Vibrações e Ondas

LEEC – VIBON 2019/2020

Trabalho 1 – Relatório

"Estudo do Movimento Oscilatório com SONAR"

Identificação

Turma: 3DA **Grupo**: 3 **Data**: Dezembro 2019

Número	**Nome**
1160736	Diogo Pinto
1160628	Carlos Silva
1161089	Pedro Apura

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

Propósito e objectivos do trabalho:

- Estudo experimental do movimento oscilatório de um sistema físico com amortecimento (sistema massa-mola), usando um sensor de distância baseado em SONAR (SOund Navegation And Ranging);
- Implementaçãoe hardware e desenvolvimento de software para automação de um sistema de aquisição e processamento de dados, controlado por computador, baseado no SONAR HY-SRF05 e microcontrolador ESP-01.

Este trabalho está dividido em 3 partes, a realizar sequencialmente:

1. **Calibração de um SONAR**, e determinação experimental da velocidade do som,
2. **Verificação da Lei de Hooke**, e determinação da constante elástica de uma mola,
3. **Oscilador Amortecido**, e determinação do coeficiente de amortecimento do ar.

2 Material e Equipamento

- Suporte com haste
- 1 Placa (caixa CD)
- 1 Fita Métrica
- 1 Mola Elástica
- 1 Suporte com gancho
- 10 Massas (10 $\rightarrow$ 100g)
- 1 ESP-01
- 1 Sonar HY-SRF05
- 1 Sensor de Temperatura e Humidade DHT11
- Python IDE

3 Procedimento Experimental

A Figura 1 ilustra, de forma esquemática, as montagens experimentais necessárias para a realização das 3 partes do trabalho: calibração do SONAR, Lei de Hooke e Oscilador Amortecido.

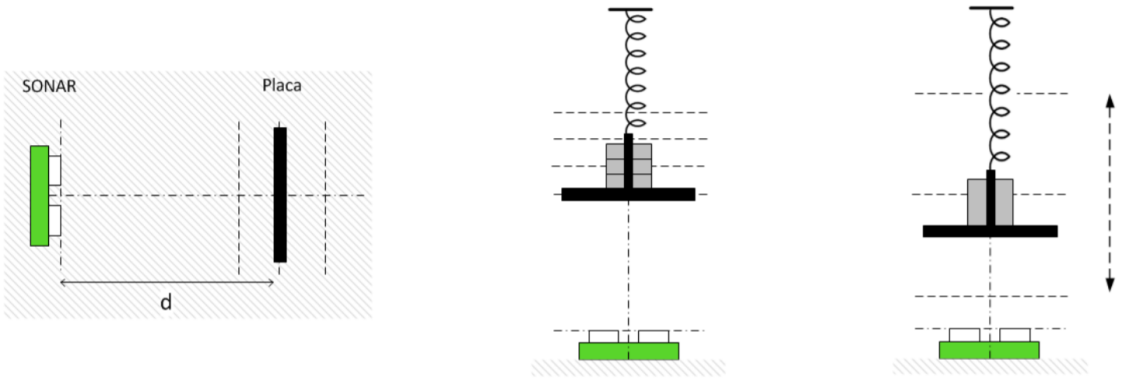


Figura 1: Montagens Experimentais.

3.1 Calibração do SONAR

O SONAR permite medir o tempo de eco, t_{eco}, definido aqui como o intervalo de tempo entre a emissão de um pulso ultrasónico e a sua deteção. Cada pulso emitido pelo SONAR propaga-se com velocidade v_{som}, ao longo de uma distância d, até ser reflectido por uma superfície, e detectado t_{eco} segundos após ter sido emitido:

$$2d = v_{som} t_{eco} \tag{1}$$

A calibração do SONAR consiste em determinar a velocidade do som, v_{som}, podendo assim ser usado posteriormente como sensor de distância.

1. Montagem experimental de acordo com a Figura 1, e registar temperatura ambiente, θ_1;
2. Colocar a placa (caixa CD) a uma distância d do SONAR; medir d com fita métrica, e registar o tempo t_{eco} com SONAR;
3. Repetir para diferentes distâncias, preenchendo a Tabela 1.

3.2 Lei de Hooke

Seja o sistema massa-mola, um corpo de massa m suspenso numa mola vertical, de constante elástica k. Na situação de equilíbrio, a resultande das forças que actuam no corpo (peso e força elástica) é nula. Pela 2ª lei de Newton,

$$mg = kx \tag{2}$$

A elongação sofrida x da mola é proporcional à massa do corpo suspenso (Lei de Hooke). Esta parte do trabalho tem por objectivos a verificação da Lei de Hooke e a determinação experimental da constante elástica da mola.

1. Montagem experimental de acordo com a Figura 1, e registar temperatura ambiente, θ_2;
2. Medir a massa m_0 e comprimento natural l_0 da mola; com a mola suspensa na haste, estimar a distância d_0 entre o SONAR e a extremidade livre da mola;
3. Medir a massa m_1 do suporte com placa; com o suporte suspenso na mola, medir a distância d_1 com SONAR; registar m_1 e d_1 na Tabela 2;
4. Escolher um corpo e medir a sua massa, m; com o corpo colocado no suporte, medir a distância d com o SONAR; repetir para diferentes corpos, preenchendo a Tabela 2.

3.3 Oscilador Amortecido

Nesta parte do trabalho, pretende-se verificar que a equação do movimento de um oscilador livre amortecido pode ser escrita sob a forma:

$$x(t) = Ae^{\frac{-\gamma}{2}t}cos(\omega t + \phi_0) \tag{3}$$

onde $x(t)$ representa a distância do oscilador à sua posição de equilíbrio, A a amplitude máxima de oscilação, γ o coeficiente de amortecimento e ω a frequência (angular) de oscilação, expressa em rad s^{-1}.

1. Montagem experimental de acordo com a Figura 1, e registar temperatura ambiente, θ_3;
2. Colocar um corpo de massa m (aprox. 100 g) no suporte com placa; com o sistema em equilíbrio, medir a distância d cm SONAR; registar m e d;
3. Fazer oscilar o sistema em torno da sua posição de equilíbrio; registar a distância d com o SONAR em função do tempo t, decorrido desde o início do movimento oscilatório;

4 Resultados e Discussão

4.1 Registo de Dados

Tabela 1: Calibração do SONAR.

Distância $d(mm)$	**Tempo** t_{eco} (μs)
100	530
170	950
240	1350
310	1730
380	2140
450	2550
520	2960
590	3380
660	3760
730	4140
800	4570

Tabela 2: Lei de HOOKE.

Massa $m(g)$	**Distância** $d(mm)$
11	534
22	510
33	483
53	441
73	392
92	349
112	306
132	263
152	220
172	174

4.1.1 Cálculo de Grandezas/Incertezas

[1]

Utilizando os valores da Tabela 1 e 2, procede se ao respetivo tratamento de dados:

A relação entre a velocidade do som e a temperatura ambiente pode ser obtida através da seguinte expressão:

$$c = c_0\sqrt{\frac{T}{T_0}} \tag{4}$$

Dado que a velocidade do som no ar a $0C(273.15K)$ é 331.45 ms^{-1},

$$c = 331.45\sqrt{\frac{\theta}{273.15}} \tag{5}$$

Deste modo é possível calcular a temperatura ambiente, θ, utilizando os dados obtidos durante a calibração do SONAR:

$$\theta = \left(\frac{347.92}{331.45}\right)^2 273.15 = 27.86C \tag{6}$$

Com vista a obter os resultados relativos à experiência do oscilador amortecido, procedeu-se ao cálculo das seguintes grandezas:

- Frequência angular própria ($rads^{-1}$):

$$\omega_0 = \sqrt{\frac{k}{m}} = \sqrt{\frac{4.39}{0.210}} = 4.57 rads^{-1} \tag{7}$$

- Período de oscilação (s):

$$T_{est} = \frac{2\pi}{\omega_0} = \frac{2\pi}{4.57} = 1.38s \tag{8}$$

$$T_{cal} = \frac{2\pi}{\omega} = \frac{2\pi}{4.57} = 1.37s \tag{9}$$

- Frequência de oscilação ($rads^{-1}$):

$$\omega = \sqrt{\omega_0^2 - \frac{\gamma^2}{4}} = \sqrt{4.57^2 - \frac{0.08^2}{4}} = 4.57 rads^{-1} \tag{10}$$

- Fator de Qualidade:

$$Q = \frac{\omega_0}{\gamma} = \frac{4.57}{0.08} = 57.125 \tag{11}$$

- Decaimento (amplitude):

$$\tau = \frac{2}{\gamma} = \frac{2}{0.08} = 25s \tag{12}$$

4.2 Registo de Resultados

Tabela 3: Resultados : Calibração do SONAR.

Grandeza	**Simb.**	**Unidade**	**Valor**	**Incerteza**
Temperatura Ambiente	θ_1	C	21	± 1
Velocidade do som	v_{som}	ms^{-1}	347.92	$\pm$
Valor de Referência	v_{ref}	ms^{-1}	343.4	$\pm$

Tabela 4: Resultados : Lei de HOOKE.

Grandeza	**Simb.**	**Unidade**	**Valor**	**Incerteza**
Temperatura Ambiente	θ_2	C	18	± 1
Massa da mola	m_0	g	15	± 0.05
Comprimento natural	l_0	mm	428	± 0.5
Distância inicial	d_0	mm	428	± 0.5
Constante elástica	k	mm	559	$\pm$

Tabela 5: Resultados : Oscliador Amortecido.

Grandeza	**Simb.**	**Unidade**	**Valor**	**Incerteza**
Temperatura Ambiente	θ_3	C	21	± 1
Massa do oscilador	m	g	210	± 0.05
Período de oscilação	T_{est}	s	1.38	$\pm$
Amortecimento (coef.)	γ	$rads_{-1}$	0.08	$\pm$
Frequência própria	ω_0	$rads$	4.57	————
Frequência de oscilação	ω	$rads$	4.57	————
Período de oscilação	T_{cal}	s	1.37	————
Factor de qualidade	Q	————	57.125	————
Decaimento (amplitude)	τ	s	25	————

Calibração do SONAR

Tabela 6: Resultados : Calibração do SONAR.

