi utilizada para garantir uma distancia controlada entre as duas chapas, tendo assim $\lambda = 0.06mm \pm 0.05mm$.

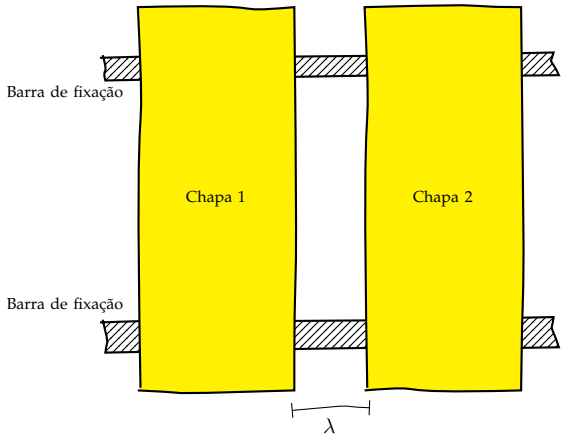


Figura 8: Fenda simples

3.2 Fenda Dupla

Para a construção da fenda dupla, foi necessário a inserção de um fio de cobre entre as duas laminas de aço. Devido ao fato do fio ser mair grosso que a própria fenda simples, foi necessário a utilização de outra lamina para uma nova distancia entre as duas chapas, produzindo assim uma fenda semelhante ao modelo descrito na figura 9, onde $\lambda_1 = \lambda_2 = 0.06mm \pm 0.05mm$.

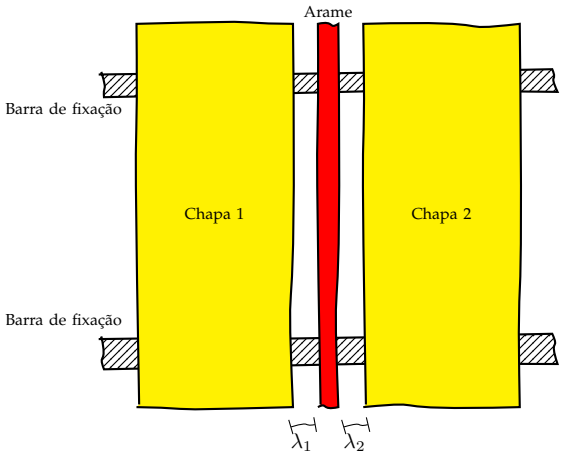


Figura 9: Fenda dupla

4 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 Arduíno e os dados serão transmitidos através de uma comunicação serial RS232.

4.1 Levantando o programa

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

4.1.1 *Shield Comum*

A forma mais simples de utilizar um kit do Arduíno 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 Arduíno, que pode ser observada na figura 10 e escrever o programa tranquilamente, utilizando apenas as funções *loop* e *setup*.

4.1.2 *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 Arduíno. 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 Arduíno pode ser observado em [1].

