Video event detection for the physically impaired

Patrick Green

January 4, 2021

1 Introduction

We were assigned the task of detecting a person in a room given a set of consecutive frames, spaced at 1 second each with some frames missing, from a static point of view in RGB color space. We used a subtraction of consecutive frames followed by dilation and eroding to obtain a binary image. We then detected the center of mass by first finding the center of mass of all the activity in the binary image followed by successive center of mass calculations on smaller windows reducing by 50 pixels, centered on the last calculated center of mass. We then used activity in hard coded regions of the binary image to decipher whether the agent was in or out the room. Lastly we smoothed the motion of the center of mass using a Savagol filter.

2 Background

2.1 Static Background Subtraction

The first technique we attempted to use was that of background subtraction. Some problems were identified in using this approach: There are objects which change position in the duration of the video which means that subtracting a static background from each frame gave us very noisy results. The lighting in the room changes significantly throughout the day and poses a big problem for this method.

2.2 Frame Differencing

The final approach we used was frame differencing. Computing the absolute difference of consecutive frames yielded very good results right away, therefore it was the technique we explored most for this assignment. 2 different approaches were tested using frame differencing. In this technique we take: We attempted to perform frame differencing on the normalized images. The normalization and differencing was attempted both in RGB space and grayscale space, however the resulting frame was still quite noisy.

$$r, g, b = \frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B}$$

Differencing of consecutive frames in RGB space produced good results, with very little post-processing and the moving parts of the person were clearly visible with less noise than the other techniques we tried.

2.3 Mean Filter

Another technique that was considered was using adaptive background subtraction. In this approach, instead of using a static background that is subtracted from each image, we compute a running average over some window of frames, the size of which is a parameter to the approach. Given that the simple frame differencing produced excellent results for this very limited task, we decided not to implement this more computationally expensive and also parameterized approach. Possible drawbacks in this approach include the fact that in many frames there is very little movement and therefore averaging over the previous frames would result in small movements being lost and also when the person is moving for example from the door to cabinet, the whole image has changed over the past few frames therefor detecting the center of the moving figure could be more complicated than necessary.

2.4 Other Approaches

More complex approaches include Running Gaussian Average which characterizes each pixel by a Gaussian Distribution and Mixture of Gaussians (Background mixture models) which aim to characterize pixels as a mixture of Gaussians in order to classify the pixel as background or foreground were considered but were not implemented since the more computationally efficient method worked with very high success rate.

2.5 Object detection

The final algorithm uses a simple frame-differencing technique. The activity in the subtraction was converted to a binary image by including every pixel above the value of 80. To detect the center of mass of the movements in the frames, which is then used to recursively compute the average activity

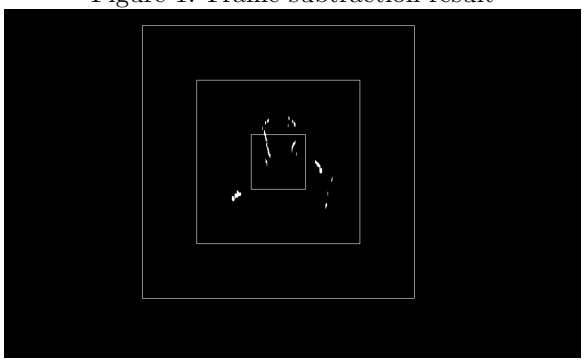
$$\frac{1}{Area(w[1], w[0])} \sum_{x,y=w[0]}^{w[1]} img[x, y]$$