Regressão Linear (mx+b)	**Valor**
m	347.798
b	0.007
r^2	0.999
Erro Estimado	

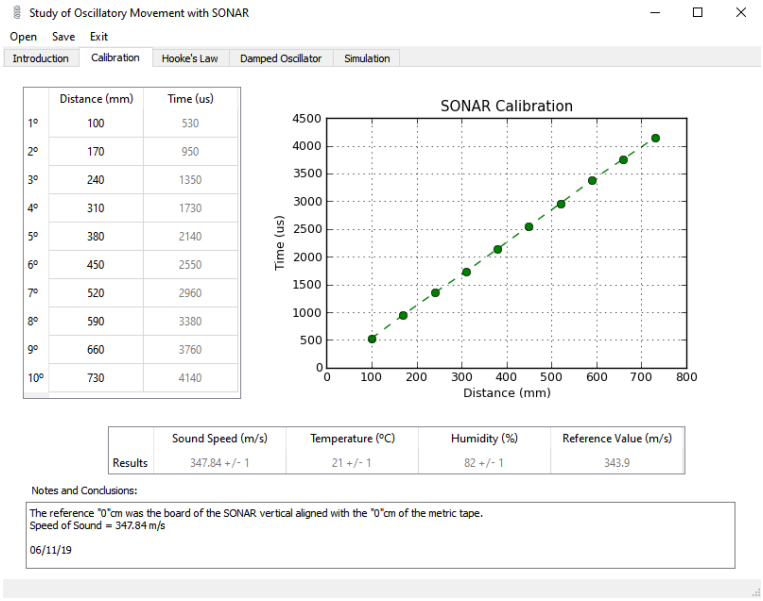


Figura 2: Calibração do SONAR Qt

Lei de Hooke

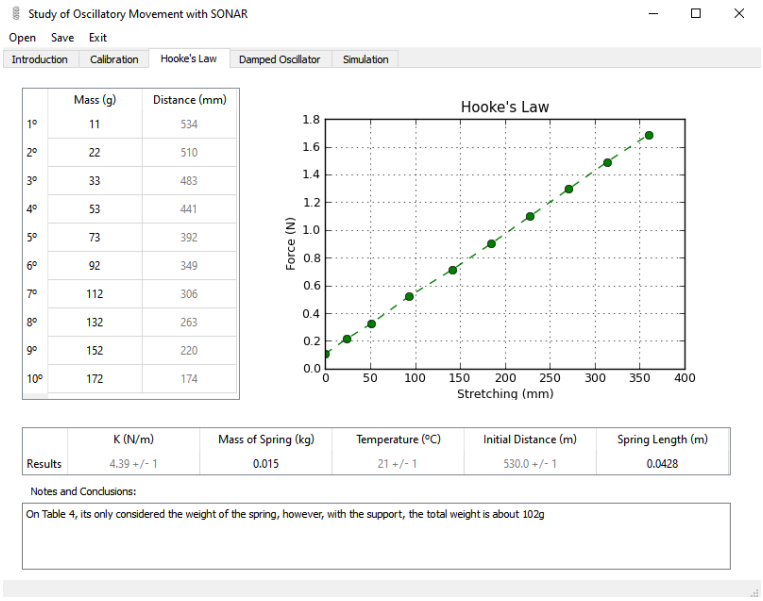


Figura 3: Lei de Hooke Qt

Tabela 7: Resultados : Lei de Hooke.

Regressão Linear (mx+b)	**Valor**
m	4.39
b	0.1036
r^2	
Erro Estimado	

Oscilador Amortecido Qt

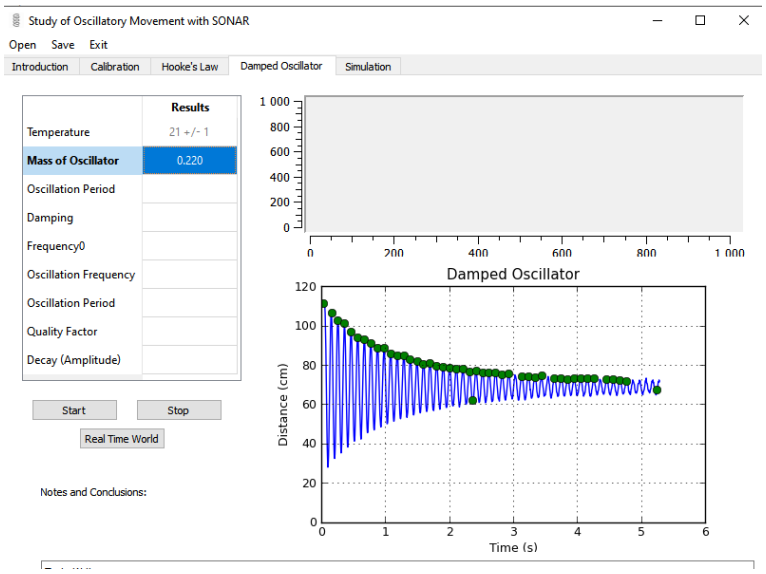


Figura 4: Oscilador Amortecido

Tabela 8: Resultados : Oscilador Amortecido.

Regressão Linear (mx+b)	**Valor**
A	0.94366
γ	0.137
r^2	
Erro Estimado	

5 Conclusão

A elaboração deste trabalho possibilitou uma abordagem mais direcionada para o propósito da Engenharia Eletrotrécnica e de Computadores, culminando na automação de experiências comuns no ramo da física. O projeto final envolve um conjunto de 3 experiências que são possíveis de serem realizadas utilizando apenas um kit *hardware* com desenvolvimento de *software*, baseado em linguagem Python, o que demonstra a versatalidade da sua aplicação em ambiente laboratoriais.

A arquitetura do sistema possibilita uma comunicação de dados sem fios, possuindo uma interface gráfica de utilizador baseada em Qt, o que torna a interação máquina-utilizador bastante versátil.

Dado que todas as experiências foram efetuadas utilizando este sistema, foi possível demonstrar o seu correto funciomaneto, sendo que a aquisição dos dados sem fios se revelou, por vezes, com alguns atrasos. Este facto deve-se, portanto, ao facto da versão do Python utilizada não possuir determinadas bibliotecas que, atualmente, possibilitam uma comunição mais rápida e eficaz, sendo que uma possível melhoria deste projeto poderia envolver a sua aplicação baseada na versão Python 3.7, efetuando as respetivas adaptações.

A nível de *Hardware* uma possível melhoria poderia envolver a aplicação de um SONAR com sensor de temperatura, com uma tensão de funcionamento de 3.3V. Esta modificação além de compactar o circuito, permitiria a remoção do regulador de tensão e do DC-DC Conversor Elevador, dado que a sua aplicação se deve ao facto do SONAR HC-SR04 ser o único componente com uma tensão de funcionamento de 5V, sendo que os restantes apresentam uma tensão de alimentação num intervalo de 3 a 5 V.

Por fim, é ainda importante referir que o sistema tem como fonte de alimentação uma bateria *Lithium-Polymer*, com respetivo controlador e carregador incorporados, possibilitando uma versátil execução das experiências.

Referências

[1] xaktly.com. Hooke's law. http://xaktly.com/HookesLaw.html. Accessed on 2019-12-05.

6 Anexos

6.1 *Hardware*

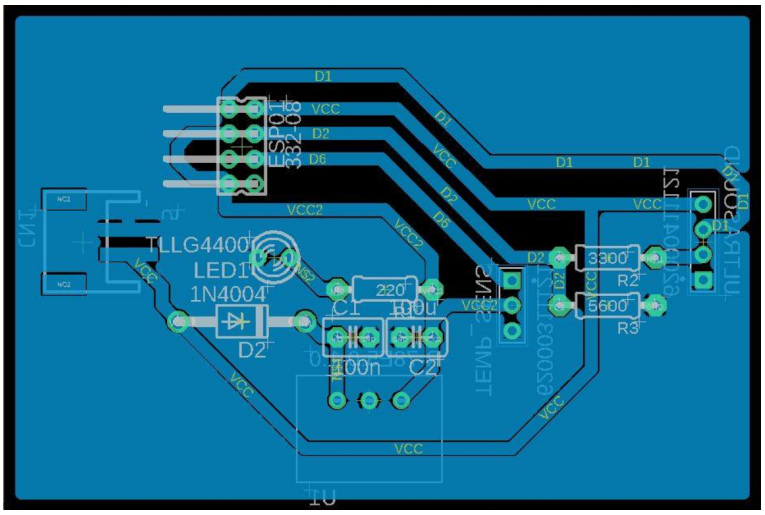


Figura 5: Esquema *EAGLE*

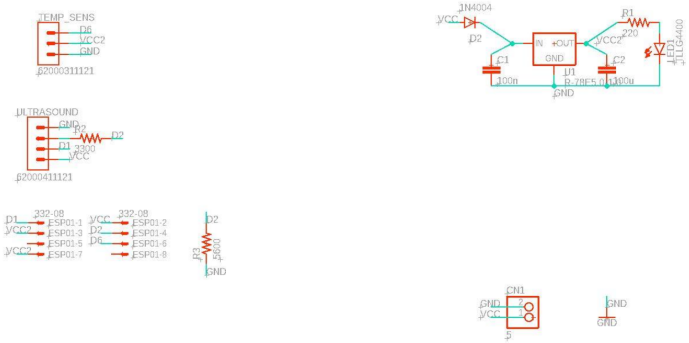


Figura 6: Esquema Eletrónico

6.2 Orçamento *Hardware*

- ESP-01 - 3 €;
- SONAR HC-SR04 - 1.80 €;
- Sensor Temperatura e Humidade DHT11 - 2 €;
- Bateria - 4.15 €;
- Módulo Carregamento Bateria Polímero de lítio - 1.20 €;
- DC-DC Conversor Elevador - 1.20 €;
- Módulo Regulador Tensão - 0.50 €;
- Interruptor - 0.50 €;
- Caixa de montagem - 2.50 €.

Total: 16.85 €.

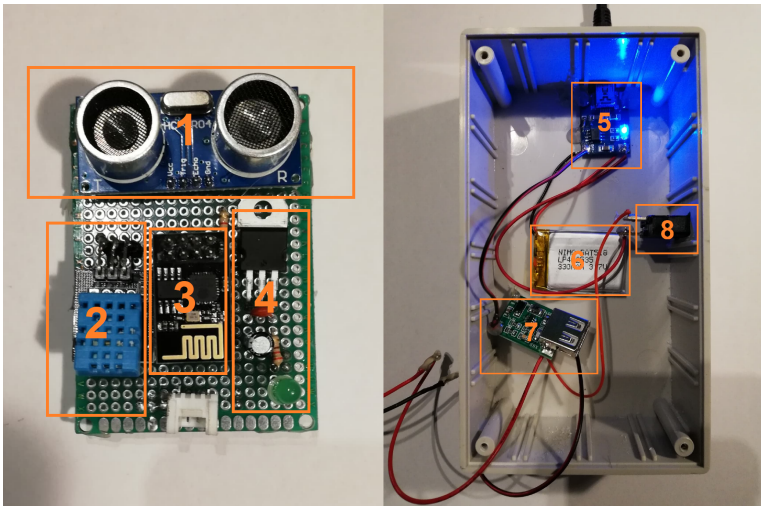


Figura 7: *Hardware*: 1- SONAR. 2- DHT11. 3- ESP-01. 4- Módulo Regulador Tensão. 5- Módulo Carregamento. 6 - Bateria. 7- Conversor Elevador. 8- Interruptor.

