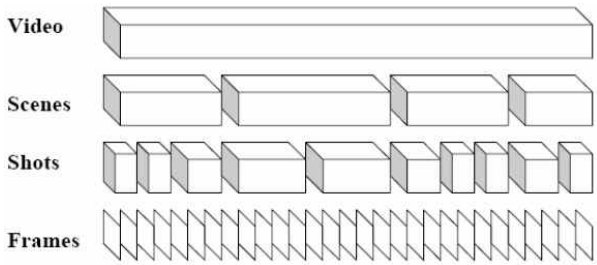
C-1. 동영상 처리

1

디지털 비디오(Digital video)

- 디지털 비디오는 일련의 디지털 영상을 보여주는 것으로 구성됨

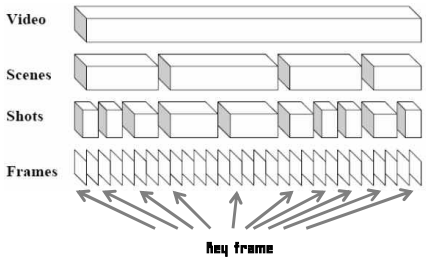


  - Frame : 한 이미지. 비디오의 최소 단위
  - Shot : 비슷한 내용의 연속된 프레임 그룹 (한 shot은 비슷한 내용으로 구성)
  - Scene : 여러 shot들의 집합
  - Video : 여러 scene들의 집합
  - Scene 변경(change) : 비디오의 내용이 바뀌고 새로운 내용이 시작함

2

Video Browsing

- Video Browsing / Summarization
  - 비디오의 내용을 대표하는 여러 이미지(프레임)로 요약
  - 비디오의 모든 프레임을 분석하기는 어려움
    => 매우 많은 계산량을 요구함
  - 대부분의 비디오는 비슷한 프레임을 가짐
    => *Shot의 프레임들을 대표하는 주요 프레임(keyframe)*만 선택하여 분석하는 것이 효과적임



3

Keyframe 선택(Selection)

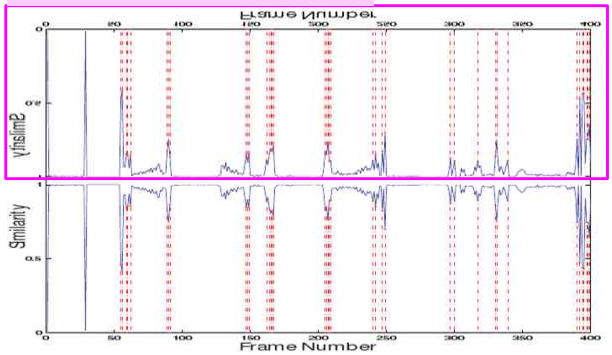
- **Keyframe**
  - A typical frame of a shot
  - Represent a salient content feature
  - Shot 당 하나 이상의 keyframe을 가질 수 있음
  - 비디오 색인, 검색 및 검색에 적합한 추상화 제공
  - 비디오 컨텐츠 분석을 위한 조직된 프레임워크 제공
  - 이미지 콘텐츠를 제공하고 검색 및 일치의 계산 복잡성을 줄일 수 있음.

4

Keyframe 선택(Selection)

- **Keyframe**
  - 접근법 : 프레임 간의 내용 유사성(similarity) 분석

Dissimilarity = 1 - Similarity
≈ Difference



5

차영상(Difference image)

- 정지된 카메라 위치 및 일정한 조명을 가정하면서
  다른 순간에 획득된 두개의 이미지를 간단하게 빼서 동작 감지

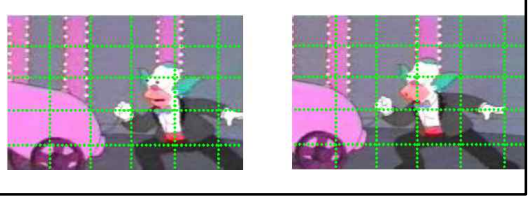
$$| f_t(x,y) - f_{t+1}(x,y) |$$

  - 차영상은 *이진 영상* → 두 개의 연속된 영상사이의 뺄셈

(a) (b) (c)

(a) Image x_1 at time instance t, (b) image x_2 at time instance t+1, and (c) difference image Δx.

