# CS 6476: Computer Vision, Fall 2020

# PS5 -- Extra Credit

### Instructor: Devi Parikh

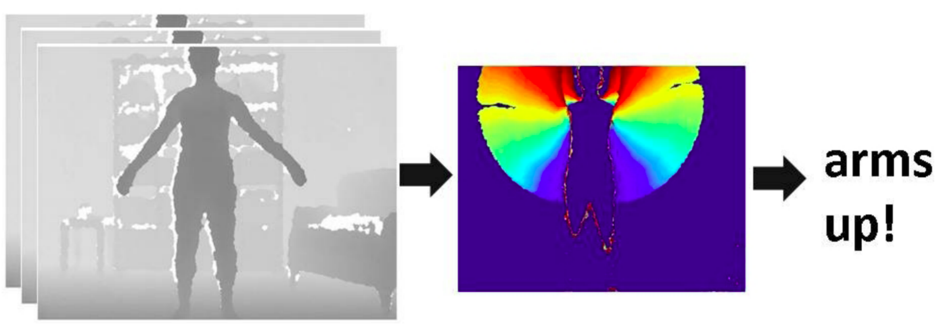
### Due: Friday, November 20, 11:59 pm

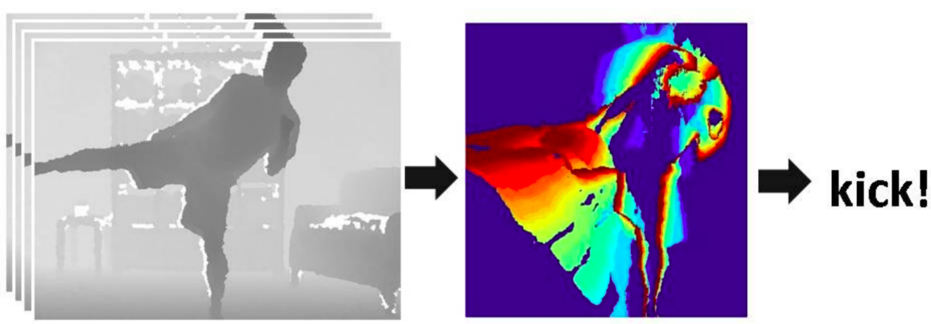
## 1. Programing Problem [100 points]

### Introduction

For this problem, you will implement an action recognition method using Motion History Images. Given a video sequence, the goal is to predict which of a set of actions the subject is performing.

The basic idea is to use a sequence of depth images to segment out the foreground person. Then for each sequence, compute its motion history image (MHI). This MHI serves as a temporal template of the action performed, and can be further compressed into a set of 7 Hu moments. Once the Hu moments have been computed for all labelled training examples, we can categorize a novel test example by using nearest neighbour classification.



## Video Data

You can access the video data here (unzipped data occupies about 530MB of disk space):

[https://gatech.box.com/shared/static/sw7dkza8drr2ouyko59haqyrczg3bbg0.zip (https://gatech.box.com/shared/static/sw7dkza8drr2ouyko59haqyrczg3bbg0.zip)](https://gatech.box.com/shared/static/sw7dkza8drr2ouyko59haqyrczg3bbg0.zip)

There are 5 directories within the zip file, each of which contains 4 sequences for one of the action categories. The 5 action categories are: botharms, crouch, leftarmup, punch, righkick. Each directory under any one of these 5 main directories contains the frames for a single sequence. For example, `punch/punch-p1-1/` contains one sequence of frames for punch.

In [1]:
```
import os
import glob
import random
import numpy as np
import matplotlib.pyplot as plt

from imageio import imread, imsave, mimsave
from IPython.display import Image

######################################################################
# TODO: Add additional imports
######################################################################

######################################################################
#                              END OF YOUR CODE
######################################################################
```

## Part 0: Working with the data

The data are stored as ".pgm" images. Each PGM is a grayscale image, where the intensity is proportional to depth in the scene.

Note that the image frames are named sequentially, so that if you use `sorted(glob.glob())` the image list will be in the correct order.

See the cells below for an examples of how to loop over the videos and read in the files.