in the area where the center of mass was detected is successively smalled windows. This area activity measures are used to activate the detector and aim to replace contour detection and finding contour sizes in order to speed up computation.

```
# Find differences
Img_diff = compute_absolute_diff(im_{t-1}, im_{t})
Img_diff = convertToGrayscale(im_diff,0,255)
```

```
Im_thresh = BinaryThreshold(im_diff,80,255)

# remove small noise
Im_thresh = Erode(Im_thresh, (1, 1), iterations=1)

# connect nearby small components and remove small ones
Im_thresh = Dilate(Im_thresh, (1, 1), iterations=4)
Im_thresh = Erode(Im_thresh, (1, 1), iterations=3)
Im_thresh = Dilate(Im_thresh, (1, 1), iterations=4)

Total_activity =  avg_act(im_thresh, all)
if total activity > 0.05:
  CoM0 = compute_center_of_mass(im_thresh,all)
  Avg_win_act0 = avg_act(im_thresh, (250px-wide window around CoM))
  CoM1 = compute_center_of_mass(im_thresh,all)
  Avg_win_act1 = avg_act(im_thresh, (150px-wide window around CoM1))
  CoM2 = compute_center_of_mass(im_thresh,all)
  Avg_win_act2 = avg_act(im_thresh, (50px-wide window around CoM2))

  # Check the ratio of activities to ensure that the activity is indeed
  # at the center of mass and is not the result of scattered activity
  CoM = RatioCheck ( Avg_win_act0, Avg_win_act1 , Avg_win_act2)
else:
  CoM = None
```

Figure 1: Frame subtraction result

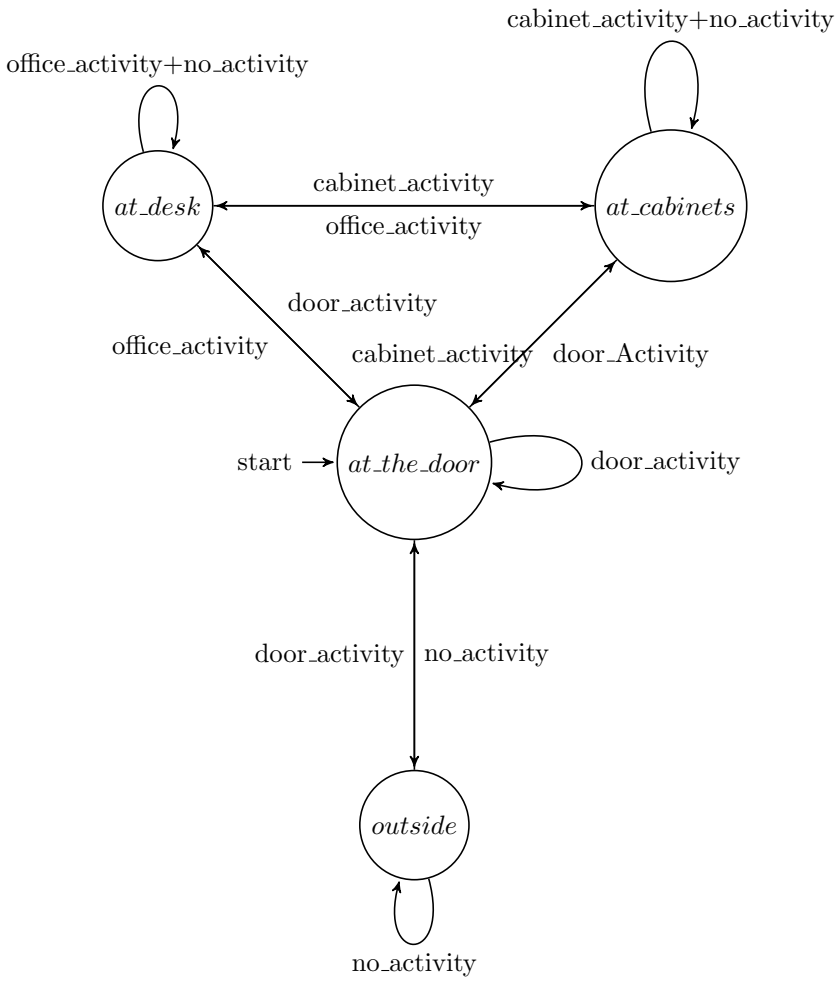


3 Modeling Behavior

3.1 State Machine

To handle the cases where the agent was not present we built a state machine to remove implausible transitions, for example moving in and out of the room

without passing through the door. The state machine for this filtering of transitions is presented bellow.



