

# 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.

### Index Terms

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



## SUMÁRIO

<b>1</b>	<b>Criação do Circuito</b>	<b>1</b>
1.1	Ponte de Wheatstone . . . . .	1
1.2	Abstração da ponte para o circuito final . . . . .	1
<b>2</b>	<b>Colhendo dados reais - Arduino</b>	<b>2</b>
2.1	Levantando o programa . . . . .	2
2.1.1	Shield Comum . . . . .	2
2.1.2	Utilizando o Eclipse para o Arduino . . . . .	2
2.2	Criação do Código . . . . .	3
<b>3</b>	<b>Coletando os dados no computador - Python</b>	<b>3</b>
3.1	Python - o início . . . . .	3
3.2	Lendo os dados . . . . .	4
3.3	Plotando os dados . . . . .	5
<b>4</b>	<b>Amostras - laser vermelho</b>	<b>5</b>
<b>5</b>	<b>Amostras - laser verde</b>	<b>6</b>
	<b>Referências</b>	<b>7</b>
	<b>Apêndice A: Códigos Fontes</b>	<b>7</b>
	<b>Apêndice B: Anexos</b>	<b>10</b>

**LISTA DE FIGURAS**

1	Ponte de Wheatstone . . . . .	1
2	Circuito do LDR . . . . .	1
3	Circuito proposto do LDR . . . . .	2
4	IDE do Arduino . . . . .	2
5	Fluxograma do microcontrolador . . . . .	3
6	Idealização do circuito do LDR . . . . .	10
7	Fenda dupla . . . . .	11
8	Difração em fenda simples . . . . .	11
9	Difração em fenda dupla . . . . .	12

**LISTA DE TABELAS****LISTINGS**

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

## 1 CRIAÇÃO DO CIRCUITO

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

### 1.1 Ponte de Wheatstone

A ponte de *Wheatstone*, representada na figura 1, é utilizada quando queremos balancear um circuito com uma resistência desconhecida. A ponte é comumente caracterizada pelo desenho na figura 1, onde pode-se obter uma corrente no amperímetro igual a zero quando a seguinte equação é respeitada:

$$R_1 \cdot R_x = R_3 \cdot R_2 \quad (1)$$

Quando a equação 1 é respeitada, as tensões entre os pontos **A** e **B** é a mesma, de forma que não existe corrente nestes pontos

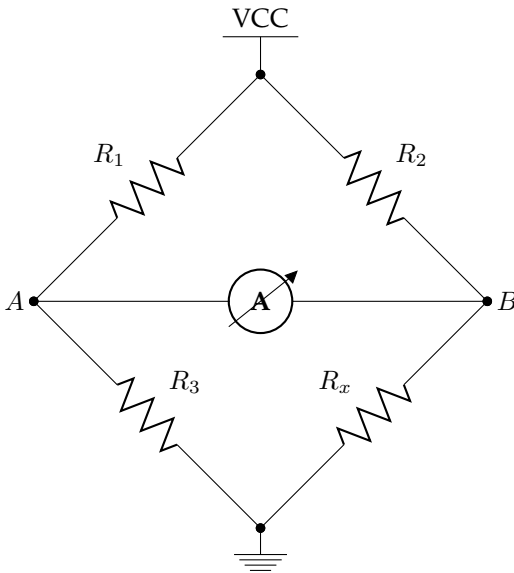


Figura 1: Ponte de Wheatstone

### 1.2 Abstração da ponte para o circuito final

Com base no conhecimento obtido na sessão 1.1, podemos inferir que no circuito da figura 2 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  $100\Omega$  quando exposto a luz e  $10^6\Omega$  quando em completa escuridão.

