高度情報処理演習

担当: 井尻 敬

課題 『**高速度カメラの新しい使い方を提案せよ**』

民生用高速度カメラが普及し始めてきた．そこで，1) 高速度カメラを活用できるシーンを**独自に提案**し，2)実際に現場にて高速度カメラ撮影を行い，3)その動画像処理を行うプログラムを作成せよ．

達成目標

- 普及の期待できる高速度カメラの独創性のある利用法を提案できる
- 提案した内容（一部でもよい）をプログラムを用いて実装できる
- 高速度カメラ計測を経験し，その特性や限界を議論できる

課題 『**高速度カメラの新しい使い方を提案せよ**』

民生用高速度カメラが普及し始めてきた．そこで，1) 高速度カメラを活用できるシーンを**独自に提案**し，2)実際に現場にて高速度カメラ撮影を行い，3)その動画像処理を行うプログラムを作成せよ．

- 形式　　：3名程度のグループワーク
- 仕様言語：Python
- カメラ　：スマホを利用（または研究室備品を貸し出し）
- 撮影対象：各グループが自由に決めてよい
  - **他のグループと異なるもの**
  - 民生用高速度カメラの活用といえるもの
  - 安全に撮影が可能（撮影者・撮影機材）
- 評価：最終日のプレゼンテーション

撮影対象の選定とその宣言法

- 履修者が確定後…
  - slackに本演習用のワークスペースを作成し履修者全員を登録する
  - グループを作成しslackにて提示する

  TODO　al*@shibaura-it.ac.jp へ届くメールをよく確認すること

  TODO　slackの使い方を調べておくこと
- グループ作成後
  - Slackにて各グループ用のchannelをこちらで作成

  TODO グループ用channelにて議論し撮影対象を確定すること

  TODO general channelにて撮影対象の宣言すること

  → 井尻とTAが安全性について議論しOKが出れば撮影対象の宣言完了

  →先に宣言したグループを優先し，既存の撮影対象は選択不可

  → ひとつのグループが2件以上の撮影対象を宣言することは不可

解析用サンプルプログラム

- サンプルコードを配布 & 解説する
  - 全手動トラッキング
  - 色追跡
  - テンプレートマッチング
  - オプティカルフロー

- このコードを計測対象に合わせてチューニングするのが近道だと思います
- 全手動トラッキングツールを改良した『使いやすい全手動トラッキングツール』なども良い評価が付くと思われます

スケジュール

- 事前
  - グループ分け
  - 撮影対象決めとその宣言（slackを利用）
- 1週目
  - Pythonを利用した高速度画像処理演習
  - 撮影練習
  - 計測と開発
- 2周目
  - 計測と開発
- 3週目
  - 計測と開発
- 4週目
  - プレゼンテーション

プレゼンテーション

- 時間: 各グループ7分程度 + 質疑
- 必要な内容
  - 計測対象と高速度カメラが必要な理由
  - 解析アルゴリズムの詳細
  - 解析結果や得られた知見
  - 有用性やサービス展開について（あれば）

本科目の評価基準

評価項目	A	B	C
観察対象の設定	高速度カメラの特性を考慮して撮影・計測対象が選択され，その撮影対象を選択した理由が明確に述べられている	撮影・計測対象が明示されているが，その撮影対象を選択した理由があいまいである	撮影・計測対象が明示されておらず，その対象を選択した理由も説明されていない
撮影における工夫	動画像処理の工程を簡便にするために撮影時に工夫がなされており，その理由や効果が説明されている	撮影時に工夫がなされているが，その理由や効果の説明があいまいである	動画像処理の工程を簡便にするために撮影時に工夫がなされていない
動画像処理における工夫	動画像処理アルゴリズムにおいて，演習中に解説した手法以上の工夫がなされている.	動画像処理アルゴリズムにおいて，演習中に開設した手法が効果的に利用できている.	動画像処理が，すべて手作業で行われている. ※効率化のための手作業の導入は減点対象ではない.
プレゼンテーション	動画・図・グラフを多用し，わかりやすいプレゼンテーションが設計されている	動画・図・グラフを利用してプレゼンテーションが設計されている	文字のみで見やすさや分かりやすさへの配慮が行われていない.
質疑	質疑に積極的に参加した場合，加点する。質問1回1点.		