Edges with the largest activity are transversed every frame. The values of $office\_activity$,$cabinet\_activity$ and $door\_activity$ are scaled dependent on the activity in the corresponding bounding boxes. This is later normalized across all the decisions that the current state can make. The boxes are marked on the graphic bellow. The value of $no\_activity$ is set to 1 when $total\_activity$ is not 0 and handles cases where the frame subtraction reveals no difference and ensures the leaving of the room or the continuation of the current state. This prevented any miss detection of frames following frames with no motion and allows no center of mass to be drawn when the agent is outside the room.

Figure 2: Bounding boxes marking zones to measure activity levels. Yellow corresponds to the door_activity, Blue to the office_activity and Green to the cabinet_activity

3.2 Statistics

The statistics were obtained by calculating whether the center of mass lye withing one of the 3 bounding boxes or not. The *outside* state was used to decipher the frames where the agent was outside or inside the room and also allowed us to obtain the time stamp the points for which he entered and exited.

```
if old_state == "outside" and state != "outside":
   out_to_in += 1
if old_state != "outside" and state == "outside":
   in_to_out += 1
if not center_of_mass:
   out_c = out_c + 1
   not_desk_c = not_desk_c+ 1
elif is_in(center_of_mass, room.office):
   desk_c = desk_c+ 1
elif is_in(center_of_mass, room.cabinet):
   cabinet_c +=1
   not_desk_c +=1
else:
   not_desk_c += 1
```

4 Smoothing

4.1 Savitzky Golay filter

A Savgol filter was chosen to smooth the data as it allowed for only movement with along a polynomial of chosen degree. This filtered out all high frequency cases when the center of mass changed position abruptly and its trajectory could not be modeled by a smooth polynomial. We chose a value of 5 for the order of the polynomial and a windows size of 27. As several of adjacent

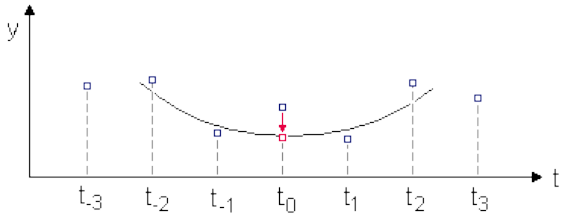
frames had varying time intervals the window size was adapted to allow for large movements in these specific periods. The approach smooths data similar to a moving average. The new data value of Y_j is derived using the formula bellow where m is the window size.

$$Y_j = \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} C_i y_{j+i}$$

The coefficients C_i are dependent on the order of the polynomial for example a 2^{nd} order polynomial with 7 data points would have the following formula.

$$y_t = (-2x_{t-3} + 3x_{t-2} + 6x_{t-1} + 7x_t + 6x_{t+1} + 3x_{t+2} + 2x_{t+3})/21$$

Figure 3: Aid to explain Savgol filtering



As the Savgol filter is for use on a 1D array the filter it has been applied to the x and y coordinates separately. This is a simplification which worked well in our case as movement was generally limited to either raising from a chair and moving right to left. However in the general the x and y coordinates are not independent.

5 Results

The algorithms presented above have been tested against 105 labeled images. The labeling was done by clicking on the image where we think the center of mass would be. In the cases where the subject is occluded, the mass label was placed considering the visible parts of the body.
The final result, as measured on a subset of 85 of 105 images that contained a person, was a RMSE of 64.85 pixels without the Savitzky Golay filter and 80 with it. Out of the 20 labeled frames where nobody was in the room, there were no occasions where a detection was reported. Detailed statistics are given below.