#### Select a random sequence

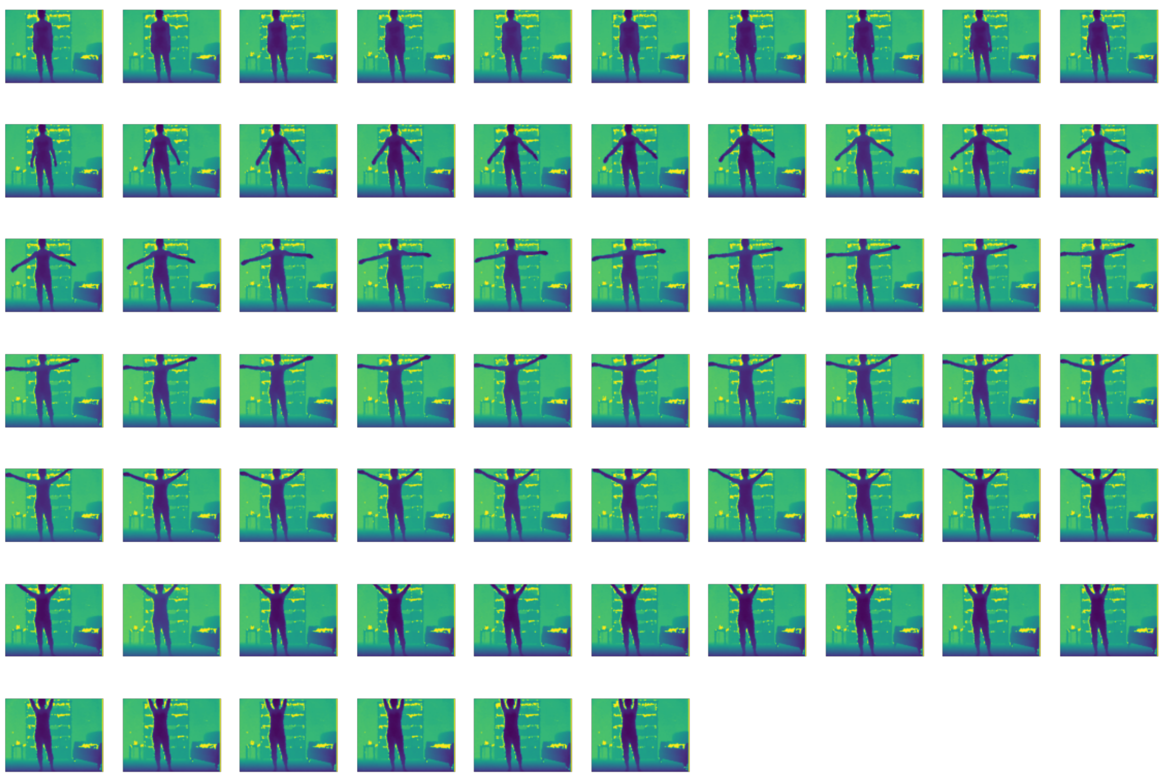
In [2]:
```
# setup
ROOT = "PS5_Data/"  # TODO: update `ROOT` to point to the "PS5_Data" f
ACTIONS = ["botharms", "crouch", "leftarmup", "punch", "rightkick"]

# select a random sequence for a random action
action = random.choice(ACTIONS)
```

```
folders = glob.glob(os.path.join(ROOT, action, "*"))
folder = random.choice(folders)

# get the sorted image files
files = sorted(glob.glob(os.path.join(folder, "*.pgm")))

# get the images
images = [np.array(imread(path)) for path in files]

print(f"Sequence: {folder}")
print(f"Sequence contains {len(files)} images")
```

```
Sequence: PS5_Data/botharms/botharms-up-p1-2
Sequence contains 66 images
```

**View all of the images in the squence**

In [3]:
```
h = 10
w = len(images) // h + 1
fig, axs = plt.subplots(w, h, figsize=(2 * h, 2 * w))
for ax in axs.flat:
    ax.axis("off")
for img, ax in zip(images, axs.flat):
    ax.imshow(img)
```