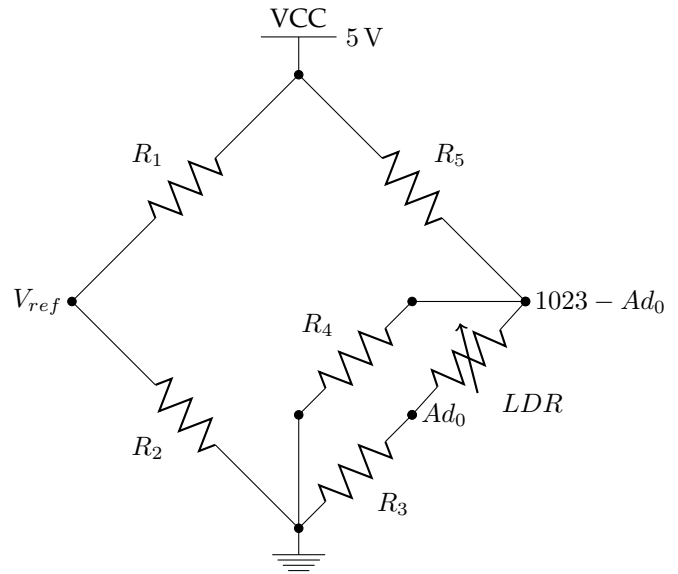


Figura 2: Circuito do LDR

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:

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

$$R_x = R$$

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

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

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

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

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 3.

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.

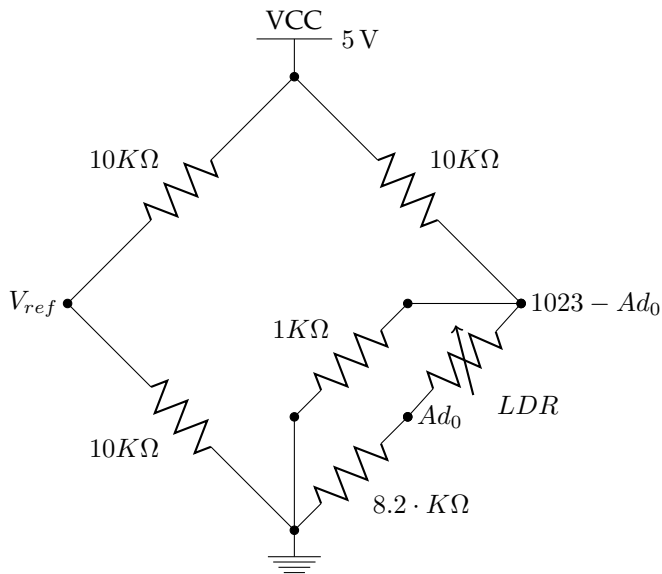


Figura 3: Circuito proposto do LDR

## 2 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.

### 2.1 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:

#### 2.1.1 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 4 e escrever o programa tranquilamente, utilizando apenas as funções *loop* e *setup*.

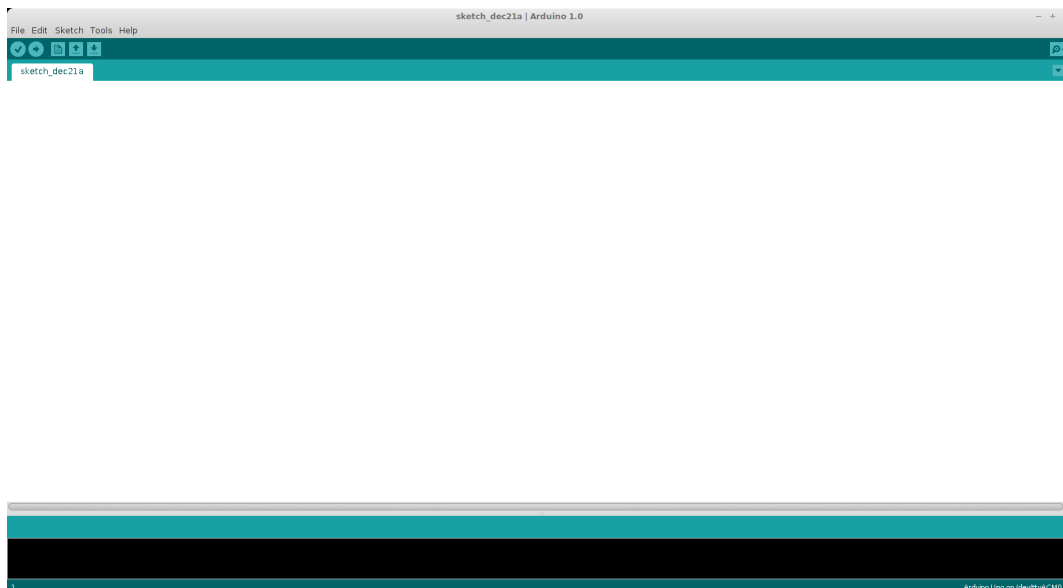


Figura 4: IDE do Arduino

#### 2.1.2 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 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 [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 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();