Table 1: Counts from all frames

Activity	Number of frames
Working at the desk	5324
No one in the office	1309
At the filing cabinet	36
Entered the office	2
Exiting the office	2
Total number of frames	6719

Table 2: Error between manual and automatic detection

Filtering	RMSE (in pixels)	Labeled images with presence	Total labeled images
With SavGol	80.3169456884	85	105
W/o SavGol	64.8537743558	85	105

5.1 Strengths

The algorithms main strength were that we were able to reuse the last frames center of mass when the state machine detected no change of state. This allowed for us to label frames where frame subtraction did not yield any activity. The continuation of a center of mass is presented in figure 4 with the pink circle instead of the green. The first binary images shows a small amount of activity and the second shows none. Both binary images are created using the difference of the image to th left of it to it and the frame before that.

Figure 4: Detecting the center of mass correctly on two consecutive frames, one with activity and the other with none

In additions we were able to prevent mis detections when the agent was not present in the room using the current state to prevent the drawing of an incorrect center of mass.

Another strong feature of this method is performance. All 6700 frames are processed in under 5 minutes on a desktop-class machine when showing the frames and in around 2 minutes when the activity is not shown in real-time and drawing on frames is not performed.

5.2 Weaknesses

The algorithms main weakness are that the mass is computed from the absolute difference and therefore when there is very large motion, the subject appears twice in the difference frame which means that the center of mass appears to be lagging behind its actual position in such cases. This effect might was enhanced by a Savitzky Golay filter which pulled the center of mass a few pixels behind however this gave a poorer RMSE as its influence on more quiet frames of very limited movement and lots of small activities. Furthermore the filter was not tuned to missing frames and therefore would smooth by the same degree for all time transitions which also would account for at 16 pixel increase in the RMSE.

The sequence of images below outline the effect described above.

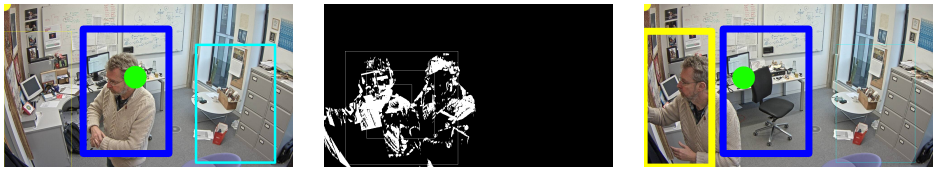


Figure 5: Incorrectly identifying the center of mass

6 Conclusion

In this report we have shown an algorithm to detect a person in a room from a static camera. The algorithm combines simple image processing techniques (images subtraction, erosion and dilation,thresholding, finding the center of mass) and application of Domain Specific Knowledge (dynamics of the room in the form of a state machine). The result was a very robust detection algorithm given the conditions which gave no mis detections and a low error for the center of mass. From this task we have learned that by applying simple techniques combined with a good model of the situation that is to be analyzed we can produce good tracking of a person. To use this approach in another environment we would have to create a different model of the environment or change the way the center of mass is detected.

7 Appendix

7.1 Time stamps of frames marking the entrance and exit of the office

OUT
*inspacecam*163_2016_02_19_13_02_40.*jpg*
*inspacecam*163_2016_02_19_13_33_43.*jpg*

IN
*inspacecam*163_2016_02_19_13_05_59.*jpg*
*inspacecam*163_2016_02_19_13_54_39.*jpg*