- **문제점**
  - 물체 또는 카메라 움직임에 민감



6

차영상

cv2.absdiff(img1,img2)

$$|f_t(x,y) - f_{t+1}(x,y)|$$

```
def frame_diff(prev_frame, cur_frame):
     return cv2.absdiff(next_frame, cur_frame)

def get_frame(cap):
    ret, frame = cap.read()
    if ret :
        return cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    else :
        return []

cap = cv2.VideoCapture(0)

prev_frame = get_frame(cap)
cur_frame = get_frame(cap)

while True:
   diff = frame_diff(prev_frame, cur_frame)
   cv2.imshow("Motion", diff)

   prev_frame = cur_frame
   cur_frame = get_frame(cap)
   if cur_frame == [] :
        break
```

7

배경 제거

```
#파이썬으로 만드는 OpenCV 프로젝트

cap = cv2.VideoCapture('../img/walking.avi')
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)

# 배경 제거 객체 생성 --- ①
fgbg = cv2.bgsegm.createBackgroundSubtractorMOG()
#fgbg = cv2.createBackgroundSubtractorMOG2()

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # 배경 제거 마스크 계산 --- ②
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',frame)
    cv2.imshow('bgsub',fgmask)
    if cv2.waitKey(delay) & 0xff == 27:
       break
```

8

차영상을 이용한 Keyframe Selection

- 차이의 합계가 임계값을 초과하는 프레임을 키 프레임으로 선택

$$D(t) = \sum_{x,y} | f_t(x,y) - f_{t+1}(x,y) |$$

  - 임계값에 따른 결과
    - 임계 값이 너무 작으면 많은 키 프레임이 추출됩니다.
    - 임계 값이 너무 크면 일부 키 프레임이 누락될 수 있습니다.

9

차영상을 이용한 Keyframe Selection

```
def frame_diff(prev_frame, cur_frame):
    return cv2.absdiff(cur_frame, prev_frame)

cap = cv2.VideoCapture(0)
prev_frame = get_frame(cap)
cur_frame = get_frame(cap)

i = 0
while True:
   diff = frame_diff(prev_frame, cur_frame)
   if np.sum(diff) > 1000000 :
            cv2.imshow("Keyframe" + str(i), cur_frame)

   prev_frame = cur_frame
   cur_frame = get_frame(cap)
   if curr_frame == [] :
         break

   key = cv2.waitKey(10)
   if key == 27:
         break
   i = i + 1
```

np.sum(배열)은 배열 내 모든 배열요소의 합을 구하는 Numpy 함수

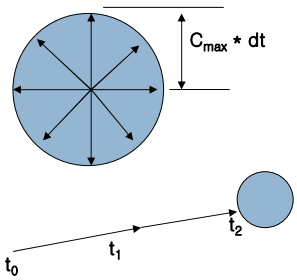
10

C-2. 움직임 추적

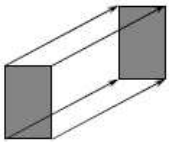
11

움직임 가정(assumptions)

- 최대 속도 = 최대 거리/1frame
  - 너무 많이 움직이지 않는다고 가정
  - 가정하지 않으면?
    물체가 빠르게(많이) 움직인 것인지,
    새로운 물체의 등장인지 혼동될 수 있음
  - 최대 거리를 벗어나면
    새로운 물체의 등장으로 간주
- 작은 가속도
  - 비슷한 속도를 유지한다고 가정
- 한 물체의 점들은 공통된 움직임
  - 주변의 점들이 같은 움직임을
    가진다고 가정
- 상호 대응
  - 대응되는 점들이 같은 움직임을
    가진다고 가정