※各グループの独自性を特に重視しています．他グループと同じ計測対象を選択した場合は，先に計測対象を宣言したグループを優先し，２件目以降は点数を一律0.5倍します.
※時間の限られた演習なのですべてを完璧に自動処理しなくとも減点対象にはしません．難しい部分には手作業を効果的に導入してください.

自分のPCでpythonを動かしたい人向けメモ

Anacondaをインストールする

- 本家ページより，インストーラーをダウンロード
  - https://www.continuum.io/downloads
  - 今回はPython3, 64bit installerを選択
  - ファイルサイズが大きいので多少時間がかかる

  ※2017/3/9現在，Anaconda4.3.0が最新だが，このバージョンはOpenCVがうまくインストールできない

  ※ https://repo.continuum.io/archive/index.html ←こちらから「 Anaconda3-4.2.0-Windows-x86_64.exe 」を利用するとうまくいく．

- 『Anaconda3-4.2.0-Windows-x86_64.exe』を起動しインストール
  - コマンドプロンプトで > python –versionとして以下の感じなら成功

```
C:¥Users¥takashi>python --version
Python 3.5.2 :: Anaconda 4.2.0 (64-bit)

C:¥Users¥takashi>
```

OpenCVをインストール

1. コマンドプロンプトを右クリックして管理者権限で起動
2. > conda install –c menpo opencv3
3. 途中でyキーを押す
4. 5分くらい待つ
5. > python sample.py

```
1 sample.py
2 img = cv2.imread('sample.jpg')
3 print( img.shape )
4 cv2.imshow('sample', img)
5 cv2.waitKey(0)
```

引くほど簡単にインストールできた
本当はもっと大変な思いをすると思ったからポイントをメモしておくためにこの文章を書き始めたのに。。。

Pythonによる動画像処理

pythonの基本はデジタルメディア処理２の講義資料か入門書参照

画像の入出力：ex1.py

実習: 適当な画像をコードと同一フォルダに配置し実行

```
$ python  ex1.py  fname.png
```

※**cv2.imread**(“fname”)で画像読み込み
※**cv2.imshow**(“caption”, img)で画像表示
※**cv2.imwrite**(“fname”, img)で画像書き出し
※ sys.argv[1] はコマンドライン入力のfname.png

※画像は np.array 形式で表現される
img.shape：画像サイズ　　(通常 高x幅x3)
img.dtype：画像データの型 (通常 np.uint8 型)

```
# -*- coding: utf-8 -*-
# ex1.py
import numpy as np
import cv2
import sys

#load image
img = cv2.imread( sys.argv[1] )
print( img.shape, type(img), img.dtype )

#display img
cv2.imshow("show image", img)
cv2.waitKey()

#save image
cv2.imwrite("img_save.png", img);
```

画像に図形描画：ex2.py

- **コードを実行し画像に図形描画を確認する**

```
$ python  ex2.py  fname.png
```

※手軽に画像への書き込みが行なえます
cv2.line　　(画像, 点1, 点2, 色, 太さ)
cv2.rectangle(画像, 点1, 点2, 色, 太さ)
cv2.circle　　(画像, 中心, 半径, 色, 太さ)

※その他の図形描画関数は以下を参照
http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

```
# -*- coding: utf-8 -*-
# ex2.py
import numpy as np
import cv2
import sys

#load image
img = cv2.imread( sys.argv[1] )
print( img.shape, type(img), img.dtype )

#draw rect and dots
cv2.line      (img,(100,100),(300,200), (0,255,255),2)
cv2.circle    (img,(100,100), 50,       (255,255,0),1)
cv2.rectangle(img,(100,100),(200,200), (255,0,255),1)

#display img
cv2.imshow("show image", img)
cv2.waitKey()
```