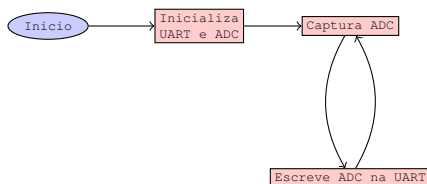
    while(true)
        loop();

    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.

## 2.2 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 5.



**Figura 5:** 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 1.2, onde a figura 3 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.

## 3 COLETANDO OS DADOS NO COMPUTADOR - PYTHON

Com o processo realizado na sessão 2, 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.

### 3.1 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 [2]*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 Arduino foi feito, na sessão 2.2. Logo em seguida é feito um simples tratamento de erro para verificar se a porta encontra-se aberta e disponível para leitura.

### 3.2 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 feita uma verificação do conteúdo capturado. Assim garante que o conteúdo que será plotado está na formatação desejada e não possui nenhum carácter inválido.

### 3.3 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,
                        )
```

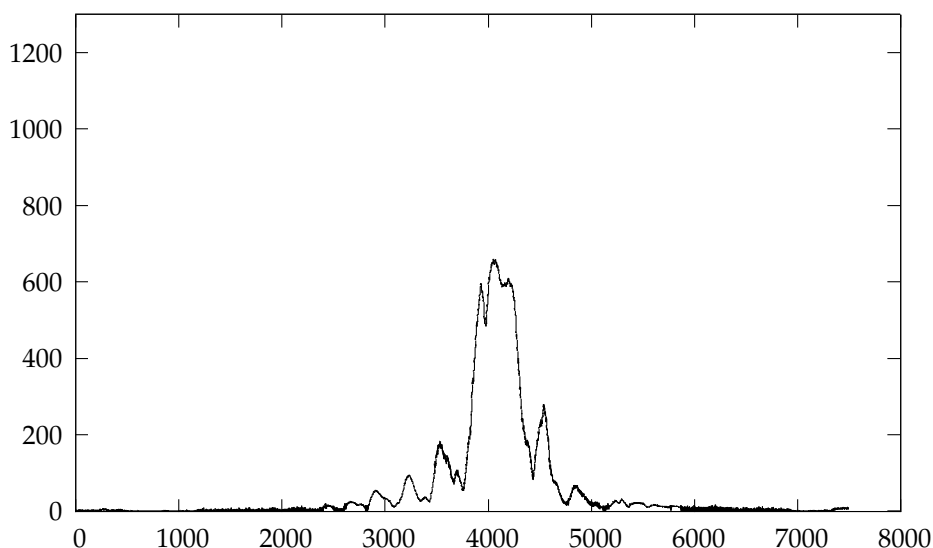