7.2 Movements of center of mass detection over all frame

Figure 6: Trajectory of agent over all frames

7.3 Code

```
# main.py

# import the necessary packages
import os
# import matplotlib.pyplot as plt
import argparse
import glob
import cv2
```

```
import numpy as np
from scipy import ndimage
from math import ceil
from tools import *
from room import roomimage, binroomimage

# from pykalman import KalmanFilter
# from tochroma import *

cv2.namedWindow("Frame", 0)
cv2.namedWindow("Final", 0)

parser = argparse.ArgumentParser(description='Basic Single Person Motion Tracker')
parser.add_argument('-c', action="store", dest="c", type=int)
parser.add_argument('-f', action="store", dest="f", type=int, default = 0)
parser.add_argument('-w', action="store", dest="w", type=int, default=1)
parser.add_argument('-p', action="store", dest="folder", type=str)
parser.add_argument('--fast', action="store_true", dest="fast")
parser.set_defaults(feature=False)
args = parser.parse_args()


errors = []
room = roomimage()
binaryroom = binroomimage()
prev_com = None
center_of_mass = None
miss_detections = []
state = "start"
positions = []

# Zero the Statistics Coutners
out_c, desk_c, cabinet_c, not_desk_c = 0,0,0,0
in_to_out, out_to_in = 0,0
intoout,outtoin = [], []
with file("transpoints.txt","w") as f:
 x=args.f
 for waittime,im3,im4,im_label,frame_name in getimages(args):
  prev_com = center_of_mass
  room.image = im3


  # Find difference
  th4 = cv2.absdiff(im3, im4)
  # convert to grayscale
  th4 = cv2.cvtColor(th4, cv2.COLOR_BGR2GRAY)
```

```
# remove noise
ret, thresh1 = cv2.threshold(th4, 80, 255, cv2.THRESH_BINARY)

# cleanup
thresh1 = cv2.erode(thresh1, (1, 1), iterations=1)
thresh1 = cv2.dilate(thresh1, (1, 1), iterations=4)
thresh1 = cv2.erode(thresh1, (1, 1), iterations=3)
thresh1 = cv2.dilate(thresh1, (1, 1), iterations=4)

binaryroom.image = thresh1

# Calculate activity for each spot
office_activity = avg_win(room.office, binaryroom.image)
cab_activity = avg_win(room.cabinet, binaryroom.image)
door_activity = avg_win(room.door, binaryroom.image)
total_activity = avg_win(room.total, binaryroom.image)


CoM = None
CoM1 = None
CoM2 = None
swc1, swc2,center_of_mass = None, None, None

# Consider only above some activity level
if total_activity > 0.05:
 try: # Successive detection of center of mass
  # pass 1
  CoM = ndimage.measurements.center_of_mass(binaryroom.image)
  center_of_mass = [int(CoM[1]), int(CoM[0])]
  swc, win = avg_win_center(center_of_mass, 250, binaryroom.image)

  # Pass 2[w[0][1]:w[1][1],w[0][0]:w[1][0]]
  region = binaryroom.image[win[0][1]:win[1][1], win[0][0]:win[1][0]]
  CoM1 = ndimage.measurements.center_of_mass(region)
  center_of_mass = [win[0][0] + int(CoM1[1]), win[0][1] + int(CoM1[0])]
  swc1, win1 = avg_win_center(center_of_mass, 150, binaryroom.image)

  # pass 3
  region = binaryroom.image[win1[0][1]:win1[1][1], win1[0][0]:win1[1][0]]
  CoM2 = ndimage.measurements.center_of_mass(region)
  center_of_mass = [win1[0][0] + int(CoM2[0]), win1[0][1] + int(CoM2[1])]
  swc2, win2 = avg_win_center(center_of_mass, 50, binaryroom.image)

  #print "act:", swc, swc1

 except:
```

```
  #print "Error or Empty", CoM, CoM1, CoM2
  waittime = 0

 binaryroom.draw(win, win1, win2)

else:
 center_of_mass = None
 print "No activity detected"

# USE the state machine to decide whether the position is valid
old_state = state
if state and state == "outside" and total_activity > 1:
