COMPUTER VISION

MIGUEL ARAUJO @MARAUJOP

HTTP://BIT.LY/CODEMOTION2013
DISCLAIMER
JUST AN AMATEUR

HTTP://BIT.LY/CODEMOTION2013
FLORIDA

TURTLE

HELPING SEA TURTLES SURVIVE
RED LIGHT HAL
HARDWARE
CAMERAS

- Compact cameras
- DSLR cameras (Reflex)
- Micro cameras
- USB cameras (webcams)
- IP cameras
- Depth field / 3D cameras
CHOOSING A CAMERA

- Volume / Weight
- Size of the sensor, bigger is always better
- Focal Length
- Resolution
- Light conditions
- Adjustable
- Price
PHOTOGRAPHY 101

3 PILLARS

• Shutter speed
• Aperture
• ISO (Film speed)


ALSO

• White balance
• etc
SHUTTER SPEED

Fig. 1.6.2 The appearance of the shutter at various speeds

APERTURE

Depth of field

LIBGPHOTO2

- Linux Open Source project
- Handles digital cameras DSLRs/compact cameras through USB.
- Supports MTP and PTP v1 & v2.
VISION

Compact Cameras

• Many take from 6-15 seconds using libgphoto2.
• Rarely can stream video in real time.
• Rarely can adjust camera settings on the go.
VISION

DSLRs

- Good time response.
- Very well supported, many features.
- Many camera parameters adjustable on the fly.
VISION

Micro Cameras

• Custom drivers
• Proprietary ports
VISION

Webcams

• Bad resolution
• Handled through V4L2
• Poor performance in bad lighting conditions
• Not very adjustable
EXTRA

- Lenses
- Number of cameras
SOFTWARE
OPENCV

- Open Source
- Known and respected
- C++ powered
- Python bindings
- Low level concepts, hard for newbies
- opencv-processing and others
SIMPLECV

- Built on top of OpenCV using Python
- Not a replacement
- High level concepts and data structures
- It also stands on the shoulders of others giants: numpy, Orange, scipy...
- Well, yeah, it uses camelCase
- simplecv-js
HELLO WORLD
COORDINATES

\[(0,0)\]  

\[x\]  

\[y\]
FEATURE DETECTION

- Edges
- Lines
- Corners
- Circles
- Blobs
A region of an image in which some properties are constant or vary within a prescribed range of values.

Blue M&Ms are blobs

```python
m_and_ms = Image('m&ms.jpg')
blue_dist = m_and_ms.colorDistance(Color.BLUE)
blue_dist.show()
```
blue_dist = blue_dist.invert()
blobs = blue_dist.findBlobs()
print len(blobs)
>> 122

blobs.draw(Color.RED, width=-1)
blue_dist.show()
findBlobs(minsize, maxsize, threshval, ...)

```python
blue_dist.findBlobs(minsize=200)
blobs = blobs.filter(blobs.area() > 200)
len(blobs)
>>> 36

average_area = np.average(blobs.area())
>>> 37792.77

blue_dist = blue_dist.scale(0.35)
blobs = blue_dist.findBlobs(threshval=177, minsize=100)
len(blobs)
>>> 25
```
RULES

- Dynamic is better than fixed, but harder to achieve.
- If color is not needed, drop it, at least until needed.
- The smaller the picture, less information, faster processing.
- Always use the easiest solution, which will usually be the fastest too.
- Real life vs laboratory situations.
- Some things are harder than they look like.
- When working in artificial vision, don't forget about other input sources (time, sounds, etc).
GOLDEN RULE

• Always do in hardware what you can do in hardware.
COLOR SPACES

RGB / BGR
image.toRGB()

HSV (HUE SATURATION VALUE)
image.toHSV()

YCBCR
image.toYCbCr()

http://bit.ly/1dSSoI2
blue_hue_dist = m_and_ms.hueDistance((0,117,245))
blue_hue_dist = m_and_ms.hueDistance(Color.BLUE)
BINARIZE

• Creates a binary (black/white) image. It's got many parameters you can tweak.
• Use Otsu's method by default, adjusting the threshold dynamically for better results.

```python
blue_dist.binarize(blocksize=501).show()
```
MATCHING
Detector

Descriptor

Matcher

Filtering or Pruning best matches
DETECTORS

They need to be effective with changes in:

- Viewpoint
- Scale
- Blur
- Illumination
- Noise
DETECTORS

Find ROIs

CORNERS
- Hessian Affine
- Harris Affine
- FAST

KEYPOINTS
- SIFT
- SURF
- MSER
- ORB (Tracking)
- BRISK (Tracking)
- FREAK (Tracking)