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

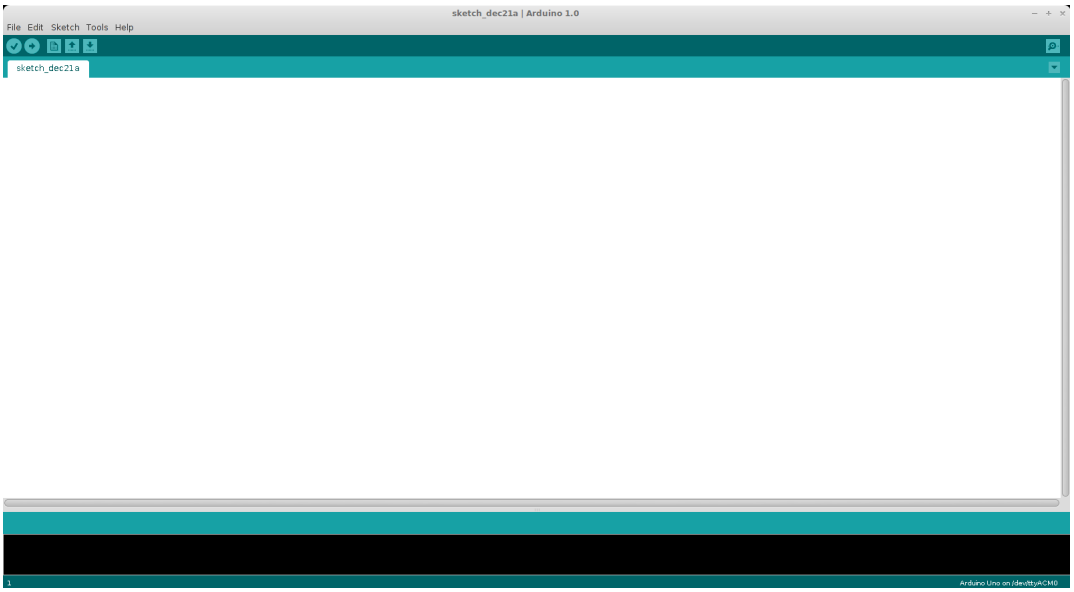


Figura 10: IDE do Arduino

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 mantem toda a estrutura básica do Arduíno, assim como a inclusão do header *Arduino.h*, necessário para a inclusão das rotinas básicas do Arduíno, assim como sua configuração.

4.2 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 11.

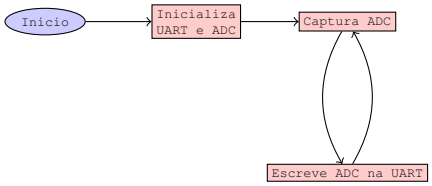


Figura 11: Fluxograma do microcontrolador

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 referencia de tensão externa. A velocidade de 9600 é uma faixa de velocidade bem comum em comunicações seriais. Já a referencia externa é necessária devido ao circuito que foi proposto na sessão 2.2, onde a figura 7 apresenta uma tensão de referencia.

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 integra 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.

5 Coletando os dados no computador - Python

Com o processo realizado na sessão 4, 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 memoria 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.

5.1 Python - o inicio

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 pratica. Para este nosso experimento, utilizaremos uma biblioteca do Python chamada *[2]pyserial*, uma biblioteca voltada para a recepção e envio de dados através de comunicações seriais.

Inicialmente, assim como em quase toda lingaugem 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 binario, 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 4.2. Logo em seguida é feito um simples tratamento de erro para verificar se a porta encontra-se aberta e disponível para leitura.

5.2 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 invalido.

5.3 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,
```

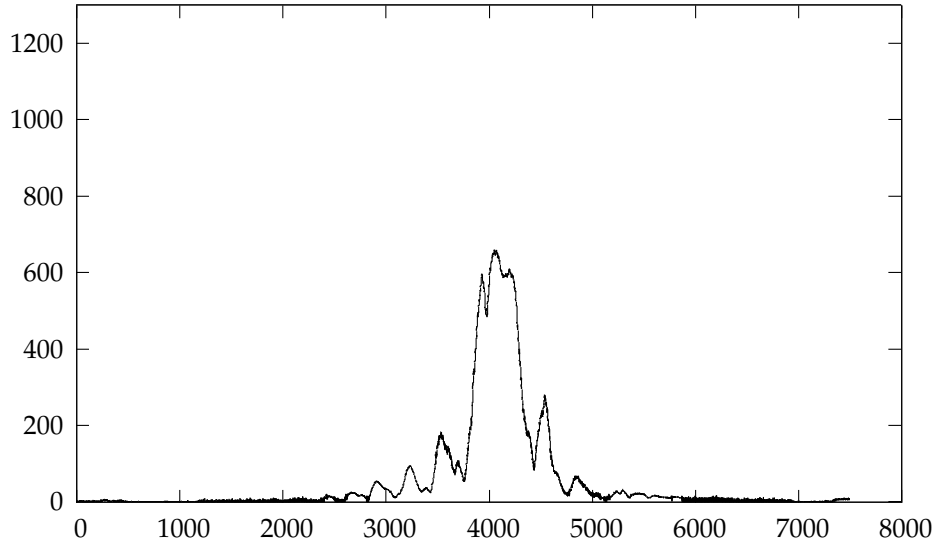
```
                                stdin=subprocess.
                                    PIPE,
                                )

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`**.