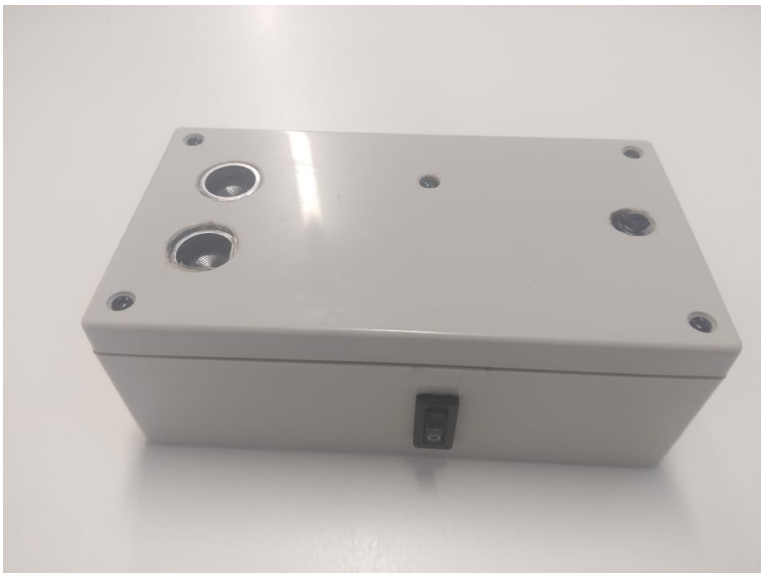


Figura 8: Produto final

6.3 Distribuição de Tarefas

Carlos Silva - Programação Python, Qt, Arduino e montagem hardware.
Diogo Pinto - Elaboração relatório LaTeX, cálculos e montagem hardware.
Pedro Apura - Cálculos e desenvolvimento modelo PCB.

6.4 Código *Python*

```
import sys
import warnings
from PyQt4 import QtGui, uic
from PyQt4.QtGui import QFileDialog, QTextEdit, QPixmap
from PyQt4.Qwt5.Qwt import QwtPlot, QwtPlotCurve
from PyQt4 import QtCore as QtCore

# Libraries
from visual import * #import vPython library
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
import matplotlib.animation as animation
from matplotlibwidget import MatplotlibWidget
from time import sleep
import os
import sympy as sym
import urllib2
import time
import csv
from scipy.signal import find_peaks_cwt
from scipy.optimize import curve_fit

x = []
y = []

def _translate(context, text, disambig):
    return QtGui.QApplication.translate(context, text, disambig, _encoding)
def _translate(context, text, disambig):
    return QtGui.QApplication.translate(context, text, disambig)
    self.retranslateUi(MainWindow)

class MyWindow(QtGui.QMainWindow):

    global d
    global t
    global dPlot
    global tPlot
    global h
    global speed_sound
    global spring_k
#
    def __init__(self):
        super(MyWindow, self).__init__()
        uic.loadUi('App_Designer.ui', self)

        #self.calibrationButton.clicked.connect(self.calibration_handle)
        self.simulation_button.clicked.connect(self.simulation)
        self.tabWidget.setCurrentIndex(0)

        self.calibrationPlot.hide()
        self.hookePlot.hide()
        self.dampedPlot.hide()
        self.dataTable.cellChanged.connect(self.table_handle)
        self.dataTable2.cellChanged.connect(self.table2_handle)
        self.actionCSV_File.triggered.connect(self.save_datacsv)
        self.actionCSV_File_2.triggered.connect(self.open_datacsv)
        self.actionCSV_File_of_Dumping.triggered.connect(self.open_dampedcsv)
        self.dataTable3.cellChanged.connect(self.w0_calc)