MANY MORE
DESCRIPTORS

Speed vs correctness

- SURF
- SIFT
- LAZY
- ORB
- BRIEF
- RIFF
- etc.
MATCHERS

- FLANN
- Brute Force
PRUNING

- Cross-check
- Ratio-Test
- shape overlapping
MATCHING

- Template or Query image (Choose wisely)
- Sample or Train image
```
result_image = sample.drawKeypointMatches(template)

skp, tkp = sample.findKeypointMatches(template)

skp - Keypoints matched in sample
tkp - Keypoints matched in template
```
**FINDKEYPOINTMATCH**

- Detection: Hessian affine
- Description: SURF
- Matching: FLANN Knn
- Filtering: Lowe's ratio test
- find an Homography
- Returns a FeatureSet with one KeypointMatch
Eat Bright
Eat Right

Lemon Blueberry Chicken Salad
(recipe on back)

SAVE 75¢
on any ONE (1) DOLE® Salad Blend,
All Natural Salad Kit or Extra Veggie™ Salad
(Excludes DOLE® Classic Iceberg, Shreds and non-kit Coleslaws)
coupons = Image("coupons.jpg")
coupon = Image("coupon.jpg")
match = coupons.findKeypointMatch(coupon)
match.draw(width=10, color=Color.GREEN)
uno.save("result.jpg")
2ND EXAMPLE
MANY OUTLIERS
def find_clusters(keypoints, separator=None):
    features = FeatureSet(keypoints)
    if separator is None:
        separator = np.average(features.area())

    features = features.filter(
        features.area() > separator
    )

    return features.cluster(
        method="hierarchical",
        properties="position"
    )
def find_biggest_cluster(clusters):
    max_number_of_clusters = 0
    for cluster in clusters:
        if len(cluster) > max_number_of_clusters:
            biggest_cluster = cluster
            max_number_of_clusters = len(cluster)

    return biggest_cluster
Point = namedtuple('Point', 'x y')
def distance_between_points(point_one, point_two):
    return sqrt(
        pow((point_one.x - point_two.x), 2) + \n        pow((point_one.y - point_two.y), 2)
    )

skp_set = FeatureSet(biggest_cluster)
x_avg, y_avg = find_centroid(skp_set)
centroid = Point(x_avg, y_avg)
uno.drawLineRectangle(
    x_avg, y_avg, 20, 20, width=30, color=Color.RED
)
distances = []
for kp in biggest_cluster:
    distances.append(distance_between_points(kp, centroid))

mu, sigma = cv2.meanStdDev(np.array(distances))
mu = mu[0][0]
sigma = sigma[0][0]

for kp in skp:
    if distance_between_points(kp, centroid) < (mu + 2*sigma):
        uno.drawRect(kp.x, kp.y, 20, 20, width=30, color=Color.GREEN)
NORMAL DISTRIBUTION
REAL WORLD EXAMPLE
58%

26%

29%

48%
DETECTION
HAAR

FACE DETECTION
Haar-like features 2001 Viola-Jones
HAAR

- Needs to be trained with hundreds/thousands
- Scale invariant
- NOT Rotation invariant
- Fast and robust
- Not only for faces

How face detection works
friends.listHaarFeatures()
['right_ear.xml', 'right_eye.xml', 'nose.xml', 'face4.xml', 'glasses.xml', ...

faces = friends.findHaarFeatures("face.xml")
faces.draw(width=10, color=Color.RED)
faces.save('result.jpg')
VIDEO DEMO

http://www.youtube.com/watch?v=VP3h8qf9GZ4
TRACKING
TRACKING

- Detection != tracking
- Uses information from previous frames
- Initially tracks what we want

SOME ALTERNATIVES

- Optic Flow: Lucas-Kanade
- Descriptors: SURF
- Probability/Statistics and histograms: Camshift
CAMSHIFT

- Effective for tracking simple and constant objects with homogeneous colors, like faces.
- Gary Bradski in 1998
- Original implementation has problems with similar color objects around or crossing trajectories and lightning changes.
from SimpleCV import *

video = VirtualCamera("jack.mp4", 'video')
video_stream = VideoStream(
    "jack_tracking.mp4", framefill=False, codec="mp4v"
)

track_set = []
current = video.getImage()

while (disp.isNotDone()):
    frame = video.getImage()
    track_set = frame.track('camshift', track_set, current, [100, 100, 50, 50])
    track_set.drawBB()
current = frame
frame.save(video_stream)
VIDEO DEMO

http://www.youtube.com/watch?v=QHOYG_CYPKo
MORE COMPLEX

Initialization

```python
video_stream = VideoStream(
    "jack_tracking.avi", framefill=False,
    codec="mp4v"
)
video = VirtualCamera("jack.mp4", 'video')

disp = Display()

detected = False
current = video.getImage().scale(0.6)
tracked_objects = []
last_diff = None
```
while (disp.isNotDone()):
    frame = video.getImage().scale(0.6)

