

实验心得

刘珏 2015201916

1 实验部分

1.1 第一次小车实验

1.1.1 实现功能

- AI Car 拼接组装完成
- 超声波传感器功能启用
- 蓝牙模块启用
- 使用蓝牙模块用手机和电脑操控小车
- 基于超声波传感器测距的随机游走

1.1.2 功能介绍

基础功能

功能	实现效果
拼接组装	小车配有左右两个带马达的前进轮与一个导向轮，供电采用四节5号电池，利用Arduino模块附有超声波模块与蓝牙模块
超声波传感器	利用声速与时间的关系，计算出与前方物体的距离
蓝牙模块	可以利用电脑或者Android手机对小车进行相应操控
随机游走	能在地上随机行走，碰到障碍物转弯或者掉头，保证全程不碰到障碍物

附加功能

功能	实现效果
自动走迷宫	利用障碍物摆一个迷宫，有一个入口和一个出口，只要道路够宽敞，小车可以在较短的时间内从一个口进入，另一个口出来，期间不碰到迷宫的边缘障碍物
识别运动物体	随机游走时，判断前方物体是障碍物还是运动物体，如果是障碍物就绕开随机走，如果是运动物体就加速冲上去直到撞击为止，之后再继续随机游走

1.1.3 实验心得

本次实验中我主要负责硬件的部分，由于是第一次接触 Arduino，我对 Arduino 板和 Arduino 的 IDE，以及其变成语言都没有了解，于是我在网上查阅论文相关资料，我们一起根据网上的示例在面包板上接线，连接红外装置，还下载了 Arduino 的 IDE，并学习了 Arduino 语言的基础语法。



尽管由于分工的不同我没有参与到程序的编写当中，我也了解了 Arduino 的原理并且知道了如何编写 Arduino 程序。这次的任务相对简单，但是为后续对小车功能的进一步完善打下了基础。

1.2 第二次小车实验

1.2.1 实现功能

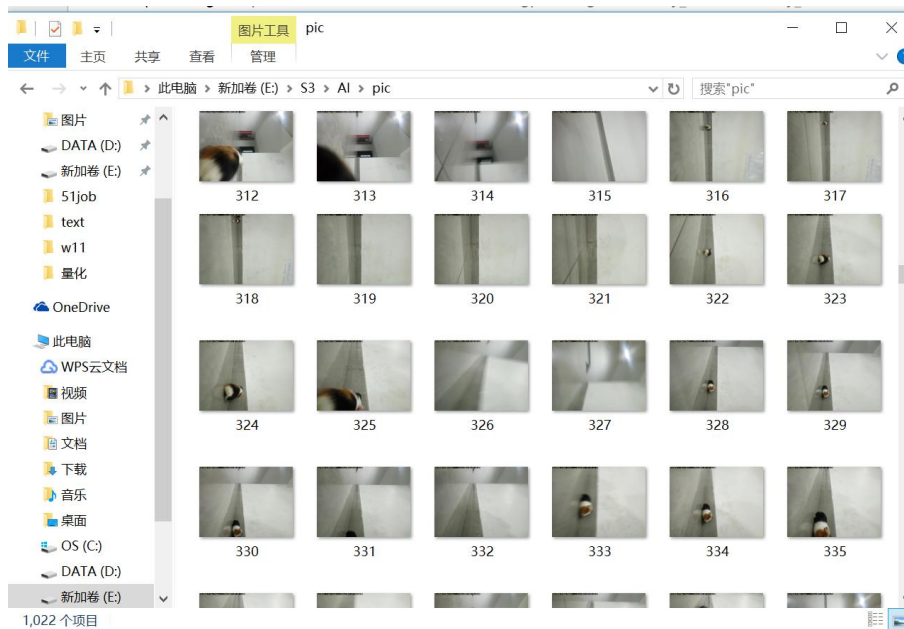
- 将手机摄像头捕捉到的影像实时传输到电脑
- 图像识别
- 目标跟踪
- AI Car 捕捉移动中小鼠
- 画出小鼠的运动轨迹

1.2.2 功能介绍

功能	实现效果
实时影像传输	使用DroidCam将手机摄像头画面传输到网页，再使用爬虫爬取画面储存到本地
蓝牙指令发送	使用python包pybluez操控电脑通过蓝牙发送指令
运动小鼠追踪	使用hog+svm图像识别算法识别出画面中的小鼠
运动轨迹标注	根据拍摄到的视频，自动判断出运动的物体并对其进行相应的像素标注与运动方向的展示

