**OpenCV安装**

1. 配置并更新树莓派系统

sudo raspi-config // 进入后打开摄像头、SSH

sudo apt-get update

sudo apt-get upgrade

sudo rpi-update

**2. 安装OpenCV的相关工具**

sudo apt-get install build-essential cmake git pkg-config

**3. 安装OpenCV的图像工具包**

sudo apt-get install libjpeg8-dev

sudo apt-get install libtiff5-dev

sudo apt-get install libjasper-dev

sudo apt-get install libpng12-dev

**4. 安装视频I/O包**

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

**5.安装gtk2.0和优化函数包**

sudo apt-get install libgtk2.0-dev

sudo apt-get install libatlas-base-dev gfortran

**6. 下载OpenCV源码**

可以在 [ OpenCV ] 查看所有版本源码

git clone https://github.com/opencv/opencv.git

**7. 安装OpenCV**

// 根据下载的版本而定

cd opencv-3.2.0

// 创建release文件夹

mkdir release

// 进入release目录下

cd release

// cmake读入所有源文件之后，自动生成makefile

cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

-D CMAKE\_INSTALL\_PREFIX=/usr/local ..

// 编译

sudo make

// 安装

sudo make install

//更新动态链接库

sudo ldconfig

**8. 解决无法打开摄像头硬件问题**

sudo nano /etc/modules

// 进入编辑界面后，在末尾添加输入

snd-bcm2835

bcm2835-v4l2

**SIFT 算法代码**

# coding: utf-8

import warnings

warnings.filterwarnings("ignore") #忽略警告

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

import os

import math

from scipy.misc import imresize

def convlove(filter,mat,padding,strides):

result = None

filter\_size = filter.shape

mat\_size = mat.shape

if len(filter\_size) == 2:

if len(mat\_size) == 3:

channel = []

for i in range(mat\_size[-1]):

pad\_mat = np.pad(mat[:,:,i], ((padding[0], padding[1]), (padding[2], padding[3])), 'constant')

temp = []

for j in range(0,mat\_size[0],strides[1]):

temp.append([])

for k in range(0,mat\_size[1],strides[0]):

val = (filter\*pad\_mat[j\*strides[1]:j\*strides[1]+filter\_size[0],

k\*strides[0]:k\*strides[0]+filter\_size[1]]).sum()

temp[-1].append(val)

channel.append(np.array(temp))

channel = tuple(channel)

result = np.dstack(channel)

elif len(mat\_size) == 2:

channel = []

pad\_mat = np.pad(mat, ((padding[0], padding[1]), (padding[2], padding[3])), 'constant')

for j in range(0, mat\_size[0], strides[1]):

channel.append([])

for k in range(0, mat\_size[1], strides[0]):

val = (filter \* pad\_mat[j \* strides[1]:j \* strides[1] + filter\_size[0],

k \* strides[0]:k \* strides[0] + filter\_size[1]]).sum()

channel[-1].append(val)

result = np.array(channel)

return result

def downsample(img,step = 2):

return img[::step,::step]

def GuassianKernel(sigma , dim):

'''

:param sigma: Standard deviation

:param dim: dimension(must be positive and also an odd number)

:return: return the required Gaussian kernel.

'''

temp = [t - (dim/2) for t in range(dim)]

assistant = []

for i in range(dim):

assistant.append(temp)

assistant = np.array(assistant)

temp = 2\*sigma\*sigma

result = (1.0/(temp\*np.pi))\*np.exp(-(assistant\*\*2+(assistant.T)\*\*2)/temp)

return result

def getDoG(img,n,sigma0,S = None,O = None):

'''

:param img: the original img.

:param sigma0: sigma of the first stack of the first octave. default 1.52 for complicate reasons.

:param n: how many stacks of feature that you wanna extract.

:param S: how many stacks does every octave have. S must bigger than 3.

:param k: the ratio of two adjacent stacks' scale.

:param O: how many octaves do we have.