    # Scene changes
    diff = cv2.absdiff(frame.getNumpyCv2(), current.getNumpyCv2())
    if last_diff and diff.sum() > last_diff * 6:
        detected = False
    last_diff = diff.sum()

    # Detects faces and restarts tracking
    faces = frame.findHaarFeatures('face2.xml')
    if faces and not detected:
        tracked_objects = []
        final_faces = []
        for face in faces:
            if face.area() > 65:
                tracked_objects.append([])
                final_faces.append(face)
                detected = True
# Restart if tracking grows too much
if detected:
    for i, track_set in enumerate(tracked_objects):
        track_set = frame.track(
            'camshift', track_set, current,
            final_faces[i].boundingBox()
        )

    # Restart detection and tracking
    if track_set[-1].area > final_faces[i].area() * 3 \
        or not detected:
        detected = False
        break

    # Update tracked object and draw it
    tracked_objects[i] = track_set
    track_set.drawBB()

    current = frame
    frame.save(video_stream)
MOG
• Separate people and objects that move (foreground) from the fixed environment (background)
• MOG - Adaptative Mixture Gaussian Model
VIDEO DEMO

http://www.youtube.com/watch?v=wm7HWdYSYkI
mog = **MOGSegmentation**(
    history=200, nMixtures=5, backgroundRatio=0.3, noiseSigma=16,
    learningRate=0.3
)

video = VirtualCamera('semaforo.mp4', 'video')
video_stream = VideoStream("mog.mp4", framefill=False, codec="mp4v")

**while** (disp.isNotDone()):
    frame = video.getImage().scale(0.5)

    mog.addImage(frame)
    # segmentedImage = mog.getSegmentedImage()
    blobs = mog.getSegmentedBlobs()
    if blobs:
        blobs.draw(width=-1)

    frame.save(video_stream)
RED-LIGHT HAL
1- Detect if traffic light is red, otherwise it's green. Using hysteresis.
2- Project a line for runners.
3- Do MOG and pruning for finding cars.
4- When traffic light is RED, if a car blob intersects the line, then it's a runner.
5- Recognize car to count it only once.
red_light_bb = [432, 212, 13, 13]
cross_line = Line(
    frame.scale(0.5), ((329, 230), (10, 360))
)

RED = False
number_of_opposite = 0
HISTERESIS_FRAMES = 5
def is_traffic_light_red(frame):
    red_light = frame.crop(*red_light_bb)

    # BLACK (30, 28, 35)
    # RED (21, 17, 51)
    if red_light.meanColor()[2] > 42:
        return True

    return False
def hysteresis(red_detected=False, green_detected=False):
    global RED, number_of_opposite

    if RED and green_detected:
        number_of_opposite += 1
        if number_of_opposite == HISTERESIS_FRAMES:
            RED = False
            number_of_opposite = 0
    elif not RED and red_detected:
        number_of_opposite += 1
        if number_of_opposite == HISTERESIS_FRAMES:
            RED = True
            number_of_opposite = 0
    else:
        number_of_opposite = 0
while (disp.isNotDone()):
    frame = video.getImage()
    small_frame = frame.scale(0.5)
    mog.addImage(small_frame)

    if is_traffic_light_red(frame):
        hysteresis(red_detected=True)
        if RED:
            blobs = mog.getSegmentedBlobs()
            if blobs:
                big_blobs = blobs.filter(blobs.area() > 1000)

                for car in big_blobs:
                    if cross_line.intersects(car.getFullMask()):
                        # RED LIGHT RUNNER
                        small_frame.drawRectangle(*car.boundingBox(), color=Color.RED, width=3)
                    else:
                        hysteresis(green_detected=True)

    small_frame.save(disp)
VIDEO DEMO

http://www.youtube.com/watch?v=RfG0HTiuBYY
FIRST PROTOTYPE
RASPBERRY

- Raspberry SimpleCV Raspicam
- Autonomous system, ethernet connected, uploads runner videos online.
- No night time support yet.
- Slower, not real time, discards green parts.
THANKS

QUESTIONS?