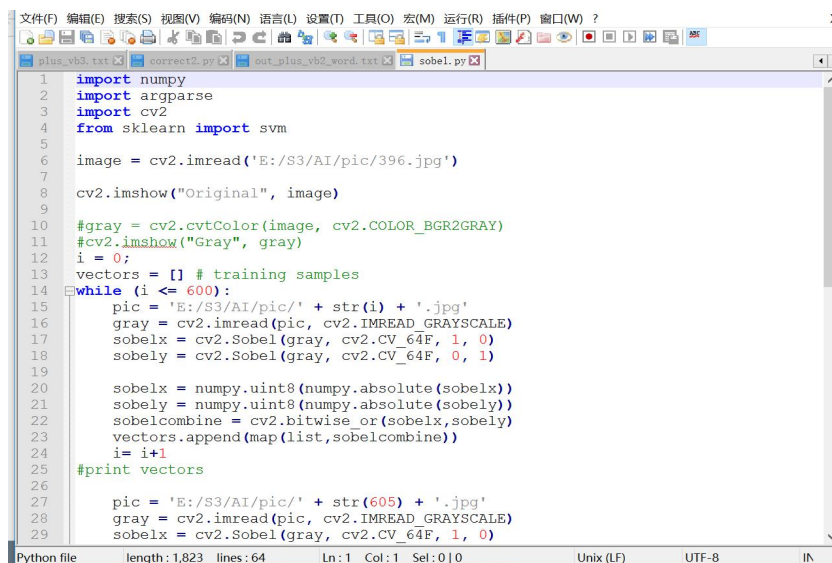
1.2.3 实验心得

这次实验中我主要负责的部分是图像的识别。实验需要识别 Droidcam 实时传输过来的图片中的小鼠，确定小鼠相对小车的位置，根据小鼠相对小车的位置利用蓝牙来发送命令给小车，从而实现小车追踪小鼠的功能。

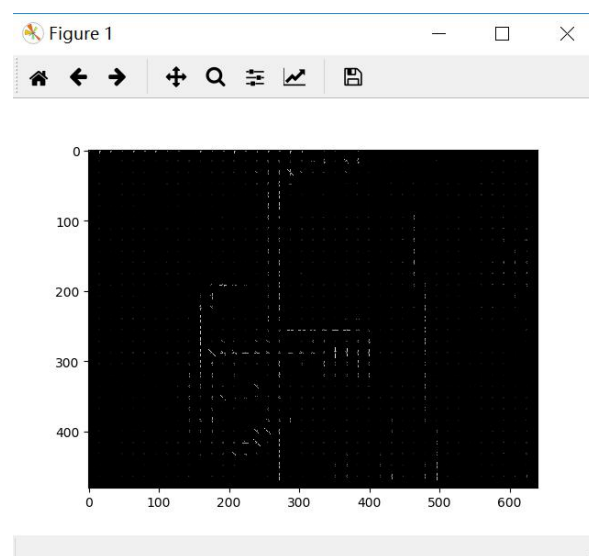


首先我们手动操纵小车前进左右转后退等来拍摄了许多小鼠的图片作为训练集，根据图片中小鼠的位置判断小车此时应该直走还是左转或者右转（小车不需要后退，当画面中没有小鼠时小车原地转圈直到画面中出现小鼠），手动给所有图片打上了标签，前进标为 1，左转为 3，右转为 4。

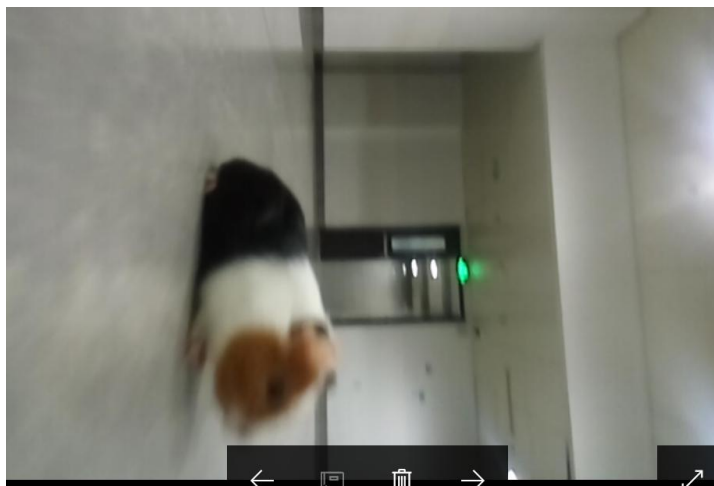
接下来我们使用了边缘检测算法来对图片进行处理。常见的边缘检测算法有 Canny 边缘检测算法、Sobel 算子边缘检测算法、Laplace 算子边缘检测算法，我们选择了 Sobel 算子算法进行试验，使用 opencv 并参考网上资料写出了代码，并使用了 svm 算法来进行训练，但是效果并不理想。