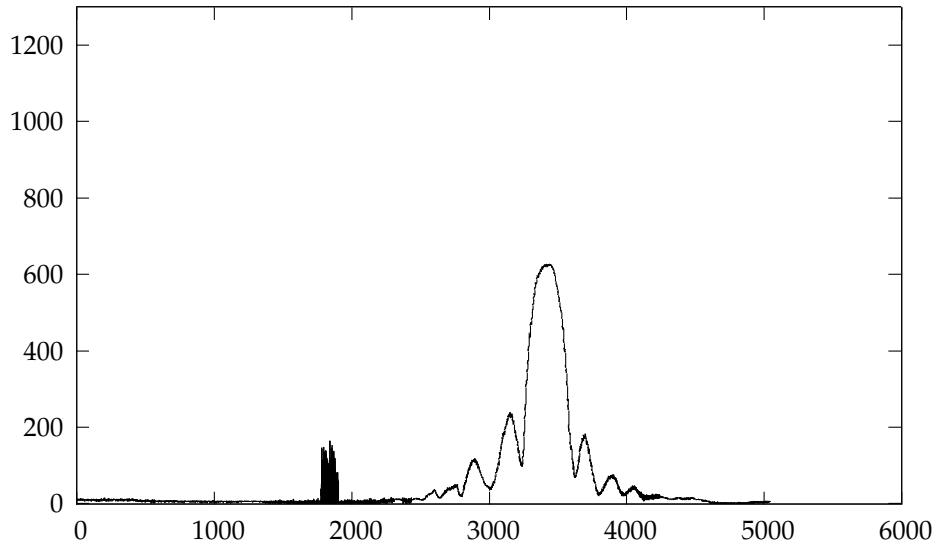
6 AMOSTRAS - LASER VERMELHO

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

1) Verificação da colheira do sinal na fenda dupla.

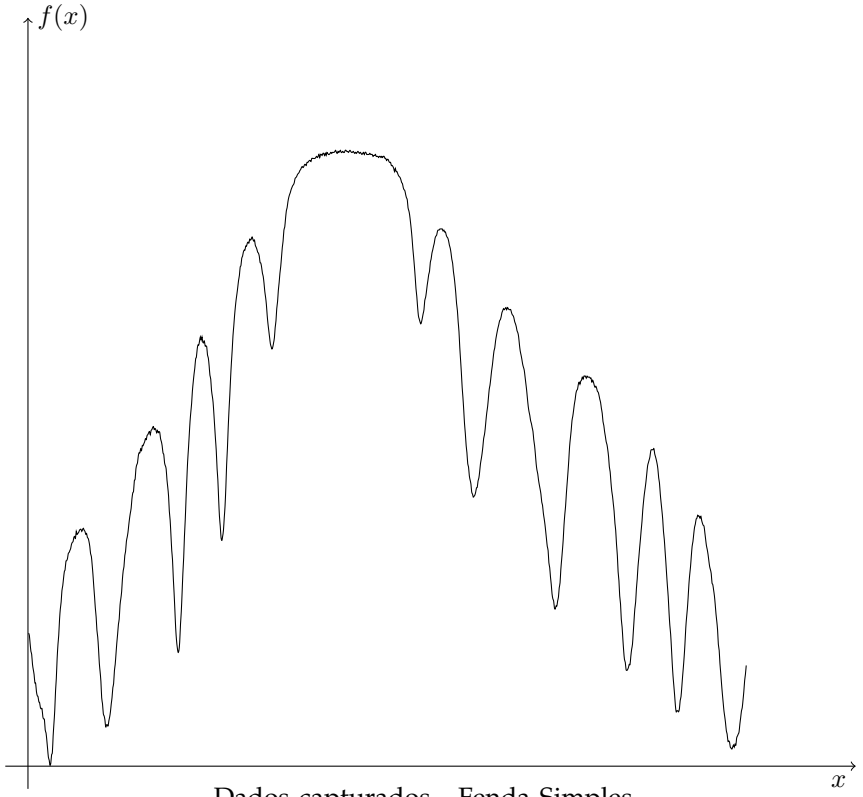


2) Colheita do sinal na fenda dupla - houve uma leve falha no inicio devido à localização do sensor em frente ao sinal da difração