:return: the DoG Pyramid

'''

if S == None:

S = n + 3

if O == None:

O = int(np.log2(min(img.shape[0], img.shape[1]))) - 3

k = 2 \*\* (1.0 / n)

sigma = [[(k\*\*s)\*sigma0\*(1<<o) for s in range(S)] for o in range(O)]

samplePyramid = [downsample(img, 1 << o) for o in range(O)]

GuassianPyramid = []

for i in range(O):

GuassianPyramid.append([])

for j in range(S):

dim = int(6\*sigma[i][j] + 1)

if dim % 2 == 0:

dim += 1

GuassianPyramid[-1].append(convlove(GuassianKernel(sigma[i][j], dim),samplePyramid[i],[dim/2,dim/2,dim/2,dim/2],[1,1]))

DoG = [[GuassianPyramid[o][s+1] - GuassianPyramid[o][s] for s in range(S - 1)] for o in range(O)]

return DoG,GuassianPyramid

def adjustLocalExtrema(DoG,o,s,x,y,contrastThreshold,edgeThreshold,sigma,n,SIFT\_FIXPT\_SCALE):

SIFT\_MAX\_INTERP\_STEPS = 5

SIFT\_IMG\_BORDER = 5

point = []

img\_scale = 1.0 / (255 \* SIFT\_FIXPT\_SCALE)

deriv\_scale = img\_scale \* 0.5

second\_deriv\_scale = img\_scale

cross\_deriv\_scale = img\_scale \* 0.25

img = DoG[o][s]

i = 0

while i < SIFT\_MAX\_INTERP\_STEPS:

if s < 1 or s > n or y < SIFT\_IMG\_BORDER or y >= img.shape[1] - SIFT\_IMG\_BORDER or x < SIFT\_IMG\_BORDER or x >= img.shape[0] - SIFT\_IMG\_BORDER:

return None,None,None,None

img = DoG[o][s]

prev = DoG[o][s - 1]

next = DoG[o][s + 1]

dD = [ (img[x,y + 1] - img[x, y - 1]) \* deriv\_scale,

(img[x + 1, y] - img[x - 1, y]) \* deriv\_scale,

(next[x, y] - prev[x, y]) \* deriv\_scale ]

v2 = img[x, y] \* 2

dxx = (img[x, y + 1] + img[x, y - 1] - v2) \* second\_deriv\_scale

dyy = (img[x + 1, y] + img[x - 1, y] - v2) \* second\_deriv\_scale

dss = (next[x, y] + prev[x, y] - v2) \* second\_deriv\_scale

dxy = (img[x + 1, y + 1] - img[x + 1, y - 1] - img[x - 1, y + 1] + img[x - 1, y - 1]) \* cross\_deriv\_scale

dxs = (next[x, y + 1] - next[x, y - 1] - prev[x, y + 1] + prev[x, y - 1]) \* cross\_deriv\_scale

dys = (next[x + 1, y] - next[x - 1, y] - prev[x + 1, y] + prev[x - 1, y]) \* cross\_deriv\_scale

H=[ [dxx, dxy, dxs],

[dxy, dyy, dys],

[dxs, dys, dss]]

X = np.matmul(np.linalg.pinv(np.array(H)),np.array(dD))

xi = -X[2]

xr = -X[1]

xc = -X[0]

if np.abs(xi) < 0.5 and np.abs(xr) < 0.5 and np.abs(xc) < 0.5:

break

y += int(np.round(xc))

x += int(np.round(xr))

s += int(np.round(xi))

i+=1

if i >= SIFT\_MAX\_INTERP\_STEPS:

return None,x,y,s

if s < 1 or s > n or y < SIFT\_IMG\_BORDER or y >= img.shape[1] - SIFT\_IMG\_BORDER or x < SIFT\_IMG\_BORDER or x >= \

img.shape[0] - SIFT\_IMG\_BORDER:

return None, None, None, None

t = (np.array(dD)).dot(np.array([xc, xr, xi]))

contr = img[x,y] \* img\_scale + t \* 0.5

if np.abs( contr) \* n < contrastThreshold:

return None,x,y,s

# 利用Hessian矩阵的迹和行列式计算主曲率的比值

tr = dxx + dyy

det = dxx \* dyy - dxy \* dxy

if det <= 0 or tr \* tr \* edgeThreshold >= (edgeThreshold + 1) \* (edgeThreshold + 1) \* det:

return None,x,y,s

point.append((x + xr) \* (1 << o))

point.append((y + xc) \* (1 << o))

point.append(o + (s << 8) + (int(np.round((xi + 0.5)) \* 255) << 16))

point.append(sigma \* np.power(2.0, (s + xi) / n)\*(1 << o) \* 2)

return point,x,y,s

def GetMainDirection(img,r,c,radius,sigma,BinNum):

expf\_scale = -1.0 / (2.0 \* sigma \* sigma)