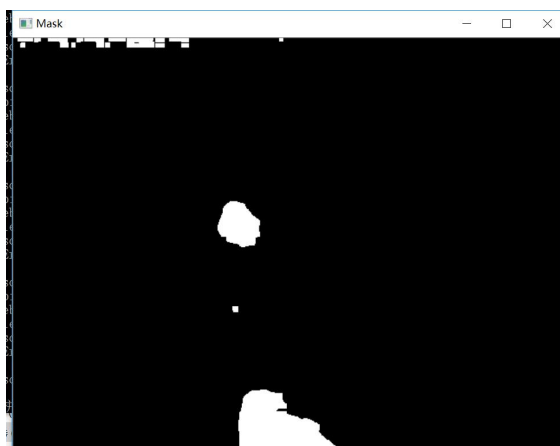
然后我们使用了 hog 特征检测的方法来处理图片，我们查阅网上的资料后写出了代码。



然后我们使用了 svm 算法进行训练，效果比使用 Sobel 算子的效果好。



为了寻找更好的方法，我们还使用了颜色识别，但由于小鼠身上颜色较多，且背景色与小鼠的颜色不够分明，效果并不理想。（下图中间白色部分为小鼠，下方白色部分为我们黑色的书包，因为书包颜色与小鼠相近所以也被标识出来了）



以下是 hog 算法的简介：

HOG算法的原理和具体实现步骤：

1. 灰度化（由于颜色在此算法中的作用不大，将图像转化为灰度图） $Y = 0.299R + 0.587G + 0.114B$
2. 采用Gamma校正法对输入图像进行颜色空间的标准化（归一化）目的是调节图像的对比度，降低图像局部的阴影和光照变化所造成的影响，同时可以抑制噪音的干扰 $I(x, y) = I(x, y)^\gamma$ 一般gamma取0.4或0.45较为符合视觉光照。
3. 计算图像每个像素的梯度（包括大小和方向）；主要是为了捕获轮廓信息，同时进一步弱化光照的干扰。

$$G_x(x, y) = H(x + 1, y) - H(x - 1, y)$$

$$G_y(x, y) = H(x, y + 1) - H(x, y - 1)$$

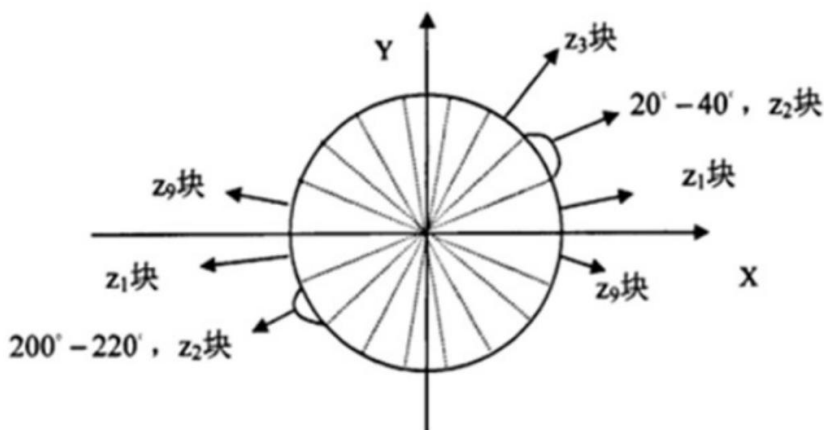
式中 $G_x(x, y)$, $G_y(x, y)$, $H(x, y)$ 分别表示输入图像中像素点 (x, y) 处的水平方向梯度、垂直方向梯度和像素点 (x, y) 处的梯度幅值和梯度方向分别为：

$$G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y(x, y)}{G_x(x, y)}\right)$$

```
height, width = img.shape
gradient_values_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
gradient_values_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
gradient_magnitude = cv2.addWeighted(gradient_values_x, 0.5, gradient_values_y, 0.5, 0)
gradient_angle = cv2.phase(gradient_values_x, gradient_values_y, angleInDegrees=True)
print gradient_magnitude.shape, gradient_angle.shape
```

4. 将图像划分成小cells（例如8*8像素/cell）
5. 加权投影法（weighted voting）每个cell的梯度直方图，即可形成每个cell的descriptor。具体方法：对每个cell的360度划分bin, 统计在每个梯度方向bin中的像素点个数进行投票，投票权重为改点的梯度幅值（也可使用与幅值相关的函数，但经试验效果没有简单幅值好）