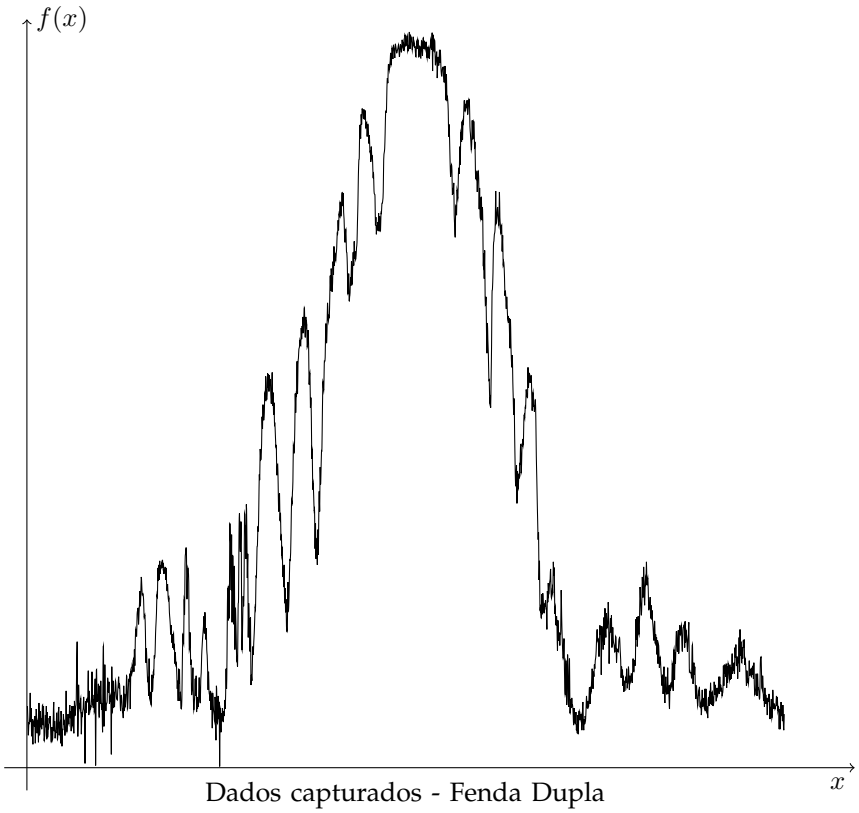
7 AMOSTRAS - LASER VERDE

1) Fenda simples, laser verde



Dados capturados - Fenda Simples

2) Fenda dupla, laser verde



Dados capturados - Fenda Dupla

REFERÊNCIAS

[1] RIGONI, F. *Arduino development with Eclipse*. [S.l.]. Disponível em: <http://horrorcoding.altervista.org/arduino-development-with-eclipse-a-step-by-step-tutorial-to-the-basic-setup/>. Acesso em: 13 de janeiro de 2013.

[2] LIECHTI, C. *pySerial API*. [S.l.]. Disponível em: <http://pyserial.sourceforge.net/pyserial_api.html>. Acesso em: 13 de janeiro de 2013.

APÊNDICE A
CÓDIGOS FONTES

Listing 8: Código principal do Microcontrolador

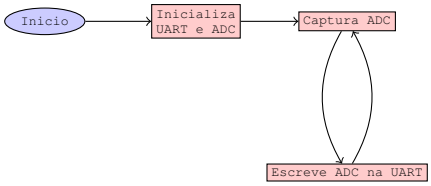
```
/**
 * Test model for Arduino - UNO
 */

#include <Arduino.h>
#include <stdio.h>
#include <stdlib.h>

/**
 *
 * The setup() method runs once, when the
 * sketch starts
 * This is just starting the UART in 9600
 * and setting the reference for the analog
 * as External
 *
 */
void setup()
{
    // wait for 10ms
    delay(10);
    //! initialize the default debug output:
    Serial.begin(9600);
    //! Setting the analog reference as
        external
    analogReference(EXTERNAL);
    // wait for 100ms to stabilize
    delay(100);
}

/***
 *
 *  The loop() method runs over and over again
     ,
 *  as long as the Arduino has power
 *
 */
void loop()
{
static int i=0;
char buf[20];

    //! raw values
    sprintf(buf,"%d %d\n",i++,analogRead(0) );
    Serial.print(buf);

    delay(1);       // wait for 1ms
}

/**
 * This is the standard main function. On
     Arduino, it's NEVER changes.
 * @return the standard returns that the main
     should return.
 */
int main(void)
{

    //! Initialize the pins I/O and standard
        configures.
    init();
    //! Call the user's setup function.
    setup();

    //! The eternal loop
    while(true) {
    loop();
    }

    return 0;
}
```