```
proc.stdin.write("plot 'saida.txt' with
                lines\n")
proc.stdin.write("pause mouse\n")
proc.stdin.write("quit\n")
```

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 íntegra 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`.

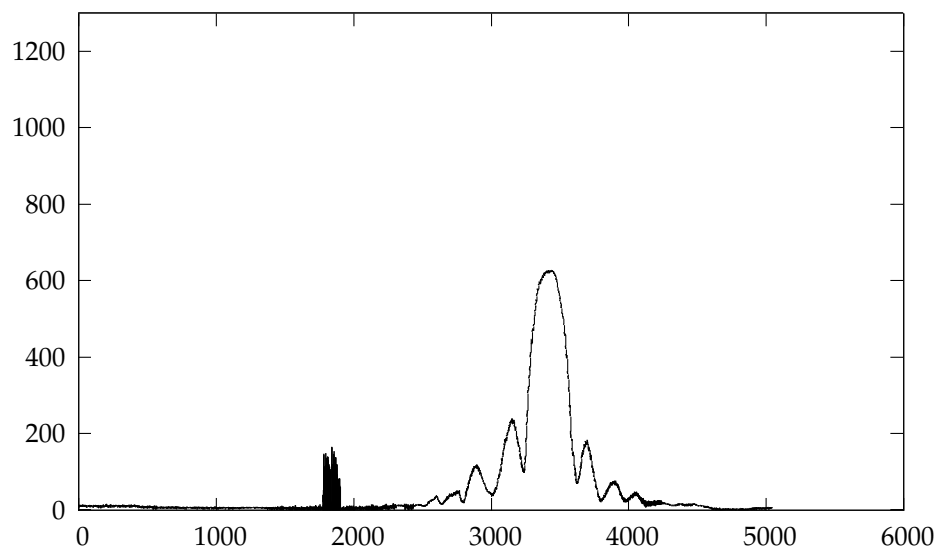
## 4 AMOSTRAS - LASER VERMELHO

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

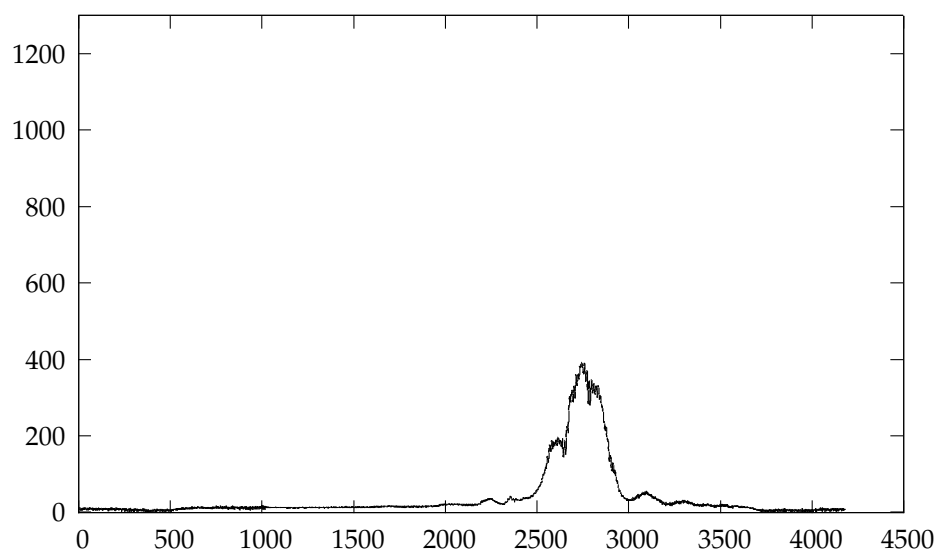
### 1) Verificação da colheita do sinal na fenda dupla.



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



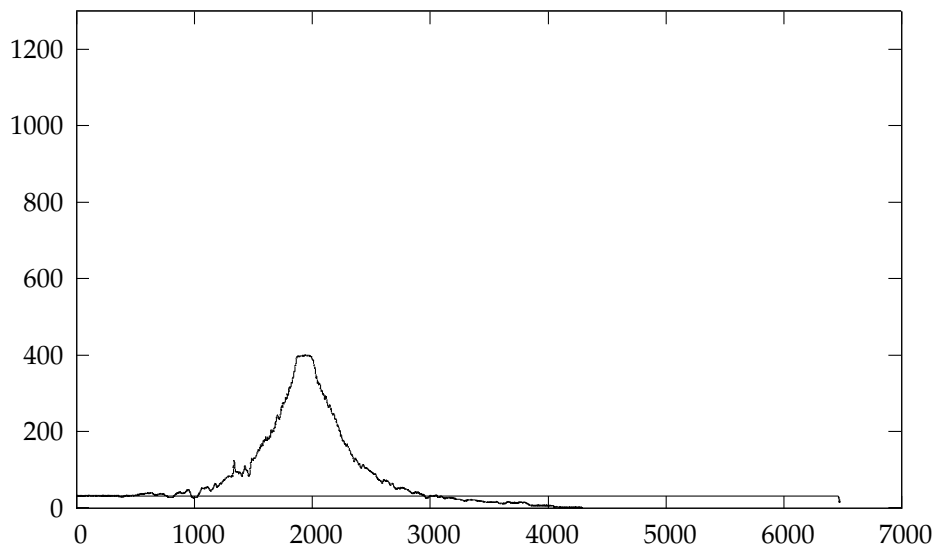
3) Colheira do sinal na fresta simples.



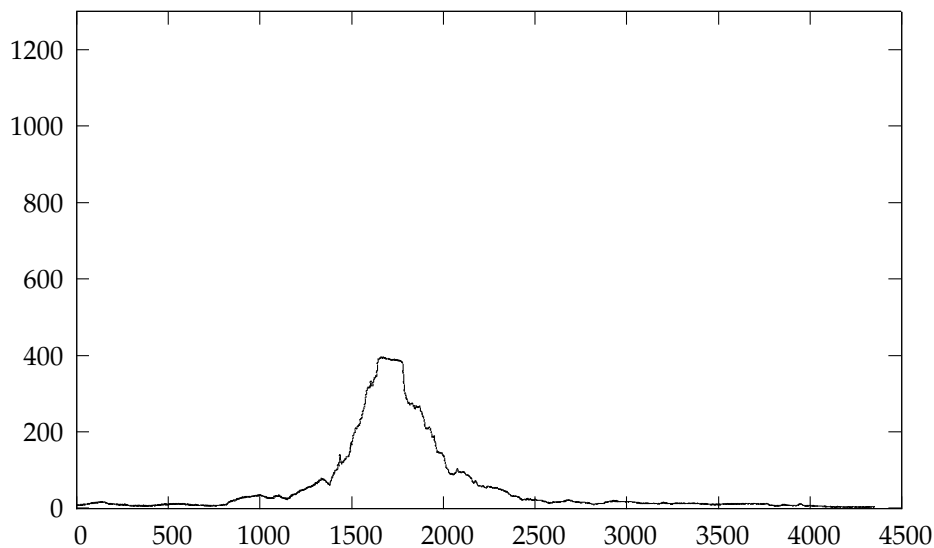
## 5 AMOSTRAS - LASER VERDE

1) Fenda simples, laser verde





2) Fenda dupla, laser verde



## 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: 4 de janeiro de 2013.
- [2] LIECHTI, C. *pySerial API*. [S.l.]. Disponível em: <[http://pyserial.sourceforge.net/pyserial\\_api.html](http://pyserial.sourceforge.net/pyserial_api.html)>. Acesso em: 4 de janeiro de 2013.

## APÊNDICE A CÓDIGOS FONTES

Listing 8: Código principal do Microcontrolador