具体操作中，往往使用三线性差值，即将当前像素的梯度方向大小、像素在cell中的x坐标与y坐标这三个值来作为插值权重，而被用来插入的值为像素的梯度幅值。采用三线性插值的好处在于：避免了梯度方向直方图在cell边界和梯度方向量化的bin边界处的突然变化。


```

cell_size = 8
bin_size = 8
angle_unit = 360 / bin_size
gradient_magnitude = abs(gradient_magnitude)
cell_gradient_vector = np.zeros((height / cell_size, width / cell_size, bin_size))

print cell_gradient_vector.shape

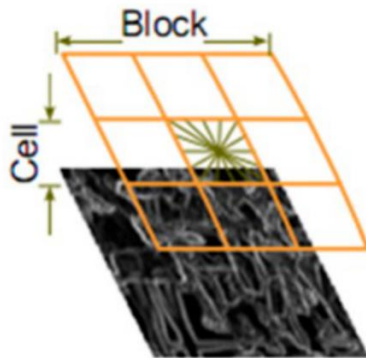
def cell_gradient(cell_magnitude, cell_angle):
    orientation_centers = [0] * bin_size
    for k in range(cell_magnitude.shape[0]):
        for l in range(cell_magnitude.shape[1]):
            gradient_strength = cell_magnitude[k][l]
            gradient_angle = cell_angle[k][l]
            min_angle = int(gradient_angle / angle_unit) % 8
            max_angle = (min_angle + 1) % bin_size
            mod = gradient_angle % angle_unit
            orientation_centers[min_angle] += (gradient_strength * (1 - (mod / angle_unit)))
            orientation_centers[max_angle] += (gradient_strength * (mod / angle_unit))
    return orientation_centers

for i in range(cell_gradient_vector.shape[0]):
    for j in range(cell_gradient_vector.shape[1]):
        cell_magnitude = gradient_magnitude[i * cell_size:(i + 1) * cell_size,
            j * cell_size:(j + 1) * cell_size]
        cell_angle = gradient_angle[i * cell_size:(i + 1) * cell_size,
            j * cell_size:(j + 1) * cell_size]
        print cell_angle.max()

        cell_gradient_vector[i][j] = cell_gradient(cell_magnitude, cell_angle)

```

6. 将每个cell组成一个block（例如3*3个cell/block），块内归一化梯度直方图。由于局部光照的变化以及前景-背景对比度的变化，使得梯度强度的变化范围非常大。这就需要对梯度强度做归一化。归一化能够进一步地对光照、阴影和边缘进行压缩。注意：block之间的是“共享”的，也即是说，一个cell会被多个block“共享”。另外，每个“cell”在被归一化时都是“block”independent，也就是说每个cell在其所属的block中都会被归一化一次，得到一个vector。这就意味着：每一个单元格的特征会以不同的结果多次出现在最后的特征向量中。这种混叠效应增强了块与块之间的联系，使得局部信息得到连接。一个block内所有cell的特征descriptor串联起来便得到该block的HOG特征descriptor。



```

import math
import matplotlib.pyplot as plt

hog_image= np.zeros([height, width])
cell_gradient = cell_gradient_vector
cell_width = cell_size / 2
max_mag = np.array(cell_gradient).max()
for x in range(cell_gradient.shape[0]):
    for y in range(cell_gradient.shape[1]):
        cell_grad = cell_gradient[x][y]
        cell_grad /= max_mag
        angle = 0
        angle_gap = angle_unit
        for magnitude in cell_grad:
            angle_radian = math.radians(angle)
            x1 = int(x * cell_size + magnitude * cell_width * math.cos(angle_radian))
            y1 = int(y * cell_size + magnitude * cell_width * math.sin(angle_radian))
            x2 = int(x * cell_size - magnitude * cell_width * math.cos(angle_radian))
            y2 = int(y * cell_size - magnitude * cell_width * math.sin(angle_radian))
            cv2.line(hog_image, (y1, x1), (y2, x2), int(255 * math.sqrt(magnitude)))
            angle += angle_gap

plt.imshow(hog_image, cmap=plt.cm.gray)
plt.show()

```

7. 将图像image内的所有block的HOG特征descriptor串联起来就可以得到该image (你要检测的目标) 的HOG特征descriptor了。这个就是最终的可供分类使用的特征向量了。