Listing 9: Código principal de exibição dos dados

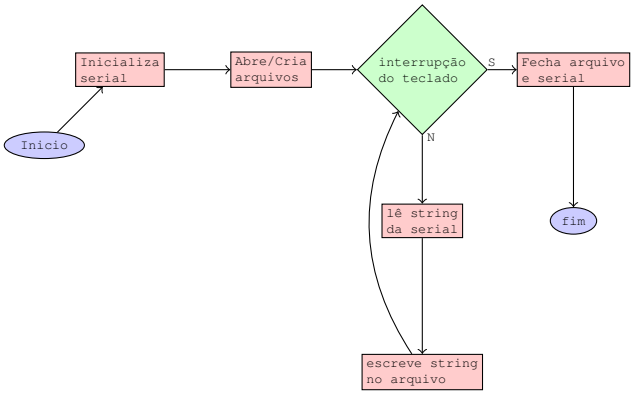
```
#!/usr/bin/python
# -*- coding:utf-8 -*-

import serial        # metodos para a serial
import os            # metodos para SO
import subprocess    # para subprocessos ( gnuplot)
import sys           # para ter acesso aos parametros
import curses        # para a biblioteca curses

"""
import signal        # pegando sinais como em C

def signal_handler(signal, frame):
    print 'You pressed Ctrl+C!'
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)
"""

try:
    ser = serial.Serial(sys.argv[1],9600)
    ser.open()
except IndexError:
    print "Passe uma porta existente com o parametro!"
    print "\teg.: python " + sys.argv[0] + " /dev/ttyUSB0"
    print "\teg.: ./" + sys.argv[0] + " /dev/ttyUSB0"
    sys.exit(0)
except IOError as e:
    print("({})".format(e))
    sys.exit(0)
    raise

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 não acessível"
    sys.exit(0)

vsel=0
l_vsel=1
teste=True
#limpa tela e buffer
#os.system('clear')
ser.flushInput()

stdscr = curses.initscr()
curses.noecho()

#print "Aperte Ctrl+C para terminar a leitura\n\n"
stdscr.addstr (0,0,"Aperte Ctrl+C para terminar a leitura\n\n",curses.A_BOLD)
stdscr.addstr (1,0,"!")
while teste:
    try:
        msg = ser.readline()
        #print  msg,
        stdscr.addstr (3,0,msg,curses.A_REVERSE)
        stdscr.refresh()
    except KeyboardInterrupt:
        teste=False
        break

    finally:
        try:
            f.write(msg)
            if (l_vsel == eval(msg.split(" ")[0]) ):
                l_vsel=l_vsel+1
            else:
                raise
            vsel = eval(msg.split(" ")[0])/10.0
            fx.write("-- (%.2f,%.2f) " % (vsel, eval(msg.split(" ")[1])/100.0  ) )
            if (vsel > 5):
                stdscr.addstr (1,0,"  ")
            else:
                stdscr.addstr (1,0,"!")
            stdscr.refresh()
        except:
#           curses.nocbreak(); stdscr.keypad(0); curses.echo()
#           curses.endwin()
            ser.close()
            ser.open()
            f=open('./saida.txt','w')
            fx=open('./trabalhada.txt','w')
            try:
                if len(msg) > 1:
#                   print "\rFalha na sincronização!\n\rFoi capturado contudo inválido \t" + msg
                    stdscr.addstr (1,0,"!")
#                   stdscr.refresh()
            except:
                print "\rNão foi capturado nada!"
                curses.nocbreak(); stdscr.keypad(0); curses.echo()
                curses.endwin()
                ser.close()
                f.close()
                fx.close()
                sys.exit(0)


stdscr.addstr (5,0, "\r   \nFim da leitura!")
ser.close()

stdscr.addstr (6,0, "Fechando arquivo \"saida.txt\"")
f.close()
fx.close()
stdscr.addstr (7,0, "Serial fechada. Abrindo o GNUPLOT")
stdscr.refresh()

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

#Margem segura para visualizar os dados
proc.stdin.write("set yrange [-1:1300]\n")
proc.stdin.write("plot 'saida.txt' with lines\
```

```
    n")
proc.stdin.write("pause mouse\n")
stdscr.addstr (8,0, " Clique na figura para fecha-la ",curses.A_BOLD)
stdscr.addstr (25,25, " ")
stdscr.refresh()

proc.stdin.write("set terminal latex\n")
proc.stdin.write("unset key\n")
proc.stdin.write("set output \"graph.tex\" \n")
proc.stdin.write("plot 'saida.txt' with lines\n")
proc.stdin.write("quit\n")
proc.wait()
#proc.terminate()
os.system('cp graph.tex ./relatorio/')

stdscr.addstr (9,0, "\n\nGráfico gerado. \nSaindo\n")
stdscr.refresh()

curses.nocbreak(); stdscr.keypad(0); curses.echo()
curses.endwin()

print "Programa concluído com sucesso"
```