```

```
        self.startDamped_Button.clicked.connect(self.startDamped)
        self.stopDamped_Button.clicked.connect(self.damped_stop)
        self.simDamped_Button.clicked.connect(self.damped_sim)

        global d
        d = zeros(10)
        global t
        t = zeros(10)
        global dPlot
        dPlot = []
        global tPlot
        tPlot = []
        global mass
        mass = zeros(10)
        global d2
        d2 = zeros(10)
        global massPlot
        massPlot = []
        global d2Plot
        d2Plot = []
        global curve1

        global xs, ys
        xs = []
        ys = []

        curve1 = QwtPlotCurve("Curve 1")
        #curve1.setData(rand, np.zeros(30))
        curve1.attach(self.qwtPlot)
        #QwtPlot.setAxisTitle(axisId, title)[source]
        #Set title
        self.qwtPlot.show()

        #app = QtGui.QApplication(sys.argv)
        self.show()

        #Take screenshot
        #filename = 'Screenshot.jpg'
        #p = QPixmap.grabWindow(self.Tab_Intro.winId())
        #p.save(filename, 'jpg')
        #QPixmap.grabWindow(self.Tab_Intro.winId()).save(filename, 'jpg')

        #self.simulationPlot = MatplotlibWidget(title='Simulation', xlabel='sal
    ', ylabel='salgrosso')

        self.simulationPlot.hide()
        self.simulationPlot.figure.set_facecolor("white")
        self.simulationPlot.axes.set_title('GRAPH') # Nothing Happens
        self.simulationPlot.axes.plot(x,y)
        self.simulationPlot.axes.set_xlabel('Sal')

        #global ax2
        #ax2 = self.calibrationPlot.axes.twinx()

        self.show()

    def save_datacsv(self):
        #Open Dialog Window
        filename = QtGui.QFileDialog.getSaveFileName(self, 'Save File',
                                                     '~/Desktop',
                                                     'Data files (*.csv)')
        ## Write Data
        with open(filename, mode='wb') as csv_file: #default open as read mode
```

```
            fieldnames = ['Distance_Calibration', 'Time_Calibration',
                          'Temperature_Calibration', 'Humidity_Calibration',
                          'Notes_Calibration',
                          'Mass_Hooke', 'Distance_Hooke',
                          'MassSpring_Hooke', 'Temperature_Hooke',
                          'Length_Hooke','Notes_Hooke',
                          'Temperature_Damped', 'MassOfOscillator_Damped',
                          'OscillationPeriod_Damped', 'Damping_Damped',
                          'Frequency0_Damped', 'OscillationFrequency_Damped',
                          'OscillationPeriod_Damped', 'QualityFactor_Damped',
                          'DecayAmplitude_Damped', 'Notes_Damped']

            writer = csv.DictWriter(csv_file, fieldnames = fieldnames)
            writer.writeheader()
            for i in range(0, 10):
                writer.writerow({'Distance_Calibration': self.dataTable.item(i,
    0).text(),
                                 'Time_Calibration': self.dataTable.item(i, 1).
    text(),
                                 'Temperature_Calibration': self.resultsTable.
    item(0, 1).text(),
                                 'Humidity_Calibration': self.resultsTable.item
    (0, 2).text(),
                                 'Notes_Calibration': self.textEdit.toPlainText
    (),
                                 'Mass_Hooke': self.dataTable2.item(i, 0).text()
    ,
                                 'Distance_Hooke': self.dataTable2.item(i, 1).
    text(),
                                 'MassSpring_Hooke': self.resultsTable2.item(0,
    1).text(),
                                 'Temperature_Hooke': self.resultsTable2.item(0,
     2).text(),
                                 'Length_Hooke': self.resultsTable2.item(0, 4).
    text(),
                                 'Notes_Hooke': self.textEdit_2.toPlainText(),
                                 'Temperature_Damped': self.dataTable3.item(0,
    0).text(),
                                 'MassOfOscillator_Damped': self.dataTable3.item
    (1,0).text(),
                                 'OscillationPeriod_Damped': self.dataTable3.
    item(2,0).text(),
                                 'Damping_Damped': self.dataTable3.item(3, 0).
    text(),
                                 'Frequency0_Damped': self.dataTable3.item(4, 0)
    .text(),
                                 'OscillationFrequency_Damped': self.dataTable3.
    item(5, 0).text(),
                                 'OscillationPeriod_Damped': self.dataTable3.
    item(6, 0).text(),
                                 'QualityFactor_Damped': self.dataTable3.item(7,
     0).text(),
                                 'DecayAmplitude_Damped': self.dataTable3.item
    (8, 0).text(),
                                 'Notes_Damped': self.textEdit_3.toPlainText()
                                })


    def open_datacsv(self):
        global speed_sound
        global spring_k

        tPlotOpen = np.zeros(10)
```

```
        dPlotOpen = np.zeros(10)
        massPlotOpen = np.zeros(10, dtype=float)
        d2PlotOpen = np.zeros(10)

        self.dataTable.cellChanged.disconnect(self.table_handle)
        self.dataTable2.cellChanged.disconnect(self.table2_handle)

        filename = QFileDialog.getOpenFileName(self,
                                               'Open File',
                                               '~/Desktop',
                                               'Data files (*.csv)')

        i = 0

        with open(filename, mode='rb') as csv_file:
            csv_reader = csv.DictReader(csv_file)
            for row in csv_reader:
                self.dataTable.item(i, 0).setText(_translate("MyWindow", row['Distance_Calibration'], None))
                self.dataTable.item(i, 1).setText(_translate("MyWindow", row['Time_Calibration'], None))
                self.resultsTable.item(0, 1).setText(row['Temperature_Calibration'])
                self.resultsTable.item(0, 2).setText(_translate("MyWindow", row['Humidity_Calibration'], None))
                self.textEdit.setPlainText(_translate("MyWindow", row['Notes_Calibration'], None))

                self.dataTable2.item(i, 0).setText(_translate("MyWindow", row['Mass_Hooke'], None))
                self.dataTable2.item(i, 1).setText(_translate("MyWindow", row['Distance_Hooke'], None))
                self.resultsTable2.item(0, 1).setText(row['MassSpring_Hooke'])
                self.resultsTable2.item(0, 2).setText(row['Temperature_Hooke'])
                self.resultsTable2.item(0, 4).setText(row['Length_Hooke'])
                self.textEdit_2.setPlainText(_translate("MyWindow", row['Notes_Hooke'], None))

                self.dataTable3.item(0, 0).setText(_translate("MyWindow", row['Temperature_Damped'], None))
                self.dataTable3.item(1, 0).setText(_translate("MyWindow", row['MassOfOscillator_Damped'], None))
                self.dataTable3.item(2, 0).setText(_translate("MyWindow", row['OscillationPeriod_Damped'], None))
                self.dataTable3.item(3, 0).setText(_translate("MyWindow", row['Damping_Damped'], None))
                self.dataTable3.item(4, 0).setText(_translate("MyWindow", row['Frequency0_Damped'], None))
                self.dataTable3.item(5, 0).setText(_translate("MyWindow", row['OscillationFrequency_Damped'], None))
                self.dataTable3.item(6, 0).setText(_translate("MyWindow", row['OscillationPeriod_Damped'], None))
                self.dataTable3.item(7, 0).setText(_translate("MyWindow", row['QualityFactor_Damped'], None))
                self.dataTable3.item(8, 0).setText(_translate("MyWindow", row['DecayAmplitude_Damped'], None))
                self.textEdit_3.setPlainText(_translate("MyWindow", row['Notes_Damped'], None))

                i=i+1

        for i in range(0, 10):
            tPlotOpen[i] = int(self.dataTable.item(i, 0).text())
```

```
            dPlotOpen[i] = float(self.dataTable.item(i, 1).text())

        for i in range(0, 10):
            massPlotOpen[i] = float(self.dataTable2.item(i, 0).text()) * 9.80665
 / 1000
            d2PlotOpen[i] = float(self.dataTable2.item(i, 1).text())

        initDistance = float(d2PlotOpen[0]/1000)

        for i in range(len(d2PlotOpen)-1, -1, -1):
            d2PlotOpen[i] = d2PlotOpen[0] - d2PlotOpen[i]

        self.calibrationPlot.figure.set_facecolor("white")
        self.calibrationPlot.axes.plot(tPlotOpen, dPlotOpen,'go--')
        self.calibrationPlot.axes.set_title('SONAR Calibration') # Nothing
Happens
        self.calibrationPlot.axes.set_xlabel('Distance (mm)')
        self.calibrationPlot.axes.set_ylabel('Time (us)')
        self.calibrationPlot.axes.grid()
        self.calibrationPlot.axes.set_xlim(xmin=0)
        self.calibrationPlot.axes.set_ylim(ymin=0)
        self.calibrationPlot.show()

        self.hookePlot.figure.set_facecolor("white")
        self.hookePlot.axes.plot(d2PlotOpen, massPlotOpen,'go--')
        self.hookePlot.axes.set_title("Hooke's Law") # Nothing Happens
        self.hookePlot.axes.set_xlabel('Stretching (mm)')
        self.hookePlot.axes.set_ylabel('Force (N)')
        self.hookePlot.axes.grid()
        self.hookePlot.axes.set_xlim(xmin=0)
        self.hookePlot.axes.set_ylim(ymin=0)
        self.hookePlot.show()

        self.dataTable.cellChanged.connect(self.table_handle)
        self.dataTable2.cellChanged.connect(self.table2_handle)

        m, b = np.polyfit(tPlotOpen, dPlotOpen, 1);
        m = (1/m)*1000*2

        speed_sound = round(m, 2)

        #m, b = np.polyfit(massPlotOpen, d2PlotOpen, 1);
        m, b = np.polyfit(d2PlotOpen, massPlotOpen, 1);
        spring_k = round(m*1000, 2)

        ref_value = 331.3 + (0.6*float(self.resultsTable.item(0, 1).text()[:3]))

        self.resultsTable.item(0, 0).setText(_translate("MyWindow", str(
speed_sound) + ' +/- 1', None))
        self.resultsTable.item(0, 3).setText(_translate("MyWindow", str(
ref_value), None))

        self.resultsTable2.item(0, 0).setText(_translate("MyWindow", str(
spring_k) + ' +/- 1', None))
        self.resultsTable2.item(0, 3).setText(_translate("MyWindow", str(
initDistance) + ' +/- 1', None))

    def w0_calc(self):
        #m = float(self.dataTable3.item(1, 0).text())
        #spring_k = float(self.resultsTable2.item(0, 0).text())
        m = 0.220
        spring_k = 4.39
        w0 = round(np.sqrt(spring_k/m), 3)
```

```
        self.dataTable3.item(4, 0).setText(_translate("MyWindow", str(w0), None)
    )

        ts = round(2*pi/w0, 3)

        self.dataTable3.item(2, 0).setText(_translate("MyWindow", str(ts), None)
    )

    def open_dampedcsv(self):
        def Read_Two_Column_File(file_name):
            with open(file_name, 'r') as data:
                x = []
                y = []
                for line in data:
                    p = line.split()
                    x.append(float(p[0]))
                    y.append(float(p[1]))

            return x, y

        x, y = Read_Two_Column_File('matrix.dat')

        t = [i / 10000 for i in x]
        d = [i * speed_sound / 10000 for i in y]

        t = np.array(t)
        d = np.array(d)

        peakind = find_peaks_cwt(d, np.arange(0.001/max(d), 20))
        peakind2 = np.array(peakind)

        periodo = np.diff(t[peakind2])
        periodo = np.mean(periodo)

        print("Period between peaks=", periodo)

        def func(x, a, b):
            return a*np.exp(-b*x)


        popt, pcov = curve_fit(func, t[peakind2], d[peakind2])

        print(popt, pcov)

        print("A=", popt[0]/100)

        print("Gama=", popt[1]*2)

        gama = popt[1]*2

        self.dataTable3.item(3, 0).setText(_translate("MyWindow", str(round(gama
    , 3)), None))

        qf = 4.467 / gama

        self.dataTable3.item(7, 0).setText(_translate("MyWindow", str(round(qf,
    3)), None))

        e = func(t, popt[0], popt[1])

        self.dampedPlot.figure.set_facecolor("white")
        self.dampedPlot.axes.plot(t, d,'-')
        self.dampedPlot.axes.hold(True)
```

```
328         self.dampedPlot.axes.plot(t[peakind2], d[peakind2], 'o')
329         self.dampedPlot.axes.plot(t, e, 'r-')
330         self.dampedPlot.axes.set_title('Damped Oscillator') # Nothing Happens
331         self.dampedPlot.axes.set_xlabel('Time (s)')
332         self.dampedPlot.axes.set_ylabel('Distance (cm)')
333         self.dampedPlot.axes.grid()
334         self.dampedPlot.axes.set_xlim(xmin=0)
335         self.dampedPlot.axes.set_ylim(ymin=0)
336         self.dampedPlot.show()
337         self.dampedPlot.axes.hold(False)
338 
339     def getData(self):
340         resp = urllib2.urlopen('http://192.168.4.1')
341         thePage = resp.read()
342         thePageArray = thePage.split(',')
343         time = int(thePageArray[1])
344         temperature = float(thePageArray[2])
345         temperature = int(round(temperature, 0))
346         humidity = float(thePageArray[3])
347         humidity = int(round(humidity, 0))
348         return time, temperature, humidity
349 
350     def table_handle(self):
351         global d
352         global t
353         global dPlot
354         global tPlot
355         global h
356         global speed_sound
357 
358         tData, tempData, humData = self.getData()
359         tSonar = float(tData)/2.0
360 
361         cur = self.dataTable.currentRow()
362         self.dataTable.cellChanged.disconnect(self.table_handle)
363         self.dataTable.item(cur, 1).setText(_translate("MyWindow", str(tSonar),
    None))
364         self.dataTable.cellChanged.connect(self.table_handle)
365 
366         d[cur] = (float(self.dataTable.item(cur, 0).text())/100)
367         t[cur] = float(tSonar)
368 
369         if(len(dPlot) < len(d) and len(dPlot) < (cur+1)):
370             for j in range(len(dPlot), len(d)):
371                 if d[j] != 0:
372                     dPlot.append(d[j])
373                 if t[j] != 0:
374                     tPlot.append(t[j])
375         else:
376             dPlot[cur] = d[cur]
377             tPlot[cur] = t[cur]
378 
379         m, b = np.polyfit(tPlotOpen, dPlotOpen, 1);
380         m = (1/m)*1000*2
381         speed_sound = round(m, 2)
382 
383         ref_value = 331.3 + (0.6*tempData)
384 
385         self.resultsTable.item(0, 0).setText(_translate("MyWindow", str(
    speed_sound) + ' +/- 1', None))
386         self.resultsTable.item(0, 1).setText(_translate("MyWindow", str(tempData
    ) + ' +/- 1', None))
```

```
387         self.resultsTable.item(0, 2).setText(_translate("MyWindow", str(humData)
      + ' +/- 1', None))
388         self.resultsTable.item(0, 3).setText(_translate("MyWindow", str(
    ref_value), None))
389 
390         h = zeros(size(dPlot))
391 
392         for k in range(0, len(dPlot)):
393             h[k] = (dPlot[k]*m+b)
394 
395         def animate_cal(i):
396             global dPlot
397             global tPlot
398             global h
399 
400             dPlot2 = [x*100 for x in dPlot]
401 
402             self.calibrationPlot.figure.set_facecolor("white")
403             self.calibrationPlot.axes.plot(dPlot2, tPlot,'go--')
404             self.calibrationPlot.axes.set_title('SONAR Calibration') # Nothing
    Happens
405             self.calibrationPlot.axes.set_xlabel('Distance (mm)')
406             self.calibrationPlot.axes.set_ylabel('Time (us)')
407             self.calibrationPlot.axes.set_xlim(xmin=0)
408             self.calibrationPlot.axes.set_ylim(ymin=0)
409             self.calibrationPlot.axes.grid()
410             #ax2.plot(dPlot, h, 'b-', label="Linear Regression")
411 
412         ani = animation.FuncAnimation(self.calibrationPlot.figure, animate_cal,
    interval=100)
413         self.calibrationPlot.show()
414 
415     def table2_handle(self):
416         global d2
417         global mass
418         global d2Plot
419         global massPlot
420         global h2
421         global spring_k
422         global speed_sound
423 
424         tData, tempData, humData = self.getData()
425 
426         tSonar = float(tData)/2.0
427         dSonar = round((tSonar / 1000000)* speed_sound * 100, 2)
428 
429         cur = self.dataTable2.currentRow()
430         self.dataTable2.cellChanged.disconnect(self.table2_handle)
431         self.dataTable2.item(cur, 1).setText(_translate("MyWindow", str(dSonar),
     None))
432         if cur == 0:
433             self.resultsTable2.item(0, 3).setText(_translate("MyWindow", str(
    dSonar) + ' +/- 1', None))
434         self.dataTable2.cellChanged.connect(self.table2_handle)
435 
436         mass[cur] = (float(self.dataTable2.item(cur, 0).text()))
437         d2[cur] = dSonar
438 
439         if(len(massPlot) < len(mass) and len(massPlot) < (cur+1)):
440             for j in range(len(massPlot), len(mass)):
441                 if mass[j] != 0:
442                     massPlot.append(mass[j])
443                 if d2[j] != 0:
```

```
                    d2Plot.append(d2[j])
        else:
            massPlot[cur] = mass[cur]
            d2Plot[cur] = d2[cur]

        m, b = np.polyfit(d2Plot, massPlot, 1);
        spring_k = round(m, 2)

        self.resultsTable2.item(0, 0).setText(_translate("MyWindow", str(
spring_k) + ' +/- 1', None))
        self.resultsTable2.item(0, 2).setText(_translate("MyWindow", str(
tempData) + ' +/- 1', None))

        def animate_hooke(i):
            global d2Plot
            global massPlot

            #d2Plot2 = [x*100 for x in dPlot]

            self.hookePlot.figure.set_facecolor("white")
            self.hookePlot.axes.plot(d2Plot, massPlot,'go--')
            self.hookePlot.axes.set_title("Hooke's Law") # Nothing Happens
            self.hookePlot.axes.set_xlabel('Distance (mm)')
            self.hookePlot.axes.set_ylabel('Mass (g)')
            self.hookePlot.axes.grid()

        anie = animation.FuncAnimation(self.hookePlot.figure, animate_hooke,
interval=100)
        self.hookePlot.show()

    def startDamped(self):
        # set up the QwtPlot (pay attention!)
        self.timer = QtCore.QTimer() #start a timer (to call replot events)
        self.timer.start(5.0) #set the interval (in ms)
        self.qwtPlot.connect(self.timer, QtCore.SIGNAL('timeout()'), self.
damped_start)
        global tempo1
        tempo1 = time.time() # Time reference

    def damped_start(self):
        yD, _, _ = self.getData()

        global tempo, tempo1, curve1, xs, ys
        global speed_sound

        tempo = time.time() - tempo1
        # Limit to 30 samples displayed
        if(len(xs) == 50):
            xs.pop(0)
        if(len(ys) == 50):
            ys.pop(0)

        xs.append(tempo)
        ys.append(yD*speed_sound/100000)

        print(yD*speed_sound/100000)

        curve1.setData(xs, ys)
        self.qwtPlot.setTitle('Damping Visualization')
        self.qwtPlot.setAxisTitle(0, 'Distance (cm)')
        self.qwtPlot.setAxisTitle(2, 'Time (s)')
        self.qwtPlot.replot()