```

1 /**
2  * Test model for Arduino - UNO
3  */
4
5 #include <Arduino.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 /**
10  *
11  * The setup() method runs once, when the
12  * sketch starts
13  * This is just starting the UART in 9600
14  * and setting the reference for the analog
15  * as External
16  *
17  */
18 void setup()
19 {
20     // wait for 10ms
21     delay(10);

```

```

22  #!/initialize the default debug output:
23  Serial.begin(9600);
24  #!/Setting the analog reference as
    external
25  analogReference(EXTERNAL);
26  // wait for 100ms to stabilize
27  delay(100);
28 }
29
30 /**
31 *
32 * The loop() method runs over and over again
    '
33 * as long as the Arduino has power
34 *
35 */
36 void loop()
37 {
38 static int i=0;
39 char buf[20];
40
41 #!/raw values
42 sprintf(buf,"%d %d\n",i++,analogRead(0) );
43 Serial.print(buf);
44
45 delay(1);          // wait for 1ms
46 }
47
48 /**
49 * This is the standard main function. On
    Arduino, it's NEVER changes.
50 * @return the standard returns that the main
    should return.
51 */
52 int main(void)
53 {
54
55 #!/Initialize the pins I/O and standard
    configures.
56 init();
57 #!/Call the user's setup function.
58 setup();
59
60 #!/The eternal loop
61 while(true) {
62 loop();
63 }
64
65 return 0;
66 }

```

Listing 9: Código principal de exibição dos dados