 state = room.changestate(office_activity, cab_activity, door_activity, total_activity)
 center_of_mass = None
elif state == "start" and total_activity < 0.08:
 # if we initialize on a frame with no activity, avoid getting locked outside
 pass
elif state == "outside":
 pass
 center_of_mass = None
else :
 state = room.changestate(office_activity, cab_activity, door_activity, total_activity)
print state
if not center_of_mass and prev_com != None:
 if state in ["desk","cabinet","room"]:
  center_of_mass = tuple(prev_com)

if not args.fast:
 room.draw(swc1, swc2, center_of_mass, office_activity, cab_activity, door_activity)


# Calculation of statistics
if waittime == 0 and (im_label[0]) :
 print "ACTIVITY"
 print "DOOR", "  OFFICE", "    Cabinet" , "COM"
 print door_activity, office_activity, cab_activity, center_of_mass
 if center_of_mass and swc2:
  print "Total    ", " Large  ", " Medium", "    Small"
  print total_activity, swc, swc1, swc2, center_of_mass
 if im_label[0] and center_of_mass:
  error_dist = eucl_dist(center_of_mass,im_label[0])
  errors.append(error_dist*error_dist)
 if  not center_of_mass and not im_label[0]:
  errors.append(0)
 waittime = 10
 if center_of_mass and not im_label[0]:
```

```
  miss_detections.append(x)
  waittime = 0

else:
 waittime = 10


if old_state == "outside" and state != "outside":
 out_to_in += 1
 outtoin.append(frame_name)
if old_state != "outside" and state == "outside":
 intoout.append(frame_name)
 in_to_out += 1

if not center_of_mass:
 out_c = out_c + 1
 not_desk_c = not_desk_c+ 1
elif is_in(center_of_mass, room.office):
 desk_c = desk_c+ 1
elif is_in(center_of_mass, room.cabinet):
 cabinet_c +=1
 not_desk_c +=1
else:
 not_desk_c += 1


# TODO::: COUNT TRANSITION FROM AND TO OUTSIDE

positions.append(center_of_mass)



# SHOW - skip if need fast
# im3 = clip(im3,office)
if not args.fast:
 try:
  cv2.imshow("Frame", room.image)
  cv2.imshow("Final", binaryroom.image)
  # if x%100 in [3,4,5] and False:
  #  cv2.imwrite("output2/" + str(x)+ "_frame.png", room.image)
  #  cv2.imwrite("output2/" + str(x)+ "_binary.png", binaryroom.image)

 except:
  pass
 key = cv2.waitKey(1) & 0xFF
 if key == ord("q"):
```

```
    break
    print "RMSE (in pixels)" , np.sqrt(np.average(errors))
    print "Missdetected frames", miss_detections
    print "COUTNS", out_c, desk_c, cabinet_c, not_desk_c
    print "INS AN OUTS" ,in_to_out, intoout, out_to_in, outtoin

  x+=1


 print "RMSE (in pixels)" , np.sqrt(np.average(errors)), "in ", len(errors),
 " labeled images"
 print "Missdetected frames", miss_detections
 print "COUTNS", out_c, desk_c, cabinet_c, not_desk_c
 print "INS AN OUTS" ,in_to_out, intoout, out_to_in, outtoin



for waittime,im3,im4,im_label,f_name in getimages(args):
 frst = None
 second = None
 while len(positions) > 0:
  second = positions[0]
  del positions[0]
  if frst and second:
   cv2.line(im3, tuple(frst), tuple(second), (0,0,255), 1)
  if second: frst = second
 cv2.imwrite("movementx.jpg", im3)
 break
```

```
# room.py

from tools import *
import cv2
```

```
from math import ceil
from scipy.signal import savgol_filter

class roomimage:
 def __init__(self):
  self.image = None
  # important locations
  self.office = ((350, 110), (730, 660))
  self.cabinet = ((850, 180), (1200, 700))
  self.door = ((0, 120), (300, 720))
  self.total = ((0, 0), (1280, 720))
  self.measurements = []
  self.positions = []