APÊNDICE B
ANEXOS



Fluxograma do microcontrolador



Fluxograma do computador

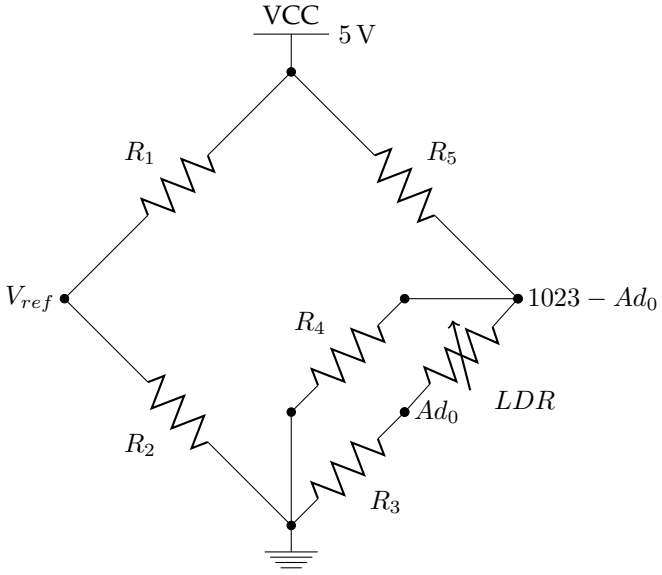


Figura 12: Idealização do circuito do LDR

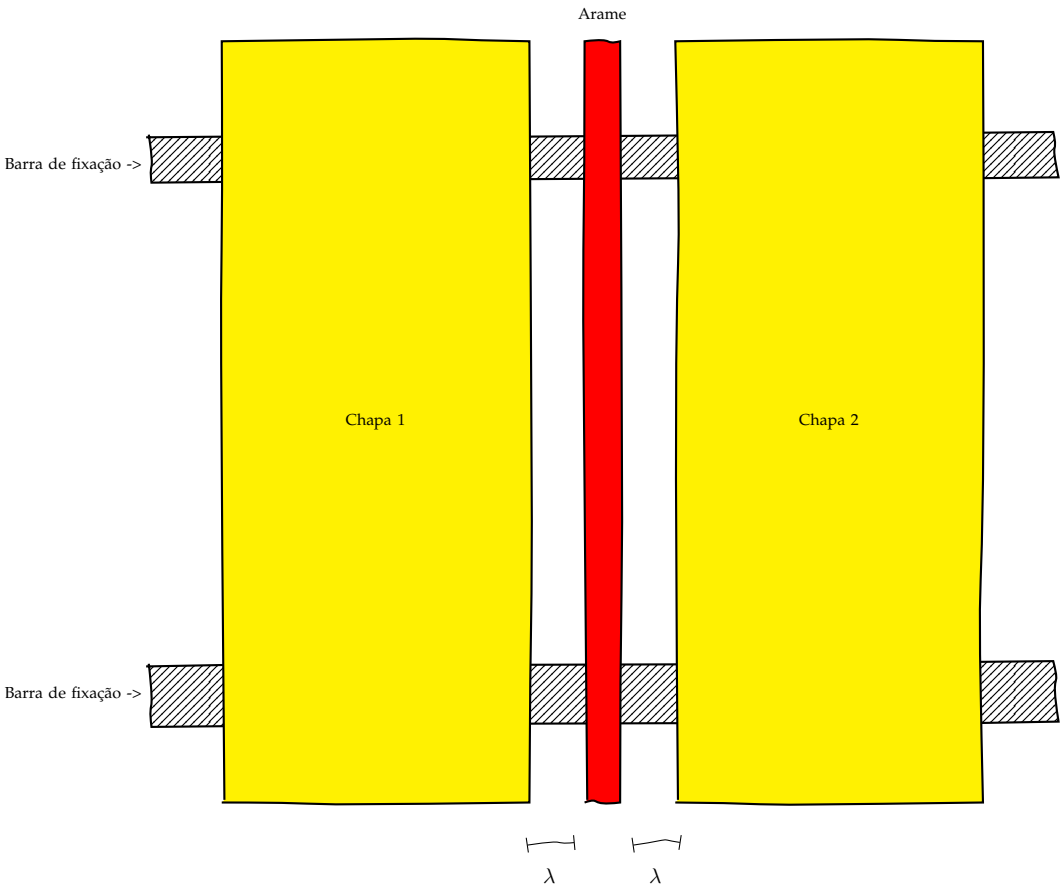