```

```
503     def damped_stop(self):
504         self.qwtPlot.disconnect(self.timer, QtCore.SIGNAL('timeout()'), self.
    damped_start)
505 
506         _, temp, _ = self.getData()
507 
508         self.dataTable3.item(0, 0).setText(_translate("MyWindow", str(temp) + '
    +/- 1', None))
509 
510     def damped_sim(self):
511         xs = []
512         ys = []
513 
514         global speed_sound
515 
516         # Scene Configuration
517 
518         MyScene=display(title='Real Time 3D Spring Simulation') #Create your
    scene and give it a title.
519         MyScene.width=800  #We can set the dimension of your visual box. 800X800
     pixels works well on my screen
520         MyScene.height= 800
521 
522 
523         #MyScene.autoscale=False #We want to set the range of the scene manually
     for better control. Turn autoscale off
524         #MyScene.range = (12,12,12) #Set range of your scene to be 12 inches by
    12 inches by 12 inches.
525 
526         #Structure Creation
527 
528         support_base = box(pos=(0, -1, 0), size=(6, 0.1, 6), material =
    materials.wood)
529 
530         #support_base.visible = False
531 
532         support_tall = box(pos=(  support_base.pos.x, 1.5 + support_base.pos.y
    -(support_base.size.y/2)+18.5,
533                                   support_base.pos.z - support_base.size.z
    /2 - support_base.size.y/2),
534                                           size=(6, 40, 0.1), material =
    materials.wood)
535 
536         support_ceil = box(pos=(support_base.pos.x, support_tall.size.y +
    support_base.pos.y, support_base.pos.z - support_tall.size.z),
537                                           size=(6, 0.1, 6), material =
    materials.wood)
538 
539         spring = helix(pos=(support_ceil.pos.x, support_ceil.pos.y, support_ceil
    .pos.z), axis=(0, -1, 0),
540                 radius=1, coils=8, thickness = 0.01, stiffness=1, length = 4.28)
541 
542         eq_pos = spring.pos + spring.axis
543 
544         endSpring  =    box(pos=eq_pos, color=color.yellow,  size=(0.75, 0.01,
    0.5) ,
545                             mass=1.0, velocity=vector(0, 0.1, 0),
546                             force=(0, 0, 0), make_trail=True, retain = 10,
    trail_color = color.green, opacity = 0.5)
547 
548         #ball = sphere(  pos=eq_pos, color=color.red, size=(0.75, 0.1, 0.5))
549                         # mass=1.0, velocity=(0, 0.1, 0),
550                         # force=(0, 0, 0)                    )
```

```
551
552         sonar_base  =    box(pos=(support_base.pos.x,support_base.pos.y+
    support_base.size.y, support_base.pos.z-0.1), size=(4.5, 0.1, 2), color =
    color.blue)
553
554         sonar_echo  =    cylinder(color=color.green, pos=(sonar_base.pos.x - 1,
    sonar_base.pos.y, sonar_base.pos.z), radius=0.75, length=1, axis=(0, 1, 0) )
555
556         sonar_trigger   =  cylinder(color=color.green, pos=(sonar_base.pos.x +
    1, sonar_base.pos.y, sonar_base.pos.z), radius=0.75,length=1, axis=(0, 1, 0)
    )
557
558         labl =          label(pos=(1, 1, 0), text="Simulation 3D") # Stays
    always in the same place
559
560         dist = endSpring.pos.y - sonar_base.pos.y
561
562         dist_label =    label(pos=(1, 0.5, 0)) # Stays always in the same place
563
564         ## Camera Controls
565
566         #print(scene.camera.pos)
567         #print(scene.camera.axis)
568
569         #scene. = vector(0, 20, 0)
570         scene.camera.axis = vector(0, 0, 20)
571
572         #scene.camera.follow(endSpring)
573
574         dt = 0.01 # Time step
575
576 #        while(True):
577 #            rate(1000)
578 #
579 #            tData, tempData, humData = self.getData()
580 #            yD = tData * speed_sound / 1000
581 #
582 #            endSpring.pos = (0, yD/1000, 0)
583 #            spring.axis = endSpring.pos - spring.pos
584 #            dist = endSpring.pos.y - sonar_base.pos.y
585 #            dist_label.text = ("Distance: " + "{0:.2f}".format(dist) + "cm")
586
587     def simulation(self):
588         scene = display(title='Simple Harmonic Oscillator', autoscale=0,
589                 width=500, height=500, range=7, forward=vector(0,0,-1))
590         chao = box(pos=vector(0,-6.1,0),size=vector(20,0.1,10),material=
    materials.wood)
591         parede = box(pos=vector(0,3.85,-5.05),size=vector(20,20,0.1), color=
    vector(0.7,0.7,0.7))
592         #sitio = text(pos=vector(0,3.85,-5), text='Carlos Silva', color=color.
    blue, align='center', depth=0)
593
594         # Mola e cilindro
595         bar0 = curve(radius=0.03)
596         for ang in arange(pi,-pi/2.,-0.1):
597             bar0.append(vector(0, 5.2+0.23*sin(ang), 0.23*cos(ang)))
598         bar0.append(pos=vector(0,4.5,0))
599         bar0.append(pos=vector(0,4.5,0.3))
600         mola1 = helix(pos=vector(0,4.5,0), radius=0.3, thickness=0.05, coils=30,
601                       axis = vector(0,-5.8,0))
602
603         bar1 = curve(radius=0.03, pos=[vector(0,-1.6,0),vector(0,-1.3,0),vector
    (0,-1.3,0.3)])
```

```
        c1 = cylinder(pos=vector(0,-2,0),radius=0.5,axis=vector(0,0.4,0),
                      color=vector(0.3,0.3,0.3))

        # Barras do suporte
        s1 = cylinder(pos=vector(3,5.2,0),radius=0.2, axis=vector(-4,0,0))
        s2 = cylinder(pos=vector(2.5,5.2,0),radius=0.6, axis=vector(-1,0,0),
    color=vector(0.5,0.5,0.6))
        s3 = cylinder(pos=vector(2,-5,0.4),radius=0.2, axis=vector(0,11,0))
        s4 = cylinder(pos=vector(2,-6.05,0.4),radius=0.8, axis=vector(0,1,0),
    color=vector(0.9,0.9,0.6))

        # Funcaoo que alonga/comprime a mola y1 unidades
        def alongar_mola(y1):
            c1.pos.y = y1 - 2
            bar1.origin = vector(0, y1, 0)
            mola1.axis.y = y1 - 5.8

        def uf(rf):          # velocidade no espaco de fase
            a = (-k1*rf.y-b1*rf.z)/m1
            return vector(0,rf.z, a)

        rf = vector(0, 0.9, 0)
        m1 = 0.3; k1 = 16; b1 = 0.02
        dt = 0.01
        alongar_mola(rf.y)

        while True:
            rate(100)
            uf1 = uf(rf)
            uf2 = uf(rf + dt*uf1 / 2.)
            uf3 = uf(rf + dt*uf2 / 2.)
            uf4 = uf(rf + dt*uf3)
            ufm = (uf1 + 2*uf2 + 2*uf3 + uf4)/6.
            rf += dt*ufm
            alongar_mola(rf.y)

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    window = MyWindow()
    sys.exit(app.exec_())
```

6.5 Código *Arduino*

```
/* Create a WiFi access point and provide a web server on it. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Adafruit_Sensor.h>
#include <WebSocketsServer.h>

#define DHT_PIN 0
#define DHTTYPE DHT11

DHT dht(DHT_PIN, DHTTYPE);

float t =0.0;
float h = 0.0;

//SimpleDHT11 dht11(2); USAR GPIO0

#define TRIGGER_PIN  1  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2  // Arduino pin tied to echo pin on the ultrasonic sensor.
int cnt=0;

#ifndef APSSID
#define APSSID "ESPap"
#define APPSK  "thereisnospoon"
#endif

/* Set these to your desired credentials. */
const char *ssid = APSSID;
const char *password = APPSK;

ESP8266WebServer server(80);

/* Just a little test message.  Go to http://192.168.4.1 in a web browser
   connected to this access point to see it.
*/

void handleRoot() {
      String out;

      t = dht.readTemperature();
      h = dht.readHumidity();
      digitalWrite(TRIGGER_PIN, HIGH);  // SONAR trigger
      delayMicroseconds( 10 );        // wait 10 us
      digitalWrite(TRIGGER_PIN, LOW);

      /*  echo time, t_eco (us) */
      unsigned long t_eco = pulseIn(ECHO_PIN, HIGH);
      cnt = cnt +1;
      out = String(cnt) + ", " + String(t_eco) + "," + String(t) + "," + String(h);

      server.send(200, "text/html", out);
}

void setup() {
```

```
  delay(1000);

  //GPIO1 (TX) swap the pin to a GPIO (TX -> FUNCTION_0)
  pinMode(1, FUNCTION_3);
  //GPIO3 (RX) swap the pin to a GPIO (RX -> FUNCTION_0)
  pinMode(3, FUNCTION_3);

  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  //dht.begin();

  /* You can remove the password parameter if you want the AP to be open. */
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  server.on("/", handleRoot);
  server.begin();
}

void loop() {
  server.handleClient();
}
```

6.6 *Datasheet ESP-01*

ESP-01 WiFi Module

Version1.0

sherry@aithinker.com

Disclaimer and Copyright Notice.

Notice

Product version upgrades or other reasons, possible changes in the contents of this manual. AI-Thinker reserves in the absence of any notice or indication of the circumstances the right to modify the content of this manual. This manual is used only as a guide, Ai-thinker make every effort to provide accurate information in this manual, but Ai-thinker does not ensure that manual content without error, in this manual all statements, information and advice nor does it constitute any express or implied warranty.

Table of Contents

1. Preambles

ESP-01 WiFi module is developed by Ai-thinker Team. core processor ESP8266 in smaller sizes of the module encapsulates Tensilica L106 integrates industry-leading ultra low power 32-bit MCU micro, with the 16-bit short mode, Clock speed support 80 MHz, 160 MHz, supports the RTOS, integrated Wi-Fi MAC/BB/RF/PA/LNA, on-board antenna.

The module supports standard IEEE802.11 b/g/n agreement, complete TCP/IP protocol stack. Users can use the add modules to an existing device networking, or building a separate network controller.