画素へのアクセス ex3.py

実習：コードを実行し出力を確認する

```
$ python  ex3.py  fname.png
```

- 画像の(y,x)画素には以下の通りアクセスできる

  r = img[y,x,2]

  g = img[y,x,1]

  b = img[y,x,0]

実習：コードを編集し画像の赤と青チャンネルを入れ替えよ

```python
# -*- coding: utf-8 -*-
# ex3.py
import numpy as np
import cv2
import sys

img = cv2.imread( sys.argv[1] )
H   = img.shape[0]
W   = img.shape[1]

for y in range(H) :
    for x in range(W) :
        img[y,x,2] = 128 #ここを編集

cv2.imshow("image", np.uint8( img) )
cv2.waitKey()
```

動画像へ

よく利用する関数

```
#動画ファイルを開きフレームを取得
#readを繰り返し呼ぶと次のフレームを取得できる
#読み込みに失敗するとret == Falseに
cap         = cv2.VideoCapture( fname )
ret, frame = cap.read()

#動画内のフレーム数取得(元はfloatなのでintキャスト)
frame_num = int( cap.get(cv2.CAP_PROP_FRAME_COUNT) )

#kフレーム目に移動（好きなフレームへ移動できる）
cap.set(cv2.CAP_PROP_POS_FRAMES, k)

# キーボードの入力待ち
# 引数は待ち時間，0なら入力があるまで待機
key = cv2.waitKey( 0 )
```

動画のロードと表示 ex4.py

```
$ python  ex4.py
```

※qキーで終了
※右キーで1フレーム進む
※左キーで1フレーム戻る

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np

CV_WAITKEY_CURSORKEY_RIGHT  = 2555904; #右カーソルID
CV_WAITKEY_CURSORKEY_LEFT   = 2424832; #左カーソルID
```

```
if __name__ == '__main__':
    # read video
    cap   = cv2.VideoCapture("video.mp4" )
    ret, frame = cap.read()

    frame_num  = int( cap.get(cv2.CAP_PROP_FRAME_COUNT) )
    frame_H    = frame.shape[0]
    frame_W    = frame.shape[1]

    frame_half = cv2.resize(frame, (frame_W // 2, frame_H // 2) )
    cv2.imshow("video vis", frame_half)
    frame_I    = 0

    while (True) :
        key = cv2.waitKey( 0 )

        if(   key == ord('q') ) :
            exit()
        elif( key == CV_WAITKEY_CURSORKEY_RIGHT ) :
            frame_I = min(frame_I+1, frame_num-1)
        elif( key == CV_WAITKEY_CURSORKEY_LEFT  ) :
            frame_I = max(frame_I-1, 0)

        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_I)
        ret, frame = cap.read()
        frame_half = cv2.resize(frame, (frame_W // 2, frame_H // 2))
        cv2.imshow("video vis", frame_half)

        print("current frame i = ", frame_I)
```

動画の書き出し ex5.py

```
$ python  ex5.py
```

※sキーで動画をセーブ（円を上から描画）

```
#画像書き出し
# “fname.mp4” :動画ファイル名
# fourcc は圧縮方法の指定
#res：結果画像
fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
out    = cv2.VideoWriter(  'fname.mp4' , fourcc, 20.0,
                                    (frame_W, frame_H))

#動画へフレームを書き出す（複数回呼ぶ）
out.write( frame )

#動画書き出しを終了
out.release()
```

```
# while文に以下の行を追加
# “s”キーを押すと、動画をそのままセーブ
elif( key == ord('s') ) :
    save_video( frame_W, frame_H, cap )
```