```

1 #!/usr/bin/python
2 # -*- coding:utf-8 -*-
3
4 import serial          # metodos para a serial
5 import os              # metodos para SO
6 import subprocess      # para subprocessos (
    gnuplot)
7 import sys              # para ter acesso aos
    parametros
8 import curses           # para a biblioteca curses
9
10 """
11 import signal           # pegando sinais como em C
12
13 def signal_handler(signal, frame):
14     print 'You pressed Ctrl+C!'
15     sys.exit(0)
16
17 signal.signal(signal.SIGINT, signal_handler)
18 """
19
20 try:
21     ser = serial.Serial(sys.argv[1],9600)
22     ser.open()
23 except 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)
28 except IOError as e:
29     print "({})".format(e)
30     sys.exit(0)
31     raise
32
33 if 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
39 else:
40     print "Porta não acessível"
41     sys.exit(0)
42
43 #limpa tela e buffer
44 #os.system('clear')
45 ser.flushInput()
46
47 stdscr = curses.initscr()
48 curses.noecho()
49
50 #print "Aperte Ctrl+C para terminar a leitura\
    n\n"
51 stdscr.addstr (0,0,"Aperte Ctrl+C para
        terminar a leitura\n\n",curses.A_BOLD)
52 while True:
53     try:
54         msg = ser.readline()
55         #print msg,
56         stdscr.addstr (3,0,msg,curses.
            A_REVERSE)
57         stdscr.refresh()
58     except KeyboardInterrupt:
59         break
60

```