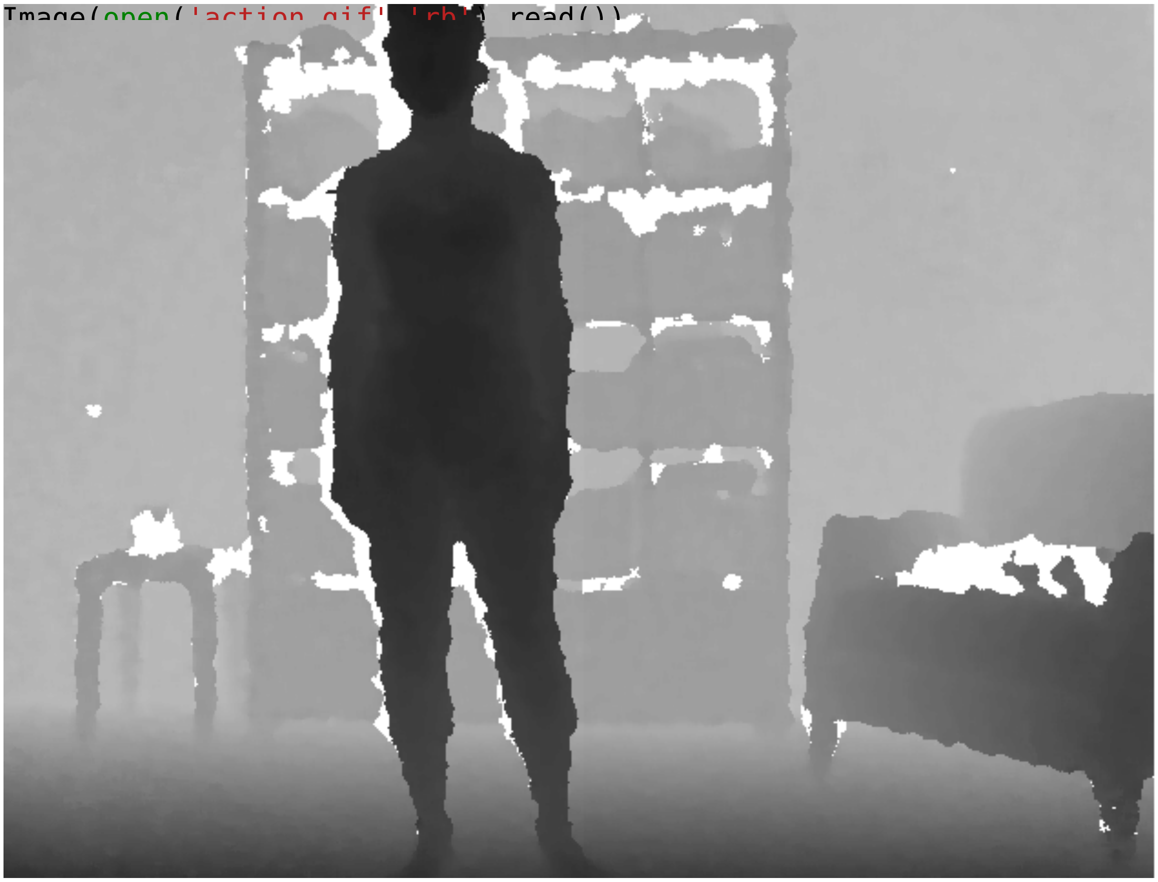
**View all of the images as a GIF**

A better way to visualize this data is as a video or gif. You can see how to do this below.

In [4]:
```
# convert images to "uint8"
converted_images = []
```

```
amin, amax = np.amin(images), np.amax(images)
for img in images:
    temp = (img.astype("float32") - amin) / (amax - amin)
    converted_images.append(np.uint8(255 * temp))

# save gif
mimsave("action.gif", converted_images)

# open gif in notebook
Image(open('action.gif','rb').read())
```

Out[4]:



## Approach Overview

The main steps are as follows:

1. Load the depth map pgms in a given sequence, and perform background subtraction by using the depth data. (Choose reasonable threshold(s) on depth based on examining an example or two)

   Now, perform adjacent two-frame differencing to identify the pixels which have changed by a reasonable amount (above a threshold). Let's call these binary foreground difference images as $D(x, y, t)$.