Common Motion



12

움직임 계산

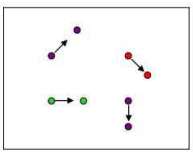
13

Optical Flow

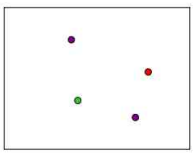
- ***Optical flow*** : 시간 간격 dt 동안의 움직임으로 인한 이미지 변화를 반영
  1. 주어진 프레임에서 일련의 특징 점들을 검출
  2. 연속 프레임들 사이의 모션 벡터로서 그것을 추적하도록 변위(displacement) 벡터를 계산

  - **벡터의 길이(Length, magnitude)**는 속도의 크기를 결정.
  - **벡터의 방향(Direction)**는 움직임의 방향을 결정.

- ***Optical flow field*** : 2D 이미지에서 물체 점들의 3D 움직임을 나타내는 속도 필드



$I(x,y,t-1)$



$I(x,y,t)$

14

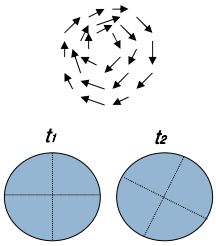
Optical Flow

- 조명 변화 및 중요하지 않은 물체 (예 : 그림자)의 움직임에 민감하지 않아야 합니다.

- **가정:**
  1) 물체 점의 밝기(색상)는 시간이 지나도 일정
  2) 이미지 면에 가까운 점들은 비슷한 방식으로 움직임(속도 변화 제한).

- **예외:**
  ① **Non-zero optical flow**
  → 고정된 구가 움직이는 조명에 의해 밝기가 변함

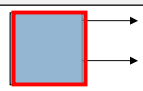
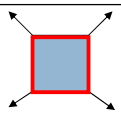
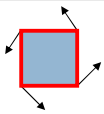
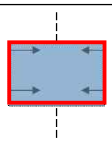
  ② **Zero optical flow**
  → 단일 색상의 구가 일정한 조명 아래 제자리에서 회전함



15

Optical Flow

- 움직임의 형태

움직임	벡터	
관측자로부터 일정한 거리(깊이)에서의 이동 (xy 평면에서의 이동)	병렬(평행) 움직임 벡터 집합	
관찰자를 기준으로 한 깊이 이동 (z축을 따라 이동)	공통된 초점을 갖는 확장 벡터의 집합	
뷰 축(view axis)에서 일정한 거리(깊이)에서 회전 (z축을 중심으로 회전)	하나의 원 중심에서의 움직임 벡터 집합.	
뷰 축(view axis)에 수직인 평면 물체 회전 (y축 중심으로 회전)	직선 세그먼트에서 시작하는 하나 이상의 벡터 집합	

16

Optical Flow

- **응용 분야**
  - 객체 모션 감지
  - 행동 인식
  - 능동적 인 시력 또는 운동 구조
  - 깊이 정보를 계산하여 3D 객체 재구성
  - 거리(깊이) 맵이 있는 경우 오브젝트의 표면을 재구성 가능
  - 표정 인식
    - 참조 → http://athos.rutgers.edu/~decarlo/pubs/ijcv-face.pdf