  self.states =
  ["working_at_desk","at_cabinets","just_in_the_room","at_the_door","outside_the_room","sta
  self.statemarkings = [self.office[0],self.cabinet[0],self.total[0],self.door[0]]
  self.tag = ["desk","cabinet","room","room","outside"]
  self.curstate = 5
  self.start = True


 def changestate(self,office_activity,cab_activity,door_activity,total_activity):
  prev_state = self.curstate
  room_activity = total_activity*area(self.total) - office_activity*area(self.office) - \
                  cab_activity*area(self.cabinet) - door_activity*area(self.door)
  no_activity = 0
  if total_activity == 0:
   no_activity =1
  transitions =  {
      "working_at_desk" :
          [office_activity+no_activity,cab_activity,room_activity,door_activity,0,0],
      "at_cabinets"  :
          [office_activity,cab_activity+no_activity,room_activity,door_activity,0,0],
      "just_in_the_room":
          [office_activity,cab_activity,room_activity+no_activity,door_activity,0,0],
      "at_the_door"  :
          [office_activity,cab_activity,room_activity,door_activity,no_activity,0],
      "outside_the_room":
          [0,0,0,door_activity,no_activity,0],
      "start"     :
          [office_activity,cab_activity,room_activity,door_activity,no_activity,0]}
  decision = np.array(transitions[self.states[self.curstate]])
  self.curstate = np.argmax(decision/np.sum(decision))
  if prev_state == 4 and (self.curstate in [0,1]):
   self.curstate = 4
  return self.tag[self.curstate]
```

```
def draw(self,swc1,swc2,center_of_mass,office_activity,cab_activity,door_activity):
 if center_of_mass:
  if self.start:
   self.measurements = np.array([[center_of_mass[0], center_of_mass[1]]])
   self.positions = np.array([[center_of_mass[0], center_of_mass[1]]])
   self.start = False
  else:
          com = [center_of_mass[0], center_of_mass[1]]
   self.measurements = np.insert(self.measurements, 1, com, axis=0)
   self.positions = np.insert(self.measurements, 1, com, axis=0)

  if len(self.measurements) > 27:
   self.measurements = np.delete(self.measurements, 0, 0)
   filtered = savgol_filter(self.measurements[:, 0], 27, 3)
   x = np.asarray(np.rint(filtered), dtype=np.dtype("int"))[0]
   filtered = savgol_filter(self.measurements[:, 0], 27, 3)
   y = np.asarray(np.rint(filtered), dtype=np.dtype("int"))[0]
   self.positions = np.delete(self.positions, -1, 0)
   center_of_mass = (x, y)
   self.positions = np.insert(self.measurements, 1, center_of_mass, axis=0)


  if swc2 and center_of_mass and swc2 > 0.05:
   cv2.circle(self.image, tuple(center_of_mass), 50, (0, 255, 255), thickness=-1)
  elif swc2 and center_of_mass and swc2 > 0.01 and swc1 > 0.005:
   cv2.circle(self.image, tuple(center_of_mass), 50, (0, 25, 255), thickness=-1)
  else:
   cv2.circle(self.image, tuple(center_of_mass), 50, (182, 24, 255), thickness=-1)



 cv2.rectangle(self.image, self.office[0], self.office[1], (255, 0, 0),
               thickness=min([int((10 * office_activity)), 30]))
 cv2.rectangle(self.image, self.cabinet[0], self.cabinet[1], (255, 255, 0),
               thickness=min([int(ceil(cab_activity)), 30]))
 cv2.rectangle(self.image, self.door[0], self.door[1], (0, 255, 255),
               thickness=min([int(ceil(door_activity)), 30]))

 if self.curstate < 4:
  cv2.circle(self.image, self.statemarkings[self.curstate], 30, (25,255,255),
                                    thickness=-1)
```

```