X = []

Y = []

W = []

temphist = []

for i in range(BinNum):

temphist.append(0.0)

# 图像梯度直方图统计的像素范围

k = 0

for i in range(-radius,radius+1):

y = r + i

if y <= 0 or y >= img.shape[0] - 1:

continue

for j in range(-radius,radius+1):

x = c + j

if x <= 0 or x >= img.shape[1] - 1:

continue

dx = (img[y, x + 1] - img[y, x - 1])

dy = (img[y - 1, x] - img[y + 1, x])

X.append(dx)

Y.append(dy)

W.append((i \* i + j \* j) \* expf\_scale)

k += 1

length = k

W = np.exp(np.array(W))

Y = np.array(Y)

X = np.array(X)

Ori = np.arctan2(Y,X)\*180/np.pi

Mag = (X\*\*2+Y\*\*2)\*\*0.5

# 计算直方图的每个bin

for k in range(length):

bin = int(np.round((BinNum / 360.0) \* Ori[k]))

if bin >= BinNum:

bin -= BinNum

if bin < 0:

bin += BinNum

temphist[bin] += W[k] \* Mag[k]

# smooth the histogram

# 高斯平滑

temp = [temphist[BinNum - 1], temphist[BinNum - 2], temphist[0], temphist[1]]

temphist.insert(0, temp[0])

temphist.insert(0, temp[1])

temphist.insert(len(temphist), temp[2])

temphist.insert(len(temphist), temp[3]) # padding

hist = []

for i in range(BinNum):

hist.append((temphist[i] + temphist[i+4]) \* (1.0 / 16.0) + (temphist[i+1] + temphist[i+3]) \* (4.0 / 16.0) + temphist[i+2] \* (6.0 / 16.0))

# 得到主方向

maxval = max(hist)

return maxval,hist

def LocateKeyPoint(DoG,sigma,GuassianPyramid,n,BinNum = 36,contrastThreshold = 0.04,edgeThreshold = 10.0):

SIFT\_ORI\_SIG\_FCTR = 1.52

SIFT\_ORI\_RADIUS = 3 \* SIFT\_ORI\_SIG\_FCTR

SIFT\_ORI\_PEAK\_RATIO = 0.8

SIFT\_INT\_DESCR\_FCTR = 512.0

# SIFT\_FIXPT\_SCALE = 48

SIFT\_FIXPT\_SCALE = 1

KeyPoints = []

O = len(DoG)

S = len(DoG[0])

for o in range(O):

for s in range(1,S-1):

threshold = 0.5\*contrastThreshold/(n\*255\*SIFT\_FIXPT\_SCALE)

img\_prev = DoG[o][s-1]

img = DoG[o][s]

img\_next = DoG[o][s+1]

for i in range(img.shape[0]):

for j in range(img.shape[1]):

val = img[i,j]

eight\_neiborhood\_prev = img\_prev[max(0, i - 1):min(i + 2, img\_prev.shape[0]), max(0, j - 1):min(j + 2, img\_prev.shape[1])]

eight\_neiborhood = img[max(0, i - 1):min(i + 2, img.shape[0]), max(0, j - 1):min(j + 2, img.shape[1])]

eight\_neiborhood\_next = img\_next[max(0, i - 1):min(i + 2, img\_next.shape[0]), max(0, j - 1):min(j + 2, img\_next.shape[1])]

if np.abs(val) > threshold and \

((val > 0 and (val >= eight\_neiborhood\_prev).all() and (val >= eight\_neiborhood).all() and (val >= eight\_neiborhood\_next).all())

or (val < 0 and (val <= eight\_neiborhood\_prev).all() and (val <= eight\_neiborhood).all() and (val <= eight\_neiborhood\_next).all())):

point,x,y,layer = adjustLocalExtrema(DoG,o,s,i,j,contrastThreshold,edgeThreshold,sigma,n,SIFT\_FIXPT\_SCALE)

if point == None:

continue

scl\_octv = point[-1]\*0.5/(1 << o)

omax,hist = GetMainDirection(GuassianPyramid[o][layer],x,y,int(np.round(SIFT\_ORI\_RADIUS \* scl\_octv)),SIFT\_ORI\_SIG\_FCTR \* scl\_octv,BinNum)