```
#以下の関数を追加
def save_video( frame_W, frame_H, cap ) :
    fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
    out    = cv2.VideoWriter(  'video_out.mp4' , fourcc, 20.0,
                                            (frame_W, frame_H))

    cap.set( cv2.CAP_PROP_POS_FRAMES, 0 )
    count = 0
    while( True ) :
        ret, frame = cap.read()
        if ret == False :
            break
        count += 2
        cv2.circle(img,(100+count,100), 50,(255,255,0),1)
        out.write( frame )

    out.release()
```

全手動トラッキング ex6.py

```
$ python  ex6.py
```

※qキーで終了
※右キーで1フレーム進む
※左キーで1フレーム戻る
※a キーでマーカーを左に移動
※wキーでマーカーを上に移動
※s キーでマーカーを下に移動
※d キーでマーカーを右に移動
※x キーで全フレームのマーカー位置を出力

大変だけど、根性があれば結果が出せるツールではある、が。。。

```
                :(省略)

#マーカー位置を格納する配列を初期化
   markers = np.zeros( (frame_num,2) )
   for i in range(frame_num) :
       markers[i][0] = frame_W // 2
       markers[i][1] = frame_H // 2

                :(省略)

   while (True) :
       key = cv2.waitKey( 0 )
       if(   key == ord('q') ) :
           exit()
       elif( key == CV_WAITKEY_CURSORKEY_RIGHT ) :
           frame_I = min(frame_I+1, frame_num-1)
       elif( key == CV_WAITKEY_CURSORKEY_LEFT  ) :
           frame_I = max(frame_I-1, 0)
       elif( key == ord('a') ) :
           markers[frame_I][0] -= 5
       elif( key == ord('w') ) :
           markers[frame_I][1] += 5
       elif( key == ord('s') ) :
           markers[frame_I][1] -= 5
       elif( key == ord('d') ) :
           markers[frame_I][0] += 5
       elif( key == ord('x') ) :
           for m in  markers :
               print(m)

                :(省略)
```

全手動トラッキング ex6.py

```
$ python  ex6.py
```

※qキーで終了
※右キーで1フレーム進む
※左キーで1フレーム戻る
※a キーでマーカーを左に移動
※wキーでマーカーを上に移動
※s キーでマーカーを下に移動
※d キーでマーカーを右に移動
※x キーで全フレームのマーカー位置を出力

大変だけど、根性があれば結果が出せるツールではある、が。。。

```
                :(省略)

#マーカー位置を格納する配列を初期化
   markers = np.zeros( (frame_num,2) )
   for i in range(frame_num) :
       markers[i][0] = frame_W // 2
       markers[i][1] = frame_H // 2

                :(省略)

   while (True) :
       key = cv2.waitKey( 0 )
       if(   key == ord('q') ) :
           exit()
       elif( key == CV_WAITKEY_CURSORKEY_RIGHT ) :
           frame_I = min(frame_I+1, frame_num-1)
       elif( key == CV_WAITKEY_CURSORKEY_LEFT  ) :
           frame_I = max(frame_I-1, 0)
       elif( key == ord('a') ) :
           markers[frame_I][0] -= 5
       elif( key == ord('w') ) :
           markers[frame_I][1] += 5
       elif( key == ord('s') ) :
           markers[frame_I][1] -= 5
       elif( key == ord('d') ) :
           markers[frame_I][0] += 5
       elif( key == ord('x') ) :
           for m in  markers :
               print(m)

                :(省略)
```

色を用いた二値化 ex7.py

```
$ python  ex6.py
```

※cキーで色による二値化
- 画像をHSVカラーに変換し、
- 閾値処理により二値化,
- erode/dilateによりノイズ除去
- 前景画素の重心を計算

```
#色による二値化
# img : 入力カラー画像
# c_min/c_max : カラーの閾値
# out : 二値化済み画像
out = cv2.inRange(img, c_min, c_max)

# erode/dilate operations
# img : 入力画像
# kernel : カーネル
out = cv2.erode (img, kernel,iter = 1)
out = cv2.dilate(img, kernel,iter = 1)
```