2 Mnist 数据集部分

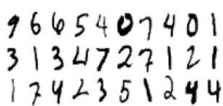
2.1 简述

本次实验的数据集是 keras 包中下载的 mnist 数据集，加上从 kaggle 平台上下载的 42000 条训练数据和 28000 条测试数据，使用的机器学习算法有决策树、随机森林、SVM、神经网络。

2.2 算法与准确率

2.2.1 决策树

在本次实验中，我首先使用了较为简单的决策树算法，在程序实践的过程中，我使用了 sklearn，程序较为简单，且效果较好。使用了 sklearn 包中决策树的默认参数，使用了全部 42000 个训练数据作为训练集，28000 个测试数据作为测试集，将结果在 kaggle 平台上提交后得到的准确率为 85%。



Digit Recognizer

Learn computer vision fundamentals with the famous MNIST data

2,061 teams · 2 years to go

[Overview](#)
[Data](#)
[Kernels](#)
[Discussion](#)
[Leaderboard](#)
[Rules](#)
[Team](#)
[My Submissions](#)
[Submit Predictions](#)

2.2.2 随机森林

使用了 sklearn 包中的随机森林算法。由于需要调参，使用了 42000 条训练数据中的 4000 条作为验证集，剩余作为训练集，得到参数 max_depth=2, random_state=1, 然后使用测试集中 28000 条数据作为测试集，将结果提交到 kaggle 平台上得到准确率为 54%，效果不佳。

2.2.3 SVM+PCA

接下来使用了 sklearn 包中的 SVM 算法，由于 SVM 算法计算的速度较慢，于是使用了 PCA 方法给数据降维，在保留了数据的信息量的前提下加快计算的速度。同时也是先使用了训练集中的 4000 条数据作为验证集来调整 SVM 的参数，最后发现使用 SVM 的默认参数准确度最高，然后使用全体 42000 条训练数据来训练模型，全体 28000 条测试数据作为测试集，将结果提交到 kaggle 平台上得到的准确率为 98%，效果很不错。

2.2.4 神经网络

使用了 keras 包中的神经网络框架来搭建神经网络，以下尝试了两种不同的搭建方法，由于神经网络本身效果比 svm 好，加之使用的训练数据是从 keras.datasets 中下载的，训练数据有 60000 条（使用原来的 42000 条训练数据的准确率也达到了 99% 以上），因此在 kaggle 平台上提交后准确率都达到了 99% 以上。代码一是算法一神经网络的搭建部分，代码二是算法二完整的代码，其中代码二的准确率是本实验中我使用的所有算法里最高的，达到了 99.6%。



代码一：

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

代码二：

```
import datetime
import keras
import csv
import numpy as np
from keras.datasets import mnist
```



```

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

now = datetime.datetime.now

batch_size = 128
num_classes = 10
epochs = 27

# input image dimensions
img_rows, img_cols = 28, 28
# number of convolutional filters to use
filters = 32
# size of pooling area for max pooling
pool_size = 2
# convolution kernel size
kernel_size = 3

if K.image_data_format() == 'channels_first':
    input_shape = (1, img_rows, img_cols)
else:
    input_shape = (img_rows, img_cols, 1)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# define two groups of layers: feature (convolutions) and classification (dense)
feature_layers = [
    Conv2D(filters, kernel_size,
           padding='valid',
           input_shape=input_shape),
    Activation('relu'),
    Conv2D(filters, kernel_size),
    Activation('relu'),
    MaxPooling2D(pool_size=pool_size),
    Dropout(0.25),
    Flatten(),
]

classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(0.5),

```

```

        Dense(num_classes),
        Activation('softmax')
    ]

# create complete model
model = Sequential(feature_layers + classification_layers)

x_train = x_train.reshape((x_train.shape[0],) + input_shape)
x_test = x_test.reshape((x_test.shape[0],) + input_shape)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

t = now()
model.fit(x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_data=(x_test, y_test))
print('Training time: %s' % (now() - t))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

csvFile2 = open("E:/S3/kaggle/test.csv", "r")
reader2 = csv.reader(csvFile2)

out = open('E:/S3/kaggle/result_ke52.csv', 'w', newline='')
csv_write = csv.writer(out, dialect='excel')
csv_write.writerow(['ImageId', 'Label'])
data = []
label = []

```

```
test = []
count = 1

for i in reader2:
    if reader2.line_num == 1:
        continue
    i = np.array(i)
    i = i.reshape(1, 28, 28, 1)
    i = i.astype('float32')
    i /= 255
    temp = str(int(model.predict_classes( i, batch_size=128)))
    #out2.write(str(temp)+'\n')
    csv_write.writerow([count, temp])
    count = count+1
    print(temp)
```