mag\_thr = omax \* SIFT\_ORI\_PEAK\_RATIO

for k in range(BinNum):

if k > 0:

l = k - 1

else:

l = BinNum - 1

if k < BinNum - 1:

r2 = k + 1

else:

r2 = 0

if hist[k] > hist[l] and hist[k] > hist[r2] and hist[k] >= mag\_thr:

bin = k + 0.5 \* (hist[l]-hist[r2]) /(hist[l] - 2 \* hist[k] + hist[r2])

if bin < 0:

bin = BinNum + bin

else:

if bin >= BinNum:

bin = bin - BinNum

temp = point[:]

temp.append((360.0/BinNum) \* bin)

KeyPoints.append(temp)

return KeyPoints

def calcSIFTDescriptor(img,ptf,ori,scl,d,n,SIFT\_DESCR\_SCL\_FCTR = 3.0,SIFT\_DESCR\_MAG\_THR = 0.2,SIFT\_INT\_DESCR\_FCTR = 512.0,FLT\_EPSILON = 1.19209290E-07):

dst = []

pt = [int(np.round(ptf[0])), int(np.round(ptf[1]))] # 坐标点取整

cos\_t = np.cos(ori \* (np.pi / 180)) # 余弦值

sin\_t = np.sin(ori \* (np.pi / 180)) # 正弦值

bins\_per\_rad = n / 360.0

exp\_scale = -1.0 / (d \* d \* 0.5)

hist\_width = SIFT\_DESCR\_SCL\_FCTR \* scl

radius = int(np.round(hist\_width \* 1.4142135623730951 \* (d + 1) \* 0.5))

cos\_t /= hist\_width

sin\_t /= hist\_width

rows = img.shape[0]

cols = img.shape[1]

hist = [0.0]\*((d+2)\*(d+2)\*(n+2))

X = []

Y = []

RBin = []

CBin = []

W = []

k = 0

for i in range(-radius,radius+1):

for j in range(-radius,radius+1):

c\_rot = j \* cos\_t - i \* sin\_t

r\_rot = j \* sin\_t + i \* cos\_t

rbin = r\_rot + d / 2 - 0.5

cbin = c\_rot + d / 2 - 0.5

r = pt[1] + i

c = pt[0] + j

if rbin > -1 and rbin < d and cbin > -1 and cbin < d and r > 0 and r < rows - 1 and c > 0 and c < cols - 1:

dx = (img[r, c+1] - img[r, c-1])

dy = (img[r-1, c] - img[r+1, c])

X.append(dx)

Y.append(dy)

RBin.append(rbin)

CBin.append(cbin)

W.append((c\_rot \* c\_rot + r\_rot \* r\_rot) \* exp\_scale)

k+=1

length = k

Y = np.array(Y)

X = np.array(X)

Ori = np.arctan2(Y,X)\*180/np.pi

Mag = (X \*\* 2 + Y \*\* 2) \*\* 0.5

W = np.exp(np.array(W))

for k in range(length):

rbin = RBin[k]

cbin = CBin[k]

obin = (Ori[k] - ori) \* bins\_per\_rad

mag = Mag[k] \* W[k]

r0 = int(rbin)

c0 = int(cbin)

o0 = int(obin)

rbin -= r0

cbin -= c0

obin -= o0

if o0 < 0:

o0 += n

if o0 >= n:

o0 -= n

# histogram update using tri-linear interpolation

v\_r1 = mag \* rbin

v\_r0 = mag - v\_r1

v\_rc11 = v\_r1 \* cbin

v\_rc10 = v\_r1 - v\_rc11

v\_rc01 = v\_r0 \* cbin

v\_rc00 = v\_r0 - v\_rc01

v\_rco111 = v\_rc11 \* obin

v\_rco110 = v\_rc11 - v\_rco111

v\_rco101 = v\_rc10 \* obin

v\_rco100 = v\_rc10 - v\_rco101

v\_rco011 = v\_rc01 \* obin

v\_rco010 = v\_rc01 - v\_rco011

v\_rco001 = v\_rc00 \* obin

v\_rco000 = v\_rc00 - v\_rco001

idx = ((r0 + 1) \* (d + 2) + c0 + 1) \* (n + 2) + o0

hist[idx] += v\_rco000

hist[idx+1] += v\_rco001

hist[idx + (n+2)] += v\_rco010

hist[idx + (n+3)] += v\_rco011

hist[idx+(d+2) \* (n+2)] += v\_rco100

hist[idx+(d+2) \* (n+2)+1] += v\_rco101

hist[idx+(d+3) \* (n+2)] += v\_rco110

hist[idx+(d+3) \* (n+2)+1] += v\_rco111

# finalize histogram, since the orientation histograms are circular

for i in range(d):