2. Use all difference images in a $\tau$ -frame sequence to compute its Motion History Image, $H_\tau$ :

$$H_\tau(x, y, t) = \begin{cases} \tau & \text{if } D(x, y, t) = 1 \\ max(0, H\tau(x, y, t-1) - 1) & \text{otherwise} \end{cases}$$

where $t$ varies from $1$ to $\tau$.

3. Normalize the Motion History Image (MHI) by the maximum value within it.

4. Use the MHI to compute a 7-dimensional vector containing the 7 Hu moments. This vector is the final representation for the entire video sequence, and describes the global shape of the temporal template in a translation- and rotation-invariant manner.

5. Having computed a descriptor vector for each video sequence, evaluate the nearest neighbour classification accuracy using *leave-one-out cross-validation*. That is, let every instance serve as a test case in turn, and classify it using the remaining instances.

6. For the nearest neighbour classifier, use the normalized Euclidean distance (i.e., where the distance per dimension is normalized according to the sample data's variance).

7. Evaluate the results over all sequences based on the mean recognition rate per class and the confusion matrix.

See the paper *The Representation and Recognition of Action Using Temporal Templates by J. Davis and A. Bobick* for more background on computing the MHI (available [here (https://gatech.box.com/s/j4tf5b2cwy630gdij7kwd82tz4yyeory)](https://gatech.box.com/s/j4tf5b2cwy630gdij7kwd82tz4yyeory)). For additional background on the properties of Hu moments, see *Visual Pattern Recognition by Moment Invariants by M. K. Hu* (available [here (https://gatech.box.com/s/5jva449d4hthiqyn55jh4vil1wfgudao)](https://gatech.box.com/s/5jva449d4hthiqyn55jh4vil1wfgudao)).

## Part 1. Motion History Image [30 points]

Complete the functions `background_subtraction`, `compute_two_frame_difference` and `compute_motion_history_image` as described below. Then, calculate Motion History Images (MHIs) for all of the action sequences in the dataset.

See the list of deliverables below.

In [5]:
```
def background_subtraction(image, threshold):
    """
    Returns an image with the background removed (zeroed out) wherever
    is greater than the threshold.

    Inputs:
    - image: HxW matrix - the image
    - threshold: integer - the background threshold

    Outputs:
    - foreground_image: the output image containing only the foregroun
    """
    foreground_image = image.copy()

    ##################################################################
    # TODO: Add your code here (note: this should only take a few line
    ##################################################################
    background_index = foreground_image > threshold
```

```
    foreground_image[background_index] = 0

    ####################################################################
    #                              END OF YOUR CODE
    ####################################################################

    return foreground_image
```

In [6]:
```
def compute_two_frame_difference(image1, image2, threshold):
    """
    Returns a binary image that is `True` where the absolute differenc
    the two images is above a threshold.

    Inputs:
    - image1: HxW matrix - the first image
    - image2: HxW matrix - the second image
    - threshold: integer - the difference threshold

    Outputs:
    - diff: HxW matrix the output binary image
    """
    diff = np.zeros_like(image1, dtype="bool")

    ####################################################################
    # TODO: Add your code here (note: this should only take a few line
    ####################################################################
    diffabs = abs(image1 - image2)
    diff[diffabs > threshold] = True
    ####################################################################
    #                              END OF YOUR CODE
    ####################################################################

    return diff
```

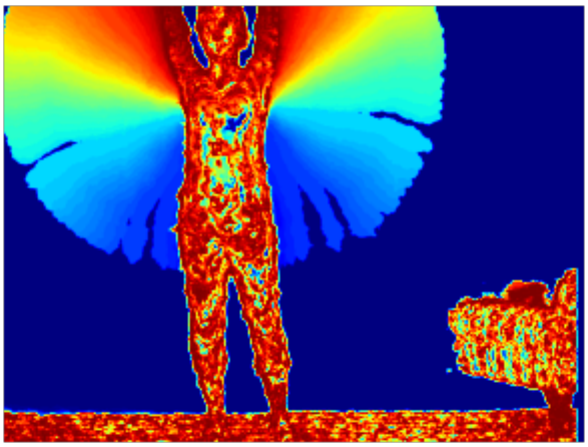
In [7]:
```
im1 = imread(r"PS5_Data/leftarmup/leftarm-up-p1-2/d-1303672778.631058-
im2 = imread(r"PS5_Data/leftarmup/leftarm-up-p1-2/d-1303672778.661285-
compute_two_frame_difference(im1, im2, 5)
```