ESP8266 is high integration wireless SOCs, designed for space and power constrained mobile platform designers. It provides unsurpassed ability to embed Wi-Fi capabilities within other systems, or to function as a standalone application, with the lowest cost, and minimal space requirement.

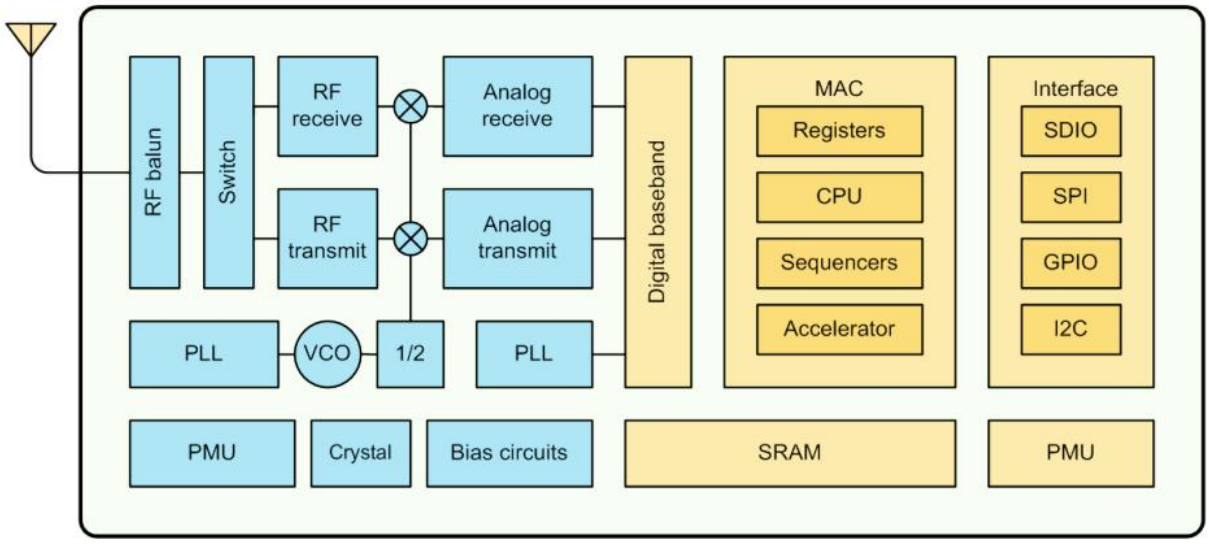


Figure 1 ESP8266EX Block Diagram

ESP8266EX offers a complete and self-contained Wi-Fi networking solution; it can be used to host the application or to offload Wi-Fi networking functions from another application processor.

When ESP8266EX hosts the application, it boots up directly from an external flash. In has integrated cache to improve the performance of the system in such applications.

Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any micro controllerbased design with simple connectivity (SPI/SDIO or I2C/UART interface).

ESP8266EX is among the most integrated WiFi chip in the industry; it integrates the antenna switches, RF balun, power amplifier, low noise receive amplifier, filters, power management modules, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area.

ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor, with on-chip SRAM, besides the Wi-Fi functionalities. ESP8266EX is often integrated with external sensors and other application specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

Espressif Systems' Smart Connectivity Platform (ESCP) demonstrates sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing. for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

1.1. Features

- 802.11 b/g/n
- Integrated low power 32-bit MCU
- Integrated 10-bit ADC
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- Integrated PLL, regulators, and power management units
- Supports antenna diversity
- Wi-Fi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- Support Smart Link Function for both Android and iOS devices
- Support Smart Link Function for both Android and iOS devices
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IRDA, PWM, GPIO

- STBC, 1x1 MIMO, 2x1 MIMO
- A-MPDU & A-MSDU aggregation and 0.4s guard interval
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C

1.2. Parameters

Table 1 below describes the major parameters.

Table 1 Parameters

Categories	Items	Values
WiFi Paramters	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4GHz-2.5GHz (2400M-2483.5M)
Hardware Paramaters	Peripheral Bus	UART/HSPI/I2C/I2S/Ir Remote Contorl
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	14.3mm*24.8mm*3mm
	External Interface	N/A
Software Parameters	Wi-Fi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network) / download and write firmware via host
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

2. Pin Descriptions

There are altogether 8 pin counts, the definitions of which are described in Table 2 below。

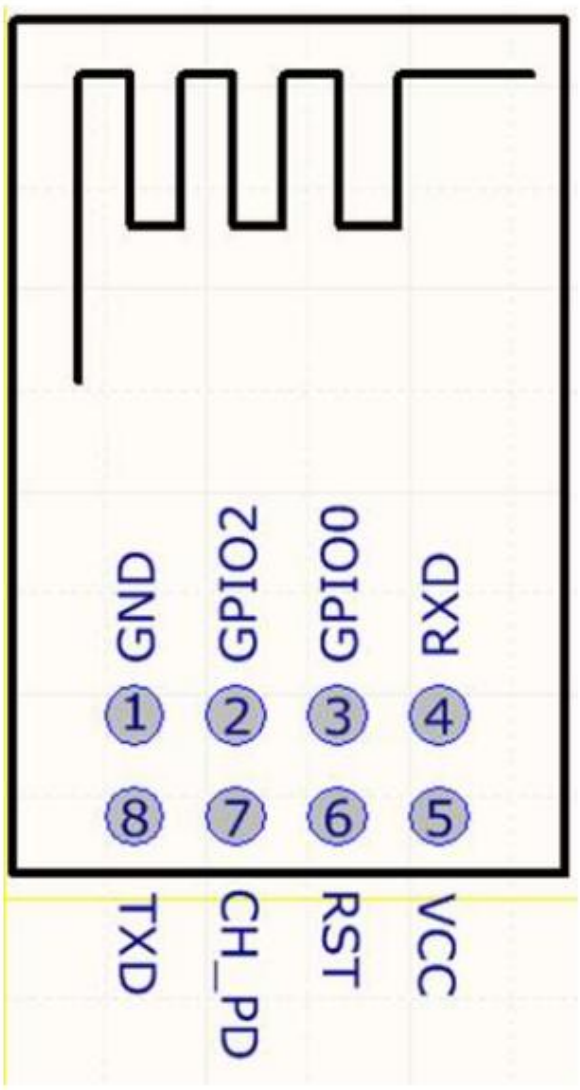


Table 2 ESP-01 Pin design

Table 2 Pin Descriptions

NO.	Pin Name	Function
1	GND	GND
2	GPIO2	GPIO,Internal Pull-up
3	GPIO0	GPIO,Internal Pull-up
4	RXD	UART0,data received pin RXD
5	VCC	3.3V power supply (VDD)
6	RST	1) External reset pin, active low 2) Can loft or external MCU ,
7	CH_PD	Chip enable pin. Active high
8	TXD	UART0,data send pin RXD

Table 3 Pin Mode

Mode	GPIO15	GPIO0	GPIO2
UART	Low	Low	High
Flash Boot	Low	High	High

Table 4 Receiver Sensitivity

Parameters	Min	Typical	Max	Unit
Input frequency	2412		2484	MHz
Input impedance		50		Ω
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
DSSS, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

3. Packaging and Dimension

The external size of the module is 14.3mm*24.8mm*3mm, as is illustrated in Figure 3 below. The type of flash integrated in this module is an SPI flash, the capacity of which is 1 MB, and the package size of which is SOP-210mil. The antenna applied on this module is a 3DBi PCB-on-board antenna.

Figure 3 [Module Pin Counts, 8 pin, 14.3 mm *24.8 mm *3.0 mm]

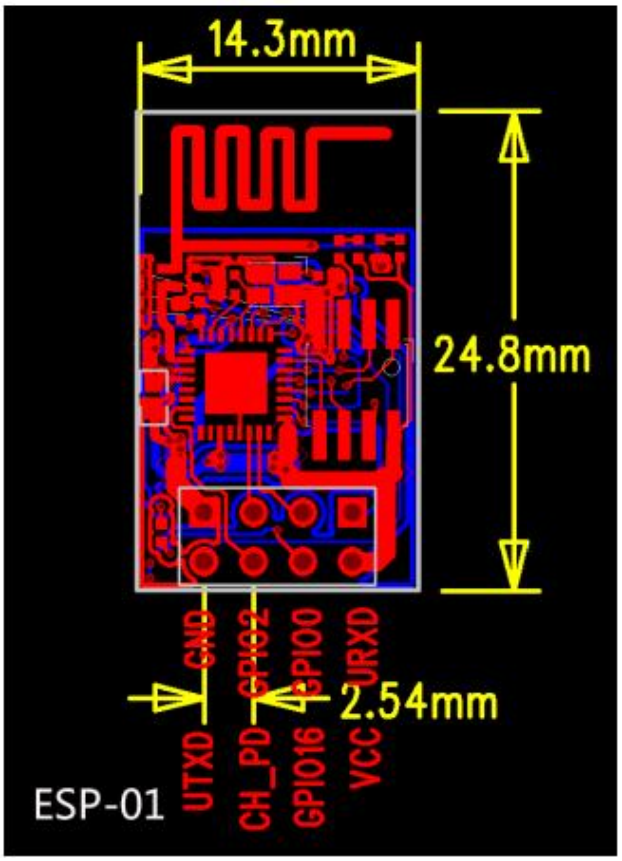


Figure 4 Top View of ESP-01 WiFi Module

Table 5 Dimension of ESP-01 WiFi Module

Length	Width	Height	PAD Size(Bottom)	Pin Pitch
14.3 mm	24.8 mm	3 mm	0.9 mm x 1.7 mm	2.54 mm

4. Functional Descriptions

4.1. MCU

ESP8266EX is embedded with Tensilica L106 32-bit micro controller (MCU), which features extra low power consumption and 16-bit RSIC. The CPU clock speed is 80MHz. It can also reach a maximum value of 160MHz. ESP8266EX is often integrated with external sensors and other specific devices through its GPIOs; codes for such applications are provided in examples in the SDK.

4.2. Memory Organization

4.2.1. Internal SRAM and ROM

ESP8266EX WiFi SoC is embedded with memory controller, including SRAM and ROM. MCU can visit the memory units through iBus, dBus, and AHB interfaces. All memory units can be visited upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK provided, SRAM space that is available to users is assigned as below:

▪RAM size < 36kB, that is to say, when ESP8266EX is working under the station mode and is connected to the router, programmable space accessible to user in heap and data section is around 36kB.)

▪ There is no programmable ROM in the SoC, therefore, user program must be stored in an external SPI flash.

4.2.2. External SPI Flash

This module is mounted with an 1 MB external SPI flash to store user programs. If larger definable storage space is required, a SPI flash with larger memory size is preferred. Theoretically speaking, up to 16 MB memory capacity can be supported.

Suggested SPI Flash memory capacity:

▪OTA is disabled: the minimum flash memory that can be supported is 512 kB;

▪OTA is enabled: the minimum flash memory that can be supported is 1 MB.

Several SPI modes can be supported, including Standard SPI, Dual SPI, and Quad SPI.

Therefore, please choose the correct SPI mode when you are downloading into the flash, otherwise firmwares/programs that you downloaded may not work in the right way.