for j in range(d):

idx = ((i+1) \* (d+2) + (j+1)) \* (n+2)

hist[idx] += hist[idx+n]

hist[idx+1] += hist[idx+n+1]

for k in range(n):

dst.append(hist[idx+k])

# copy histogram to the descriptor,

# apply hysteresis thresholding

# and scale the result, so that it can be easily converted

# to byte array

nrm2 = 0

length = d \* d \* n

for k in range(length):

nrm2 += dst[k] \* dst[k]

thr = np.sqrt(nrm2) \* SIFT\_DESCR\_MAG\_THR

nrm2 = 0

for i in range(length):

val = min(dst[i], thr)

dst[i] = val

nrm2 += val \* val

nrm2 = SIFT\_INT\_DESCR\_FCTR / max(np.sqrt(nrm2), FLT\_EPSILON)

for k in range(length):

dst[k] = min(max(dst[k] \* nrm2,0),255)

return dst

def calcDescriptors(gpyr,keypoints,SIFT\_DESCR\_WIDTH = 4,SIFT\_DESCR\_HIST\_BINS = 8):

# SIFT\_DESCR\_WIDTH = 4，描述直方图的宽度

# SIFT\_DESCR\_HIST\_BINS = 8

d = SIFT\_DESCR\_WIDTH

n = SIFT\_DESCR\_HIST\_BINS

descriptors = []

for i in range(len(keypoints)):

kpt = keypoints[i]

o = kpt[2] & 255

s = (kpt[2] >> 8) & 255 # 该特征点所在的组序号和层序号

scale = 1.0 / (1 << o) # 缩放倍数

size = kpt[3] \* scale # 该特征点所在组的图像尺寸

ptf = [kpt[1] \* scale, kpt[0] \* scale] # 该特征点在金字塔组中的坐标

img = gpyr[o][s] # 该点所在的金字塔图像

descriptors.append(calcSIFTDescriptor(img, ptf, kpt[-1], size \* 0.5, d, n))

return descriptors

def SIFT(img,showDoGimgs = False):

SIFT\_SIGMA = 1.6

SIFT\_INIT\_SIGMA = 0.5 # 假设的摄像头的尺度

sigma0 = np.sqrt(SIFT\_SIGMA\*\*2-SIFT\_INIT\_SIGMA\*\*2)

n = 3

DoG,GuassianPyramid = getDoG(img, n,sigma0)

if showDoGimgs:

for i in DoG:

for j in i:

plt.imshow(j.astype(np.uint8), cmap='gray')

plt.axis('off')

plt.show()

KeyPoints = LocateKeyPoint(DoG, SIFT\_SIGMA, GuassianPyramid, n)

discriptors = calcDescriptors(GuassianPyramid,KeyPoints)

return KeyPoints,discriptors

def Lines(img,info,color = (255,0,0),err = 700):

if len(img.shape) == 2:

result = np.dstack((img,img,img))

else:

result = img

k = 0

for i in range(result.shape[0]):

for j in range(result.shape[1]):

temp = (info[:,1]-info[:,0])

A = (j - info[:,0])\*(info[:,3]-info[:,2])

B = (i - info[:,2])\*(info[:,1]-info[:,0])

temp[temp == 0] = 1e-9

t = (j-info[:,0])/temp

e = np.abs(A-B)

temp = e < err

if (temp\*(t >= 0)\*(t <= 1)).any():

result[i,j] = color

k+=1

print k

return result

def drawLines(X1,X2,Y1,Y2,dis,img,num = 10):

info = list(np.dstack((X1,X2,Y1,Y2,dis))[0])

info = sorted(info,key=lambda x:x[-1])

info = np.array(info)

info = info[:min(num,info.shape[0]),:]

img = Lines(img,info)