Out[7]:
```
Array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       ...,
       [ True,  True,  True, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [ True, False,  True, ..., False, False, False]])
```

In [8]:
```
def compute_motion_history_image(images, background_subtraction_thresh
    """ Returns a Motion History Image (MHI) for an action sequence.

    Steps:
    - Use your `background_subtraction` function to remove the backgro
    - Use your `compute_two_frame_difference` function to calculate $D
      for the image sequence.
    - Calculate the MHI using the equation in step 2 of the approach o
    - Remember to normalize the Motion History Image (step 3)

    Inputs:
    - images: List of HxW matrices - the image sequence
```

```
    - background_subtraction_threshold: integer - the threshold for ba
    - difference_threshold: Integer - threshold for frame differencing

    Outputs:
    - MHI: HxW matrix - motion history image
    """
    assert len(images) > 0
    MHI = np.zeros_like(images[0])

    ###################################################################
    # TODO: Add your code here.
    ###################################################################
    firstsub = background_subtraction(images[0], background_subtractio
    for i in images[1:]:
        sub = background_subtraction(i, background_subtraction_thresho
        diff = compute_two_frame_difference(firstsub, sub, difference_
        MHI = MHI - 1
        MHI[diff == True] = len(images) - 1
        MHI[MHI < 0] = 0
        firstsub = sub
    MHI = MHI / MHI.max()
    ###################################################################
    #                              END OF YOUR CODE
    ###################################################################

    return MHI
```

In [9]:
```
##############################################################################
# TODO: Try different threshold values to get a reasonable looking MHI
#       select a threshold so that only the foreground (person) contri
#       to the MHI (small noise in the background is okay). Then, see
#       your threshold works well for other sequences by re-running th
#       random sequence selection code above.
##############################################################################

MHI_test = compute_motion_history_image(images, background_subtraction
                                        difference_threshold=0)
_, ax = plt.subplots()
ax.axis("off");
ax.imshow(MHI_test, cmap="jet");

##############################################################################
#                              END OF YOUR CODE
##############################################################################
```



#### Generate MHIs for all of the image sequnces

Use your `compute_motion_history_image` function to compute the Motion History Images (MHIs) for all the data. Save the MHIs as a 480x640x20 numpy array named `all_MHIs.npy` and submit this file.

Display all 20 MHIs in a single figure with a 5x4 grid.