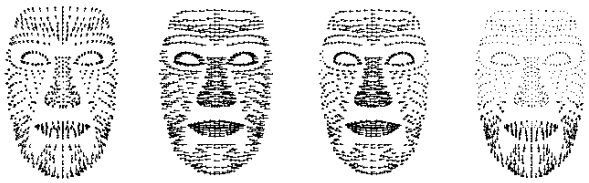


Figure 3: Sample vector fields for various motion parameters

17

Optical Flow

두 연속 프레임

- **cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, winSize, maxLevel, criteria) → nextPts, status, err**
  - **prevImg** – buildOpticalFlowPyramid()에 의해 생성된 첫번째 8 비트 입력 이미지 또는 피라미드
  - **nextImg** – **prevImg**와 같은 크기의 두번째 입력 이미지 또는 피라미드
  - **prevPts** – opticalflow 2D 벡터
  - **nextPts** – 두번째 이미지에서 입력 특징점의 새로운 위치 벡터 OTFLOW_USE_INITIAL_FLOW 플래그가 전달되면, 벡터는 입력과 같은 크기를 가짐.
  - **winSize** – (각 피라미드 수준에서) 검색창의 크기
  - **maxLevel** – 0부터 시작하는 최대 피라미드 레벨 수. 0으로 설정하면 피라미드는 사용되지 않으며(단일 레벨), 1로 설정된 경우 두 레벨이 사용됨.
  - **criteria** –반복 검색 알고리즘의 종료 기준을 지정하는 매개 변수(지정된 최대 반복 횟수 criteria.maxCount 또는 검색창이 criteria.epsilon 보다 작게 움직일 때)

- **cv2.calcOpticalFlowFarneback(prev, next, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags[, flow]) → flow**
  - 보다 조밀한(dense) opticalflow 계산
  - 단점 : 속도가 느림

18

Optical Flow

- **cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, winSize, maxLevel, criteria) → nextPts, status, err**
  - **원리** : 큰 움직임을 계산하지 못하는 단점을 개선하기 위해 영상 스케일에 따른 영상 피라미드를 구성한다.

  - **단점** : 몇 개의 특징점을 추출하고 그 특징점에 대하여 Optical Flow를 계산하기 때문에 Dense optical flow 보다는 정확성이 떨어진다는 단점이 있다.
  - **장점** : 영상 피라미드의 상위계층에서 하위계층으로 추적하면서 다양한 스케일의 이미지를 탐색하기 때문에 커다란 움직임도 찾아 낼 수 있다.

19

예제 : calcOpticalFlowPyrLK

```
#파이썬으로 만드는 OpenCV 프로젝트

cap = cv2.VideoCapture('images/walking.avi')
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)

color = np.random.randint(0,255,(200,3)) # 추적 경로를 그리기 위한 랜덤 색상
lines = None  #추적 선을 그릴 이미지 저장 변수
prevImg = None  # 이전 프레임 저장 변수
# calcOpticalFlowPyrLK 중지 요건 설정
termcriteria =  (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)

while cap.isOpened():
    ret,frame = cap.read()
    if not ret:
        break
    img_draw = frame.copy()

    cv2.imshow('OpticalFlow-LK', img_draw)
    key = cv2.waitKey(delay)
    if key == 27 : # Esc:종료
        break
```

20

예제 : calcOpticalFlowPyrLK

```
img_draw = frame.copy()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
if prevImg is None: # 최초 프레임 경우
    prevImg = gray
    lines = np.zeros_like(frame) # 추적선 그릴 이미지를 프레임 크기에 맞게 생성
    prevPt = cv2.goodFeaturesToTrack(prevImg, 200, 0.01, 10) # 추적 시작을 위한
코너 검출 ---①
else:
    nextImg = gray
    # 옵티컬 플로우로 다음 프레임의 코너점  찾기 ---②
    nextPt, status, err = cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPt,
None, criteria=termcriteria)
    prevMv = prevPt[status==1] # 대응점이 있는 코너, 움직인 코너 선별 ---③
    nextMv = nextPt[status==1] # 대응점이 있는 코너, 움직인 코너 선별 ---③
    for i,(p, n) in enumerate(zip(prevMv, nextMv)):
      px,py = p.ravel()
      nx,ny = n.ravel()
      cv2.line(lines, (px, py), (nx,ny), color[i].tolist(), 2) # 이전 코너와
새로운 코너에 선그리기 ---④
      cv2.circle(img_draw, (nx,ny), 2, color[i].tolist(), -1) # 새로운 코너에
점 그리기
    img_draw = cv2.add(img_draw, lines) # 누적된 추적선을 출력 이미지에 합성 ---⑤
    prevImg = nextImg                # 다음 프레임을 위한 프레임과 코너점 이월
    prevPt = nextMv.reshape(-1,1,2) # 다음 프레임을 위한 프레임과 코너점 이월
cv2.imshow('OpticalFlow-LK', img_draw)
```