class binroomimage:
 def __init__(self):
  self.image = None
  self.office = ((400, 200), (650, 550))
  self.cabinet = ((850, 180), (1200, 700))
  self.door = ((0, 120), (300, 720))
  self.total = ((0, 0), (1280, 720))

 def draw(self,win,win1,win2):
  cv2.rectangle(self.image, win[0], win[1], (255, 0, 0), thickness=1)
  cv2.rectangle(self.image, win1[0], win1[1], (255, 0, 0), thickness=1)
  cv2.rectangle(self.image, win2[0], win2[1], (255, 0, 0), thickness=1)
```

```
# tools.py

import numpy as np
import os
import cv2
import glob

def fix_win(w, limits=(1280, 720)):
 w1 = max([w[0][0], 0])
 w2 = max([w[0][1], 0])
 w3 = min([w[1][0], limits[0]])
 w4 = min([w[1][1], limits[1]])
 return ((w1, w2), (w3, w4))


def eucl_dist(p1,p2):
 dx = np.abs(p1[0] - p2[0])
```

```
 dy = np.abs(p1[1] - p2[1])
 return np.sqrt(dx*dx + dy*dy)

def clip(img, w):
 return img[w[0][1]:w[1][1], w[0][0]:w[1][0]]


def is_in(p, area):
 return p[0] > area[0][0] and p[0] < area[1][0] and p[1] > area[0][1] and p[1] < area[1][1]


def sum_win_center(c, width, img):
 w = fix_win((c[1] - width, c[0] - width), (c[1] + width, c[0] + width))
 r = np.sum(img[w[0][0]:w[1][0], w[0][1]:w[1][1]])
 # This function returns the average activity in the area
 # This funciton sums up the activity white pixels in the window and divides by the total ar
 w = ((c[0] - width, c[1] - width), (c[0] + width, c[1] + width))
 return r


def sum_win(w, img):
 # This function returns the summed activity in the area
 # This funciton sums up the activity white pixels in the window and divides by the total ar
 return np.sum(img[w[0][0]:w[1][0], w[0][1]:w[1][1]])


def avg_win_center(c, width, img):
 # This function returns the average activity in the area
 # This funciton sums up the activity white pixels in the window and divides by the total ar
 w = ((c[1] - width, c[0] - width), (c[1] + width, c[0] + width))
 s = np.sum(img[w[0][0]:w[1][0], w[0][1]:w[1][1]])
 a = (w[1][0] - w[0][0]) * (w[1][1] - w[0][1])
 w = fix_win(((c[0] - width, c[1] - width), (c[0] + width, c[1] + width)))
 return s / a, w


def avg_win(w, img):
 # print w
 # This function returns the average activity in the area
 # This funciton sums up the activity white pixels in the window and divides by the total ar
 return float(np.sum(img[w[0][1]:w[1][1], w[0][0]:w[1][0]])) / ((w[1][0] - w[0][0]) * (w[1][

def area(w):
 return (w[1][0] - w[0][0]) * (w[1][1] - w[0][1])

def getimages(args):
```

```
try:
 read_dictionary = np.load('labels_xy.npy').item()
except:
 print "No labels found"

labels = read_dictionary or {}

orig = sorted(glob.glob("day2/*.jpg"))
images = sorted(glob.glob("chroma/*.jpg"))

if args.folder:
 orig = sorted(glob.glob(args.folder+"/*.jpg"))
works = args.c or 0
waittime = args.w

for x in range(len(orig) - 1):

 # skip first x frames using -fs
 if x < args.f: continue
 print "Image:", x, "of ", len(orig), orig[x + 1][32:]
 label = labels.get(os.path.basename(orig[x + 1]))
 # Pause on labeled images
 if label:
  print "LABLED IMAGE:", label
  waittime = 0
 else:
  waittime = args.w
  label = None
 # read 2 chromaticity images and computer their absolute difference
 # or do on the spot

 im3 = cv2.imread(orig[x + 1])
 im4 = cv2.imread(orig[x])
 if works == 1:
  pass
 # im1 = toChroma2(im4)
 # im2 = toChroma2(im3)
 else:
  im1 = cv2.imread(images[x])
  im2 = cv2.imread(images[x + 1])


 yield waittime, im3, im4 , label, os.path.basename(orig[x + 1])
```