```

61     finally:
62         try:
63             f.write(msg)
64             vsel = eval(msg.split(" ")[0])
65                 /10.0
66             fx.write("-- (%.2f,%.2f) " % (vsel
67                 , eval(msg.split(" ")[1])
68                 /100.0 ) )
69         except:
70             curses.nocbreak(); stdscr.keypad
71                 (0); curses.echo()
72             curses.endwin()
73             try:
74                 if len(msg) > 1:
75                     print "\rFalha na
76                         sincronização!\nFoi
77                         capturado contudo invá
78                         lido \n" + msg
79             except:
80                 print "\rNão foi capturado
81                     nada!"
82             ser.close()
83             f.close()
84             fx.close()
85             sys.exit(0)
86
87 stdscr.addstr (5,0, "\r \nFim da leitura!")
88 ser.close()
89
90 stdscr.addstr (6,0, "Fechando arquivo \"saida.
91     txt\"")
92 f.close()
93 fx.close()
94 stdscr.addstr (7,0, "Serial fechada. Abrindo o
95     GNUPLOT")
96 stdscr.refresh()
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117

```

## APÊNDICE B

### ANEXOS

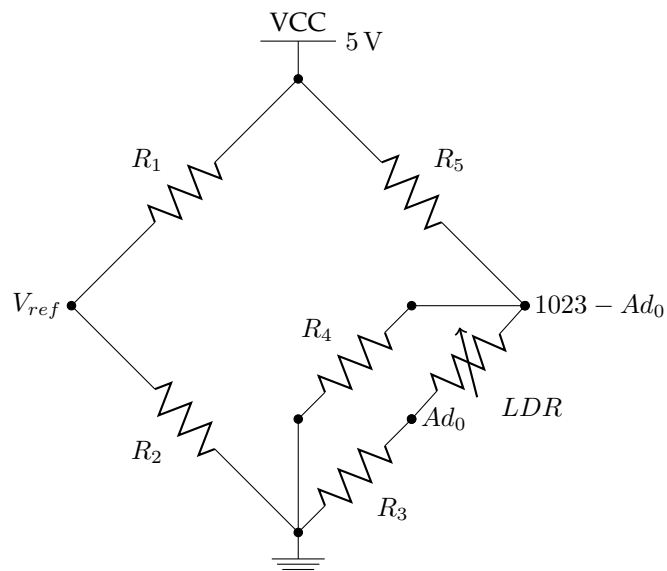
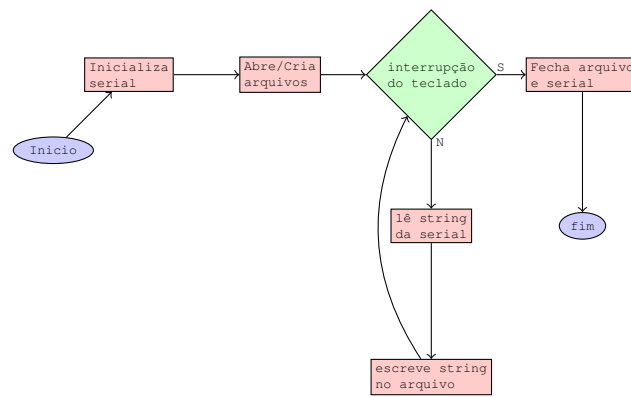
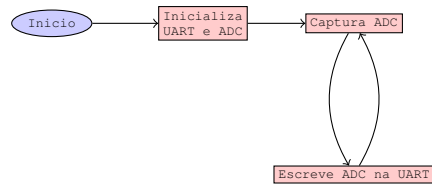
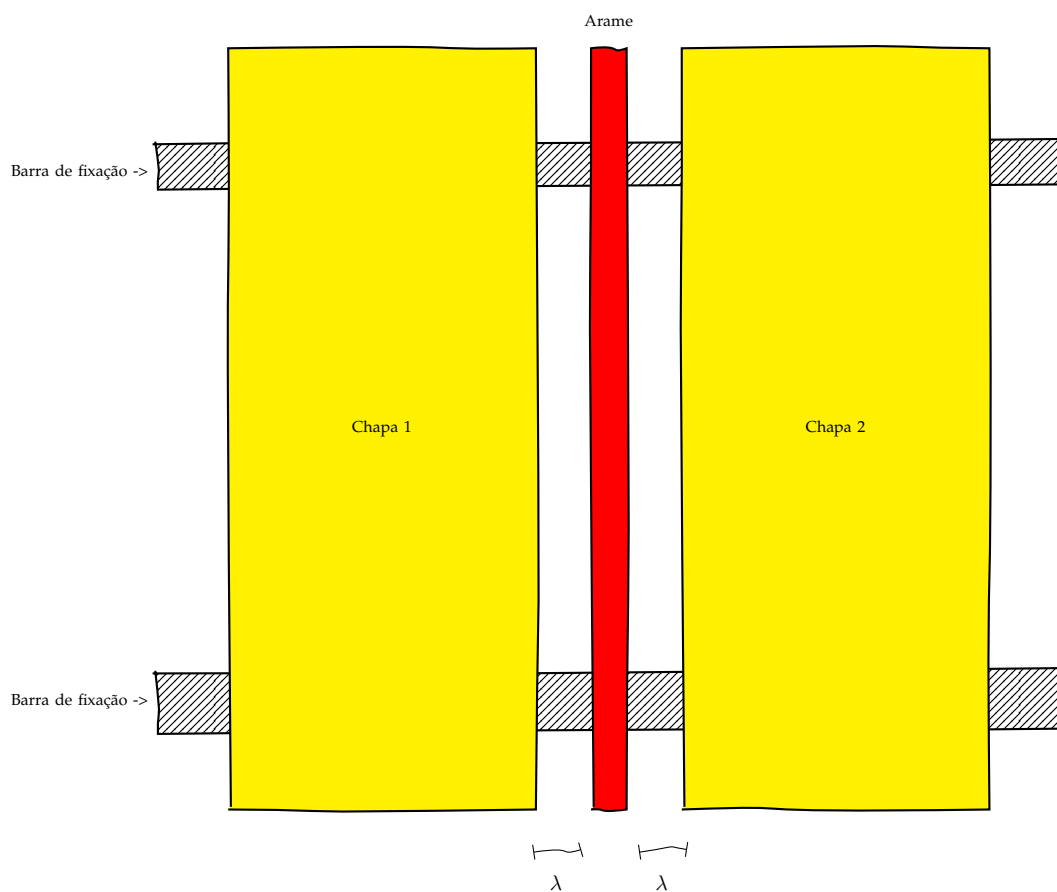
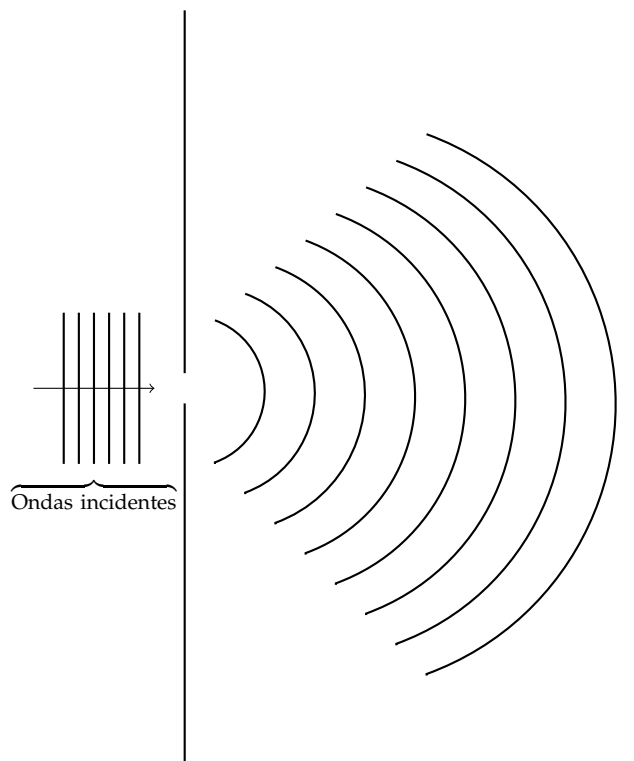


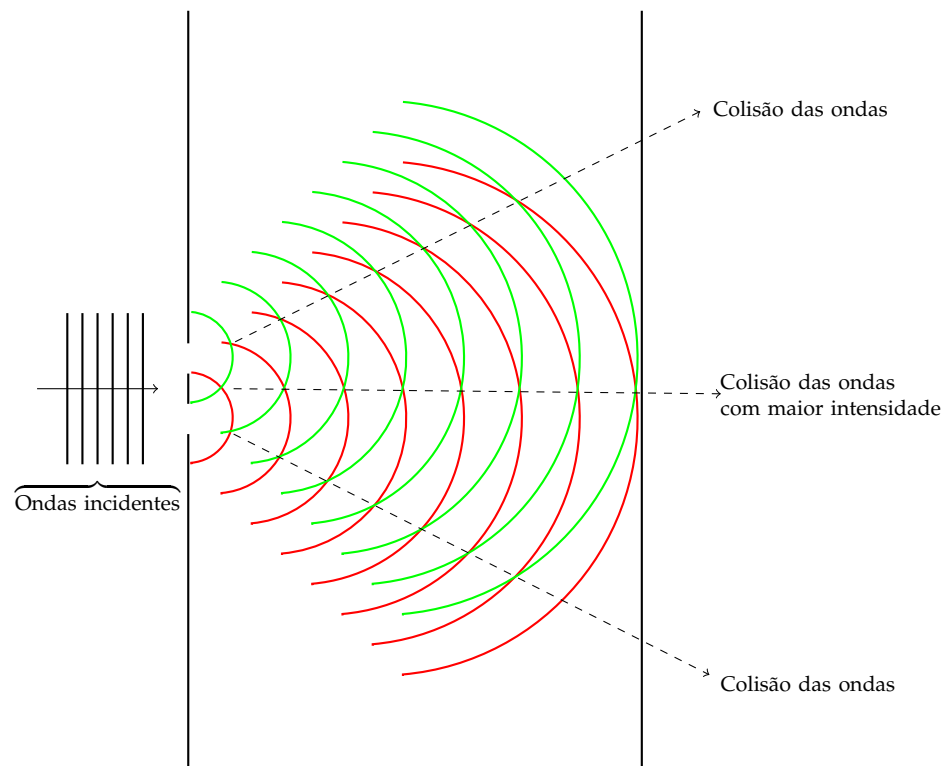
Figura 6: Idealização do circuito do LDR



**Figura 7:** Fenda dupla



**Figura 8:** Difração em fenda simples



**Figura 9:** Difração em fenda dupla