```
def color_threshold(frame_H, frame_W, cap ) :

    cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

    while (1) :
        ret, frame = cap.read()
        if ret == False :
            break

        # color threshold 水色である程度明るい色を
        c_min = np.array([70 , 100,  30], np.uint8)
        c_max = np.array([120, 255, 200], np.uint8)

        frame_hsv  = cv2.cvtColor( frame, cv2.COLOR_BGR2HSV)
        binary_img = cv2.inRange(frame_hsv, c_min, c_max          )
        binary_img = cv2.resize(binary_img, (frame_W // 2, frame_H // 2))

        #openningによるノイズ除去
        kernel = np.ones((3,3),np.uint8)
        binary_img = cv2.erode(binary_img, kernel,iterations = 1)
        binary_img = cv2.dilate(binary_img, kernel,iterations = 1)

        #重心の取得（ここは遅い，もっと高速な方法はある）
        X = Y = S = 0
        for y in range( binary_img.shape[0]) :
            for x in range( binary_img.shape[1] ) :
                if( binary_img[y,x] == 255) :
                    X += x
                    Y += y
                    S += 1
        X /= S
        Y /= S
             :  省略
```

テンプレートマッチング ex8.py

```
$ python  ex8.py
```

t キーでテンプレートマッチングを実行し結果を動画として書き出す

```
#テンプレートマッチング
#img:ターゲット画像
#tmp: テンプレート画像
#flg ： 計算法
#res：結果画像
res = cv2.matchTemplate(img, tmp, flg)

#画像内の最大・最小値検索
#minV/maxV：画像内の最小・最大値
#minL/maxL：画像内の最小位置・最大位置
minV, maxV, minL, maxL = cv2.minMaxLoc(res)
```

```
def template_matching( frame_W, frame_H, cap ) :

    template = cv2.imread("temp.png")
    h = template.shape[0]
    w = template.shape[1]
    fourcc  = cv2.VideoWriter_fourcc('m','p','4','v')
    out     = cv2.VideoWriter(  'video_out.mp4'  , fourcc, 20.0,
                                          (frame_W, frame_H) )
    count = 0
    cap.set( cv2.CAP_PROP_POS_FRAMES, 0 )
    while( True ) :
        ret, frame = cap.read()
        if ret == False :
            break

        # Apply template Matching, get min/max location
        res = cv2.matchTemplate(frame, template, cv2.TM_SQDIFF)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

        cv2.rectangle(frame, min_loc,
                (min_loc[0] + w, min_loc[1] + h), (255,0,0), 3)
        out.write( frame )
        print(min_loc[0] + h//2, min_loc[1] + w//2, )

        count += 1
        if( count > 100) :
            break #時間がかかるので途中で止める
    out.release()
```

オプティカルフロー ex9.py

```
$ python  ex9.py
```

※ oキーでオプティカルフロー計算を実行
※ opticalflow_corners関数を参照
※ コーナー画素を発見し，その画像の移動先を検索していく

```
#コーナー位置の検索
# feature_params：コーナー検索のパラメータ
# img_gry：対象画像（グレースケール）
# pixels：発見したコーナー画素位置 （np.array型）
# 詳細は http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html
feature_params = dict( maxCorners = 100, qualityLevel = 0.3, minDistance = 7, blockSize = 7 )
pixels = cv2.goodFeaturesToTrack( img_gry, mask = None, **feature_params)

#optical flow検出
# lk_params：optical flow検出のパラメータ
# img_gry1, img_gry2: 連続する2枚の画像．この2枚の画像から画素の移動を計算する
# pixels：移動を計算したい画素の配列（np.array型）
# pixels1：pixelsの移動後の位置を表す配列（np.array型）
# 詳細は http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html
lk_params = dict( winSize=(15,15), maxLevel=2, criteria =
                                  (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
pixels1, st, err = cv2.calcOpticalFlowPyrLK(img_gry1, img_gry2, pixels, None, **lk_params)
```