Figura 13: Fenda dupla

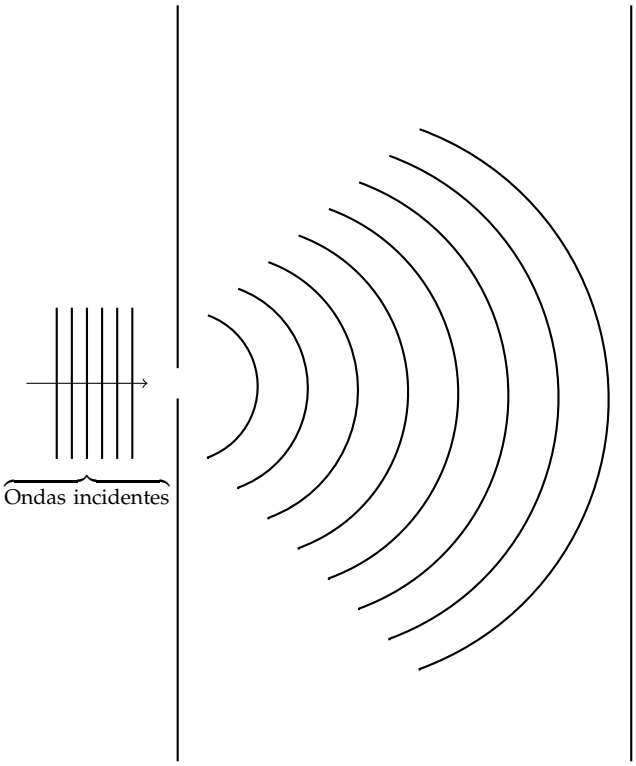


Figura 14: Difração em fenda simples

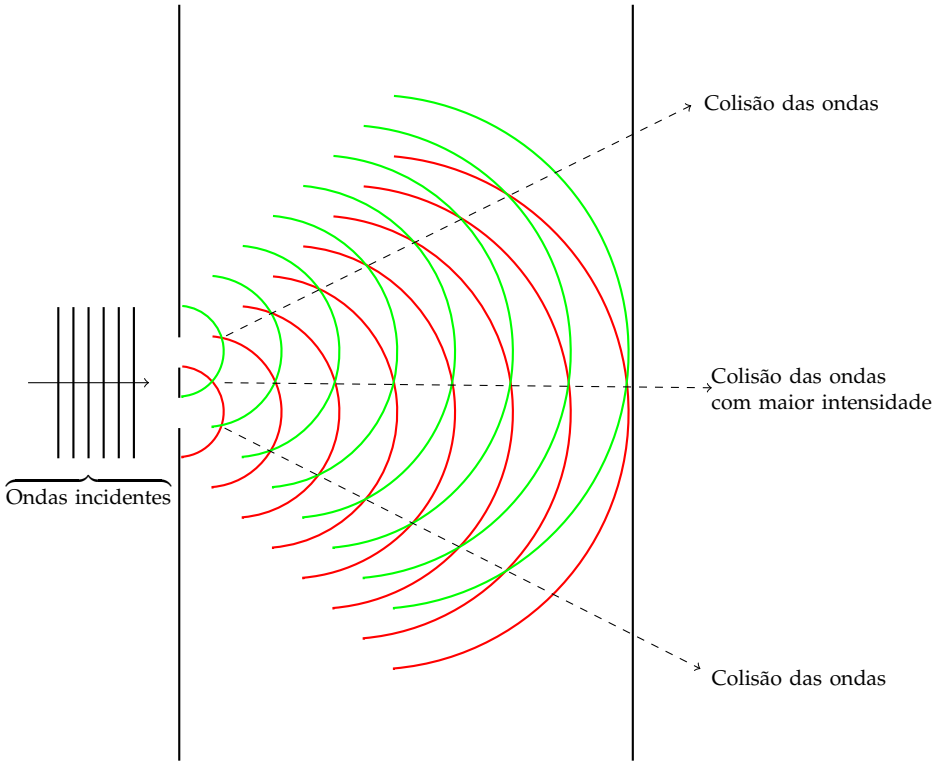


Figura 15: Difração em fenda dupla