#### Deliverables:

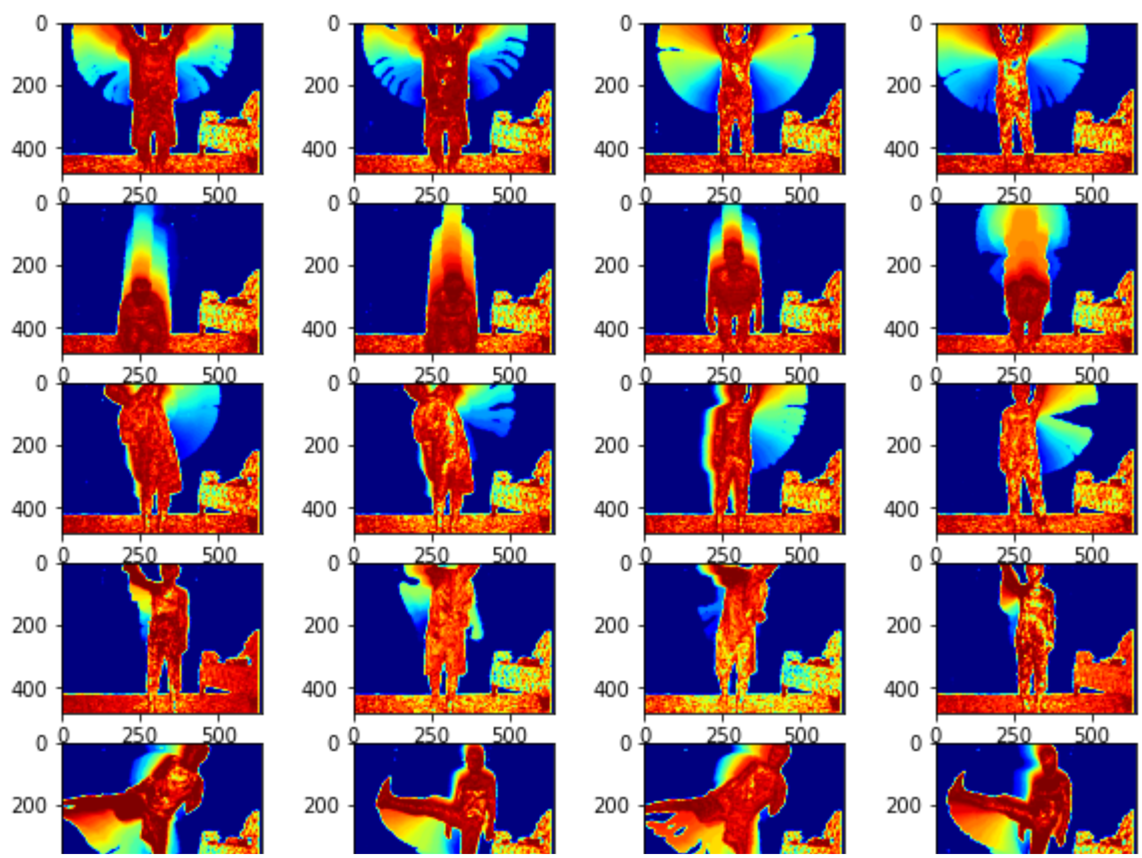
- ☐ Manually add `background_subtraction` , `compute_two_frame_difference` , `compute_motion_history_image` , and the necessary imports to `ps5.py` .
- ☐ Submit `all_MHIs.npy` and `ps5.py` on Gradescope.
- ☐ Display the 5x4 grid of MHIs on your answer sheet.

In [32]:
```
##############################################################################
# TODO: Compute MHI for all the sequences and save them in allMHIs.npy
#       Use `matplotlib.pyplot.subplots()` to display a 5x4 grid of al
#       the MHI images.
##############################################################################
```

```
h = 5
w = 4
fig, axs = plt.subplots(5, 4, figsize=(2 * h, 2 * w))
# fig, axs = plt.subplots(5, 10, figsize=(16, 8))
# for ax in axs.flat:
#     ax.axis("off")

ROOT = "PS5_Data/"  # TODO: update `ROOT` to point to the "PS5_Data" f
ACTIONS = ["botharms", "crouch", "leftarmup", "punch", "rightkick"]
# result = np.empty(shape=(w,h))
result = None
k = 0
for action in ACTIONS:
    folders = glob.glob(os.path.join(ROOT, action, "*"))
    for folder in folders:
        files = sorted(glob.glob(os.path.join(folder, "*.pgm")))
        print(f"Sequence: {folder}")
        print(f"Sequence contains {len(files)} images")
        images = [np.array(imread(path)) for path in files]
        MHI = compute_motion_history_image(images, background_subtract
                                           difference_threshold=0)
#         axs[i][j].imshow(MHI, cmap = "jet")
        if result is None:
            result = MHI
        else:
            result = np.dstack((result, MHI))
        name = "gen" +folder + ".png"
        img = MHI * 255
        img = img.astype("uint8")
        imsave(name, img)
#         plt.imshow(img, cmap = "jet")
#         plt.show()
        j = int(k/4)
        i = k - j*4
        k=k+1
        axs[j][i].imshow(img,cmap = "jet")
np.save("all_MHIs.npy", result)
# fig, axs = plt.subplots(5, 10, figsize=(16, 8))
# for i in range(5):
#     for j in range(10):
#         axs[i][j].imshow(patches[word_idx][i * 5 + j][0],cmap ="gray
# fig.savefig(f'gen/word_{word_idx + 1}_patches.png')


#####################################################################
#                              END OF YOUR CODE
#####################################################################
```

```
Sequence: PS5_Data/botharms/botharms-up-p2-2
Sequence contains 43 images
Sequence: PS5_Data/botharms/botharms-up-p2-1
Sequence contains 31 images
Sequence: PS5_Data/botharms/botharms-up-p1-1
Sequence contains 77 images
Sequence: PS5_Data/botharms/botharms-up-p1-2
Sequence contains 66 images
Sequence: PS5_Data/crouch/crouch-p1-2
```

```
Sequence contains 55 images
Sequence: PS5_Data/crouch/crouch-p1-1
Sequence contains 54 images
Sequence: PS5_Data/crouch/crouch-p2-2
Sequence contains 52 images
Sequence: PS5_Data/crouch/crouch-p2-1
Sequence contains 38 images
Sequence: PS5_Data/leftarmup/leftarm-up-p2-1
Sequence contains 61 images
Sequence: PS5_Data/leftarmup/leftarm-up-p2-2
Sequence contains 33 images
Sequence: PS5_Data/leftarmup/leftarm-up-p1-1
Sequence contains 48 images
Sequence: PS5_Data/leftarmup/leftarm-up-p1-2
Sequence contains 53 images
Sequence: PS5_Data/punch/punch-p1-2
Sequence contains 25 images
Sequence: PS5_Data/punch/punch-p2-1
Sequence contains 27 images
Sequence: PS5_Data/punch/punch-p2-2
Sequence contains 17 images
Sequence: PS5_Data/punch/punch-p1-1
Sequence contains 29 images
Sequence: PS5_Data/rightkick/rightkick-p2-2
Sequence contains 41 images
Sequence: PS5_Data/rightkick/rightkick-p1-1
Sequence contains 49 images
Sequence: PS5_Data/rightkick/rightkick-p2-1
Sequence contains 41 images
Sequence: PS5_Data/rightkick/rightkick-p1-2
Sequence contains 49 images
```