21

예제 : calcOpticalFlowPyrLK

22

추적 : Mean-Shift

- **Mean-Shift**
  - 관심영역(ROI, Region of Interest)을 **초기 탐색 윈도우**로 이용하여 가장 밀도가 높은 곳인 물체 중심을 반복적으로 탐색
  - 히스토그램 역투영으로 물체를 크기 변화없이 추적
    - 추적할 대상 영역의 히스토그램을 저장
    - histogram backprojection : 영상의 픽셀값들을 확률값으로 변경
    - 초기 탐색 윈도우를 이용하여 물체의 중심을 반복적으로 탐색
    - 확률값 분포에 mean shift를 적용 물체 추적
  - 단점 : 최적의 ROI 지정이 힘들다, 속도 느림, 비슷한 색상에서 에러, 크기가 변하면 에러

  - **`cv2.meanshift(probImage, window, criteria)` → `retval, window`**
    - `window`는 검색창의 초기 위치 및 크기
    - 반환값 : `retval`는 탐색 반복횟수, `window`는 추적결과 윈도우

23

추적 : CamShift

- **CamShift** (Continuously Adaptive Mean Shift Algorithm)
  - A. Adam, E. Rivlin, and I. Shimshoni, Robust fragments-based tracking using the integral histogram. In: *2006 IEEE Conference on Computer Vision and Pattern Recognition*. 2006.
  - Mean-Shift의 단점 보강
  - 초기 탐색 윈도우를 이용하여 물체의 중심, 크기, 방향 추적 -> 윈도우 크기 자동 조절
    - 검출된 객체의 Hue 값의 분포를 이용하여 추적
  - 장점 : 속도빠름, 추적 객체크기(ROI) 자동 조절
  - 단점 : 조도변화, 잡음에 영향을 많이 받음, 객체가 여러 색이면 Mean-Shift보다 성능 떨어짐

  - **`cv2.camshift(probImage, window, criteria)` → `box, window`**
    - `window`는 검색창의 초기 위치 및 크기
    - 반환값 : `retval`는 탐색 반복횟수, `box`는 회전가능한 추적결과 박스

24

예제 10.7 : meanShift

```
term_crit = (cv2.TERM_CRITERIA_MAX_ITER+cv2.TERM_CRITERIA_EPS,10, 1)

x1, y1, x2, y2 = roi
mask_roi = mask[y1:y2, x1:x2]
hsv_roi  = hsv[y1:y2, x1:x2]

hist_roi = cv2.calcHist([hsv_roi], [0], mask_roi, [16], [0,180])
cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX)
track_window = (x1, y1, x2-x1, y2-y1)
tracking_start = True

if tracking_start:
    backP = cv2.calcBackProject([hsv], [0], hist_roi, [0,180], 1)
    backP &= mask
    cv2.imshow('backP',backP)

    ret, track_window = cv2.meanShift(backP, track_window, term_crit)
    x,y,w,h = track_window
    cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255),2)
```

25

예제 10.7 : camShift

```
x1, y1, x2, y2 = roi
mask_roi = mask[y1:y2, x1:x2]
hsv_roi  = hsv[y1:y2, x1:x2]

hist_roi = cv2.calcHist([hsv_roi], [0], mask_roi, [16], [0,180])
cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX)
track_window = (x1, y1, x2-x1, y2-y1)
tracking_start = True

if tracking_start:
    backP = cv2.calcBackProject([hsv], [0], hist_roi, [0,180], 1)
    backP &= mask
    cv2.imshow('backP',backP)

    track_box, track_window = cv2.CamShift(backP, track_window, term_crit)
    x,y,w,h = track_window
    cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0),2)
    cv2.ellipse(frame, track_box, (0, 255, 255), 2)
    pts = cv2.boxPoints(track_box)
    pts = np.int0(pts) # np.int32
    dst = cv2.polylines(frame,[pts],True, (0, 0, 255),2)
```

26