#plt.imsave('./sift/3.jpg', img)

if len(img.shape) == 2:

plt.imshow(img.astype(np.uint8),cmap='gray')

else:

plt.imshow(img.astype(np.uint8))

plt.axis('off')

#plt.plot([info[:,0], info[:,1]], [info[:,2], info[:,3]], 'c')

# fig = plt.gcf()

# fig.set\_size\_inches(int(img.shape[0]/100.0),int(img.shape[1]/100.0))

#plt.savefig('./sift/2.jpg')

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

origimg = plt.imread('./SIFTimg/1.jpeg')

if len(origimg.shape) == 3:

img = origimg.mean(axis=-1)

else:

img = origimg

keyPoints,discriptors = SIFT(img)

origimg2 = plt.imread('./SIFTimg/2.jpeg')

if len(origimg.shape) == 3:

img2 = origimg2.mean(axis=-1)

else:

img2 = origimg2

ScaleRatio = img.shape[0]\*1.0/img2.shape[0]

img2 = imresize(img2,(img.shape[0],int(round(ScaleRatio\*img2.shape[1]))))

keyPoints2, discriptors2 = SIFT(img2,True)

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(discriptors,[0]\*len(discriptors))

match = knn.kneighbors(discriptors2,n\_neighbors=1,return\_distance=True)

keyPoints = np.array(keyPoints)[:,:2]

keyPoints2 = np.array(keyPoints2)[:,:2]

keyPoints2[:, 1] = img.shape[1] + keyPoints2[:, 1]

origimg2 = imresize(origimg2,img2.shape)

result = np.hstack((origimg,origimg2))

keyPoints = keyPoints[match[1][:,0]]

X1 = keyPoints[:, 1]

X2 = keyPoints2[:, 1]

Y1 = keyPoints[:, 0]

Y2 = keyPoints2[:, 0]

drawLines(X1,X2,Y1,Y2,match[0][:,0],result)

**RFID READ代码**

!/usr/bin/env python

# -\*- coding: utf8 -\*-

#

# Copyright 2014,2018 Mario Gomez <mario.gomez@teubi.co>

#

# This file is part of MFRC522-Python

# MFRC522-Python is a simple Python implementation for

# the MFRC522 NFC Card Reader for the Raspberry Pi.

#

# MFRC522-Python is free software: you can redistribute it and/or modify

# it under the terms of the GNU Lesser General Public License as published by

# the Free Software Foundation, either version 3 of the License, or

# (at your option) any later version.

#

# MFRC522-Python is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU Lesser General Public License for more details.

#

# You should have received a copy of the GNU Lesser General Public License

# along with MFRC522-Python. If not, see <http://www.gnu.org/licenses/>.

#

import RPi.GPIO as GPIO

import MFRC522

import signal

continue\_reading = True

# Capture SIGINT for cleanup when the script is aborted

def end\_read(signal,frame):

global continue\_reading

print "Ctrl+C captured, ending read."

continue\_reading = False

GPIO.cleanup()

# Hook the SIGINT

signal.signal(signal.SIGINT, end\_read)

# Create an object of the class MFRC522

MIFAREReader = MFRC522.MFRC522()

# Welcome message

print "Welcome to the MFRC522 data read example"

print "Press Ctrl-C to stop."

# This loop keeps checking for chips. If one is near it will get the UID and authenticate

while continue\_reading:

# Scan for cards

(status,TagType) = MIFAREReader.MFRC522\_Request(MIFAREReader.PICC\_REQIDL)

# If a card is found

if status == MIFAREReader.MI\_OK:

print "Card detected"

# Get the UID of the card

(status,uid) = MIFAREReader.MFRC522\_Anticoll()

# If we have the UID, continue

if status == MIFAREReader.MI\_OK:

# Print UID

print "Card read UID: %s,%s,%s,%s" % (uid[0], uid[1], uid[2], uid[3])

# This is the default key for authentication

key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

# Select the scanned tag

MIFAREReader.MFRC522\_SelectTag(uid)

# Authenticate

status = MIFAREReader.MFRC522\_Auth(MIFAREReader.PICC\_AUTHENT1A, 8, key, uid)

# Check if authenticated

if status == MIFAREReader.MI\_OK:

MIFAREReader.MFRC522\_Read(8)

MIFAREReader.MFRC522\_StopCrypto1()

else:

print "Authentication error"