## Part 2. Hu Moments (15 pts)

Complete the function `compute_hu_moments` that takes a Motion History Image as input and returns a vector of length 7 containing the [Hu moments (https://gatech.box.com/s/tykh1w2svotcv1hniy8newzhcfq3z6co)](https://gatech.box.com/s/tykh1w2svotcv1hniy8newzhcfq3z6co) [(Normalized Variant) (https://en.wikipedia.org/wiki/Image_moment)](https://en.wikipedia.org/wiki/Image_moment).

In [11]:
```
#######################################################################
# Implement any additional helper functions (if needed) here.
#######################################################################


#######################################################################
#                              END OF YOUR CODE
#######################################################################


def compute_hu_moments(MHI):
    """
    Calculate the 7 Hu Moments of a Motion History Image

    Inputs:
    - MHI: HxW matrix - the Motion History Image

    Returns:
    - hm: A 1d array of length 7 containing the Hu Moments
    """
    hm = np.zeros(7)
    ###################################################################
    # TODO: Compute Hu Moments for the given Motion History Image (H)
    ###################################################################
    hm = np.zeros(7)

    x, y = MHI.shape
    sum_y_axis = np.sum(MHI, axis=1)
    sum_x_axis = np.sum(MHI, axis=0)
    M00 = np.sum(MHI)
    M10 = np.sum(sum_y_axis * list(range(x)))
    M01 = np.sum(sum_x_axis * list(range(y)))
    x_bar = M10 / M00
    y_bar = M01 / M00

    x_temp = [X - x_bar for X in list(range(x))]
    x_temp = np.array(x_temp)

    y_temp = [Y - y_bar for Y in list(range(y))]
    y_temp = np.array(y_temp)

    mu20 = np.sum(sum_y_axis * pow(x_temp, 2))  # good
    mu02 = np.sum(sum_x_axis * pow(y_temp, 2))  # good

    mu11 = np.sum(x_temp * np.sum(y_temp * MHI, axis=1))  # good
```

```
    mu30 = np.sum(sum_y_axis * pow(x_temp, 3))

    mu12 = np.sum(x_temp * np.sum(pow(y_temp, 2) * MHI, axis=1))

    mu21 = np.sum(pow(x_temp, 2) * np.sum(y_temp * MHI, axis=1))

    mu03 = np.sum(sum_x_axis * pow(y_temp, 3))

    mu20 = mu20/pow(M00,2)
    mu02 = mu02/pow(M00,2)

    mu11 = mu11/pow(M00,2)

    mu30 = mu30/pow(M00,2.5)
    mu03 = mu03/pow(M00,2.5)

    mu12 = mu12/pow(M00,2.5)
    mu21 = mu21/pow(M00,2.5)

    hm[0] = mu02 + mu20
    hm[1] = pow((mu02 - mu20), 2) + 4 * pow(mu11, 2)

    hm[2] = pow((mu30 - 3 * mu12), 2) + pow((3 * mu21 - mu03), 2)

    hm[3] = pow((mu30 + mu12), 2) + pow((mu03 + mu21), 2)

    hm[4] = (mu30 - 3 * mu12) * (mu30 + mu12) * (pow((mu30 + mu12), 2)
                3 * mu21 - mu03) * (mu21 + mu03) * (3 * pow((mu30 + mu
    hm[5] = (mu20 - mu02) * (pow((mu30 + mu12), 2) - pow((mu03 + mu21)
    hm[6] = (3 * mu21 - mu03) * (mu30 + mu12) * (pow((mu30 + mu12), 2)
                mu30 - 3 * mu12) * (mu21 + mu03) * (3 * pow((mu30 + mu

    ####################################################################
    #                              END OF YOUR CODE
    ####################################################################

    return hm
```