4.3. Crystal

Currently, the frequency of crystal oscillators supported include 40MHz, 26MHz and 24MHz. The accuracy of crystal oscillators applied should be ±10PPM, and the operating temperature range should be between -20°C and 85°C.

When using the downloading tools, please remember to select the right crystal oscillator type. In circuit design, capacitors C1 and C2, which are connected to the earth, are added to the input and output terminals of the crystal oscillator respectively. The values of the two capacitors can be flexible, ranging from 6pF to 22pF, however, the specific capacitive values of C1 and C2 depend on further testing and adjustment on the overall performance of the whole circuit. Normally, the capacitive values of C1 and C2 are within 10pF if the crystal oscillator frequency is 26MHz, while the values of C1 and C2 are 10pF<C1, C2<22pF if the crystal oscillator frequency is 40MHz.

4.4. Interfaces

Table 6 Descriptions of Interfaces

Interface	Pin Name	Description
HSPI	IO12(MISO) IO13(MOSI) IO14(CLK) IO15(CS)	SPI Flash 2, display screen, and MCU can be connected using HSPI interface.
PWM	IO12(R) IO15(G) IO13(B)	Currently the PWM interface has four channels, but users can extend the channels according to their own needs. PWM interface can be used to control LED lights, buzzers, relays, electronic machines, and so on.
IR Remote Control	IO14(IR_T) IO5(IR_R)	The functionality of Infrared remote control interface can be implemented via software programming. NEC coding, modulation, and demodulation are used by this interface. The frequency of modulated carrier signal is 38KHz.
ADC	TOUT	ESP8266EX integrates a 10-bit analog ADC. It can be used to test the power-supply voltage of VDD3P3 (Pin3 and Pin4) and the input power voltage of TOUT (Pin 6). However, these two functions cannot be used simultaneously. This interface is typically used in sensor products.
I2C	IO14(SCL) IO2(SDA)	I2C interface can be used to connect external sensor products and display screens, etc.

Interface	Pin Name	Description
UART	**UART0:** TXD (U0TXD) RXD (U0RXD) IO15 (RTS) IO13 (CTS) **UART1:** IO2(TXD)	Devices with UART interfaces can be connected with the module. Downloading: U0TXD+U0RXD or GPIO2+U0RXD Communicating: UART0: U0TXD, U0RXD, MTDO (U0RTS), MTCK (U0CTS) Debugging: UART1_TXD (GPIO2) can be used to print debugging information.
		By default, UART0 will output some printed information when the device is powered on and is booting up. If this issue exerts influence on some specific applications, users can exchange the inner pins of UART when initializing, that is to say, exchange U0TXD, U0RXD with U0RTS, U0CTS.
I2S	**I2S Input:** IO12 (I2SI_DATA) ; IO13 (I2SI_BCK); IO14 (I2SI_WS); **I2S Output:** IO15 (I2SO_BCK); IO3 (I2SO_DATA); IO2 (I2SO_WS).	I2S interface is mainly used for collecting, processing, and transmission of audio data.

4.5. Absolute Maximum Ratings

Table 7 Absolute Maximum Ratings

Rating	Condition	Value	Unit
Storage Temperature		-40 to 125	°C
Maximum Soldering Temperature		260	°C
Supply Voltage	IPC/JEDEC J-STD-020	+3.0 to +3.6	V

4.6. Recommended Operating Conditions

Table 8 Recommended Operating Conditions

Operating Condition	Symbol	Min	Typ	Max	Unit
Operating Temperature		-40	20	125	°C
Supply voltage	VDD	3.0	3.3	3.6	V

4.7. Digital Terminal Characteristics

Table 9 Digital Terminal Characteristics

Terminals	Symbol	Min	Typ	Max	Unit
Input logic level low	V_{IL}	-0.3		0.25VDD	V
Input logic level high	V_{IH}	0.75VDD		VDD+0.3	V
Output logic level low	V_{OL}	N		0.1VDD	V
Output logic level high	V_{OH}	0.8VDD		N	V

Note: Test conditions: VDD = 3.3V, Temperature = 20 ℃, if nothing special is stated.

5. RF Performance

Description	Min.	Typ.	Max	Unit
Input frequency	2400		2483.5	MHz
Input impedance		50		ohm
Input reflection			-10	dB
Output power of PA for 72.2Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
Sensitivity				
CCK, 1Mbps		-98		dBm
CCK, 11Mbps		-91		dBm
6Mbps (1/2 BPSK)		-93		dBm
54Mbps (3/4 64-QAM)		-75		dBm
HT20, MCS7 (65Mbps, 72.2Mbps)		-72		dBm
Adjacent Channel Rejection				
OFDM, 6Mbps		37		dB
OFDM, 54Mbps		21		dB
HT20, MCS0		37		dB
HT20, MCS7		20		dB

Table 10 RF Performance

6. Power Consumption

Parameters	Min	Typical	Max	Unit
Tx802.11b, CCK 11Mbps, P OUT=+17dBm		170		mA
Tx 802.11g, OFDM 54Mbps, P OUT =+15dBm		140		mA
Tx 802.11n, MCS7, P OUT =+13dBm		120		mA
Rx 802.11b, 1024 bytes packet length , -80dBm		50		mA
Rx 802.11g, 1024 bytes packet length, -70dBm		56		mA
Rx 802.11n, 1024 bytes packet length, -65dBm		56		mA
Modem-Sleep①		15		mA
Light-Sleep②		0.9		mA
Deep-Sleep③		10		uA

Table 11 Power Consumption

❶Modem-Sleep requires the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it saves power to shut down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission. E.g. in DTIM3, to maintain a sleep 300mswake 3ms cycle to receive AP's Beacon packages, the current is about 15mA.

❷ During Light-Sleep, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power according to the 802.11 standard (U-APSD). E.g. in DTIM3, to maintain a sleep 300ms-wake 3ms cycle to receive AP's Beacon packages, the current is about 0.9mA.

❸ Deep-Sleep does not require Wi-Fi connection to be maintained. For application with long time lags between data transmission, e.g. a temperature sensor that checks the temperature every 100s ,sleep 300s and waking up to connect to the AP (taking about 0.3~1s), the overall average current is less than 1mA.

7. Reflow Profile

Table 12 Instructions

T_S max to T_L (Ramp-up Rate)	3°C/second max
Preheat Temperature Min.(T_S Min.) Temperature Typical.(T_S Typ.) Temperature Min.(T_S Max.) Time(T_S)	 150°C 175°C 200°C 60~180 seconds
Ramp-up rate (T_L to T_P)	3°C/second max
Time Maintained Above: --Temperature(T_L)/Time(T_L)	 217°C/60~150 seconds
Peak Temperature(T_P)	260°C max. for 10 seconds
Target Peak Temperature (T_P Target)	260°C +0/-5°C
Time within 5°C of actual peak(t_P)	20~40 seconds
T_S max to T_L (Ramp-down Rate)	6°C/second max
Tune 25°C to Peak Temperature (t)	8 minutes max

8. Schematics

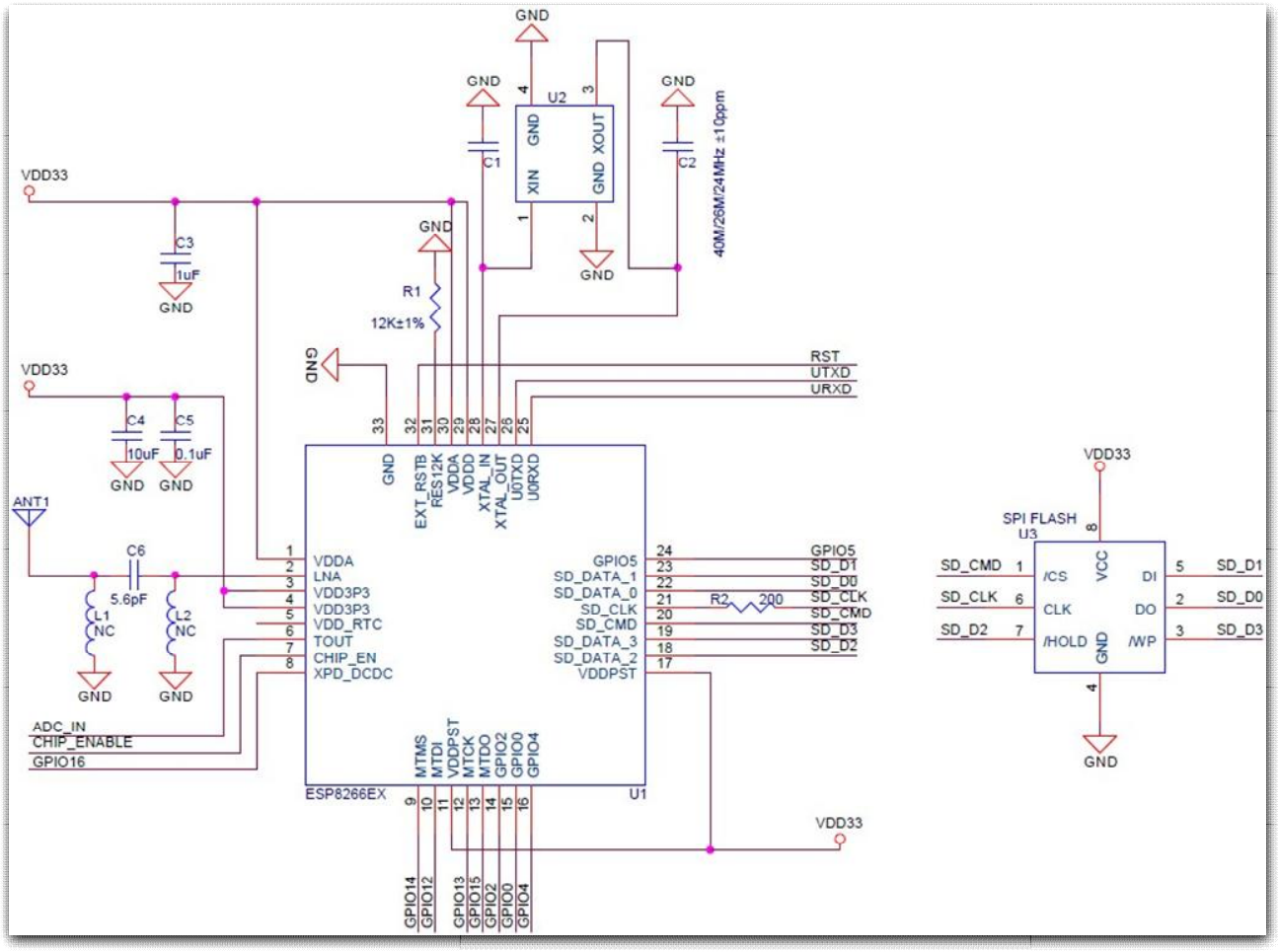


Figure 4 Schematics of Esp-01 WiFi Module

6.7 *Datasheet SONAR HC-SR04*

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

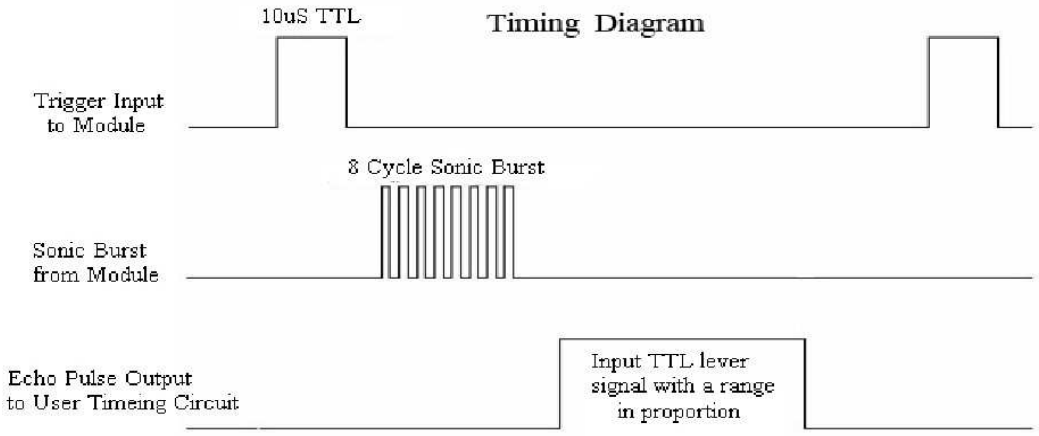
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

SparkFun Electronics:
SEN-13959

6.8 *Datasheet DHT11*

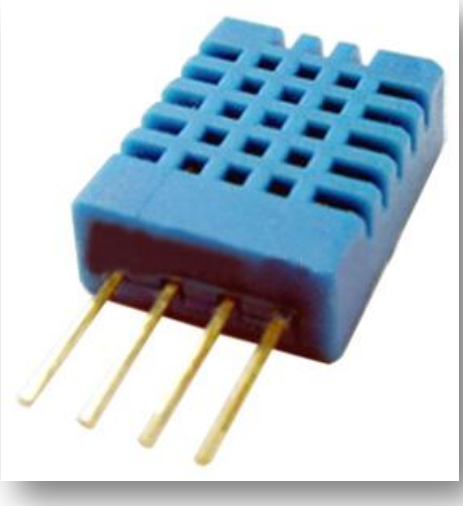
DHT11 Humidity & Temperature Sensor

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output.

DHT 11 Humidity & Temperature Sensor

1. Introduction

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

Detailed Specifications:

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

3. Typical Application (Figure 1)

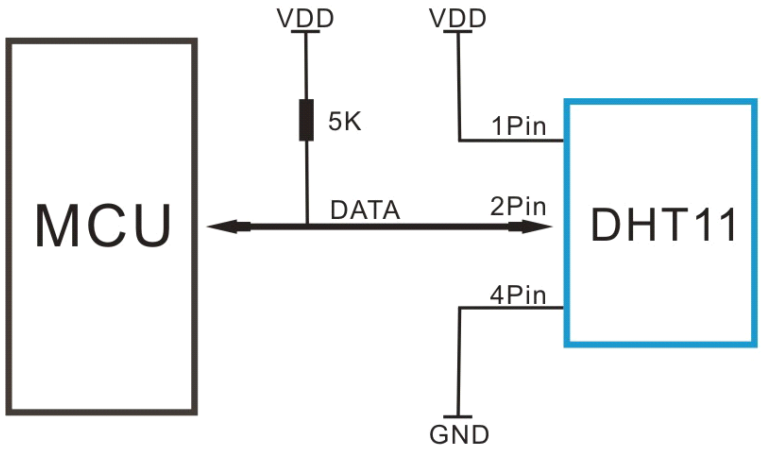


Figure 1 Typical Application

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.
Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

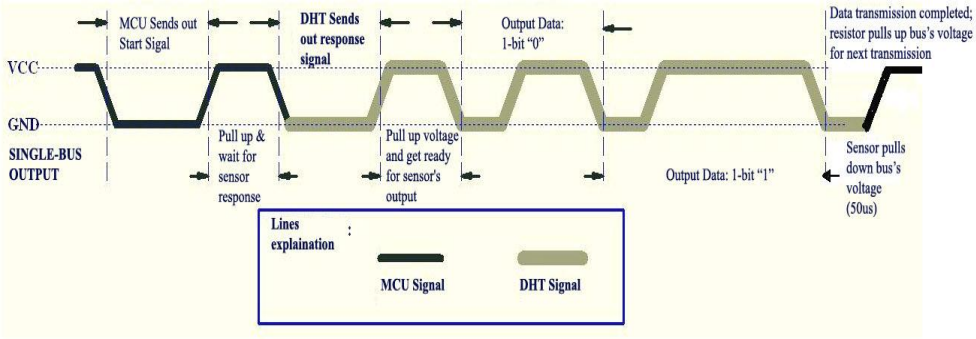


Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

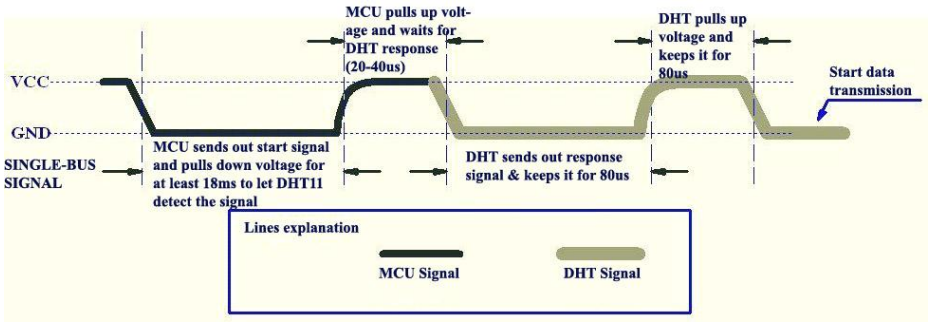


Figure 3 MCU Sends out Start Signal & DHT Responses

5.3 DHT Responses to MCU (Figure 3, above)

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).

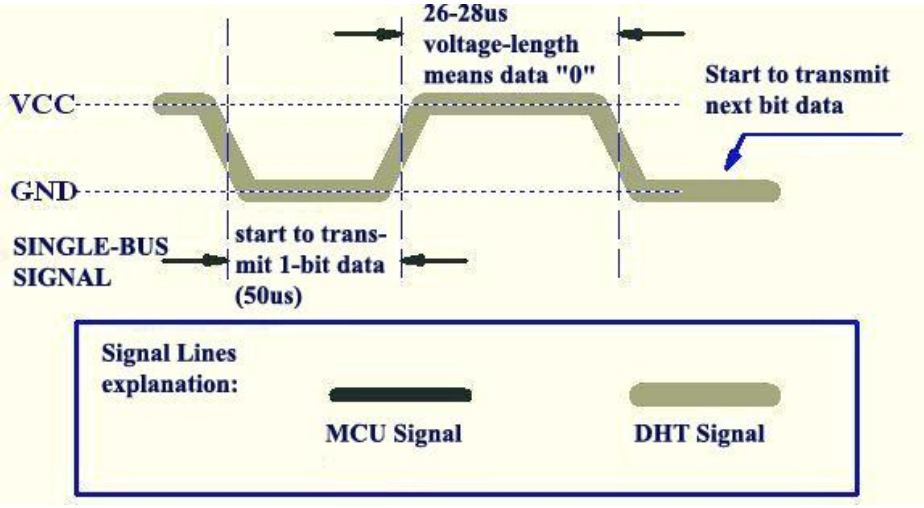


Figure 4 Data "0" Indication

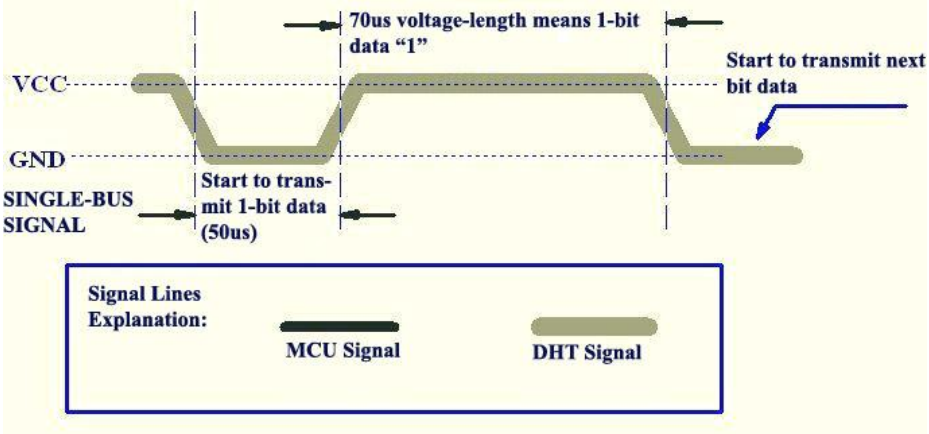


Figure 5 Data "1" Indication

If the response signal from DHT is always at high-voltage-level, it suggests that DHT is not responding properly and please check the connection. When the last bit data is transmitted, DHT11 pulls down the voltage level and keeps it for 50us. Then the Single-Bus voltage will be pulled up by the resistor to set it back to the free status.

6. Electrical Characteristics

VDD=5V, T = 25℃ (unless otherwise stated)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

7. Attentions of application

(1) Operating conditions

Applying the DHT11 sensor beyond its working range stated in this datasheet can result in 3%RH signal shift/discrepancy. The DHT11 sensor can recover to the calibrated status gradually when it gets back to the normal operating condition and works within its range. Please refer to (3) of

this section to accelerate its recovery. Please be aware that operating the DHT11 sensor in the non-normal working conditions will accelerate sensor's aging process.

(2) Attention to chemical materials

Vapor from chemical materials may interfere with DHT's sensitive-elements and debase its sensitivity. A high degree of chemical contamination can permanently damage the sensor.

(3) Restoration process when (1) & (2) happen

Step one: Keep the DHT sensor at the condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;
Step two:K keep the DHT sensor at the condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Temperature Affect

Relative humidity largely depends on temperature. Although temperature compensation technology is used to ensure accurate measurement of RH, it is still strongly advised to keep the humidity and temperature sensors working under the same temperature. DHT11 should be mounted at the place as far as possible from parts that may generate heat.

(5) Ligh t Affect

Long time exposure to strong sunlight and ultraviolet may debase DHT's performance.

(6) Connection wires

The quality of connection wires will affect the quality and distance of communication and high quality shielding-wire is recommended.

(7) Other attentions

* Welding temperature should be bellow 260Celsius and contact should take less than 10 seconds.
* Avoid using the sensor under dew condition.
* Do not use this product in safety or emergency stop devices or any other occasion that failure of DHT11 may cause personal injury.
* Storage: Keep the sensor at temperature 10-40℃, humidity <60%RH.

Disclaimer

This is a translated version of the manufacturer's data sheet. OSEPP is not responsible for the accuracy of the translated information.

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

OSEPP Electronics:
HUMI-01