Let's do a sanity check to test your Hu Moments implementation for any errors.

In [12]:

```
input_image = np.array([[0.,1.,2.],[0.,1.,2.],[0.,1.,2.]])
expected_hu = np.array([ 8.00000000e+00,  1.60000000e+01,  4.44444444e
  1.97530864e-01, -1.77777778e+00, -6.77927340e-32])
expected_hu_normalized = np.array([ 9.87654321e-02,  2.43865264e-03,
  5.66512986e-11, -3.71689170e-07, -1.86540852e-41])

def relative_error(expected, actual):
  return np.mean(np.abs((expected-actual)/np.maximum(expected,1.0)))

output = compute_hu_moments(input_image)
error = min(relative_error(expected_hu, output), relative_error(expect
print("Relative error: ",error)
if error < 1e-4:
  print("Sanity Check passed.")
else:
```

```
    print("Sanity Check failed.")
    print("Expected Output: \n", expected_hu, "\n or \n", expected_hu_no
    print("Actual Output: ", output)
```

```
Relative error:  8.076867188233254e-13
Sanity Check passed.
```

Please save and submit the Hu moments vectors of all the sequences in a file called `hu_vectors.npy`. This file should contain a matrix of size 20x7.

**Deliverables:**

- [ ] Manually add `compute_hu_moments` and the necessary imports and helper functions to `ps5.py`.
- [ ] Submit `hu_vectors.npy` and `ps5.py` on Gradescope.

In [13]:

```
##################################################################
# TODO: Compute Hu Moments for all the MHIs from all_MHIs.npy
#       and save them in hu_vectors.npy
##################################################################
##################################################################
allMHIs = np.load("all_MHIs.npy")
huVectors = None
for i in range (allMHIs.shape[2]):
    h = np.asarray(compute_hu_moments(allMHIs[:][:][i]))
    print(h)
    if huVectors is None:
        huVectors = h
    else:
        huVectors = np.vstack((huVectors, h))
    print(huVectors.shape)
np.save("hu_Vectors.npy", huVectors)

##################################################################
#                              END OF YOUR CODE
##################################################################
```

```
[ 4.01864335 15.99926193  5.49228789  5.21341105 27.89694743 20.67832
955
 -0.10220972]
(7,)
[ 3.94549559 15.42072965  4.90580383  4.64033516 22.13989019 18.05608
152
 -0.0907754 ]
(2, 7)
[ 3.92507749 15.26225447  4.93762639  4.67335278 22.44907341 18.09350
423
 -0.09065755]
(3, 7)
[ 3.92050001 15.22737558  4.84667824  4.58053296 21.58202712 17.70796
845
 -0.09122317]
(4, 7)
[ 3.89607853 15.03864776  4.84624252  4.58491386 21.61203665 17.61965
083
 -0.08857458]
(5, 7)
[ 3.85207855 14.70019183  4.56532705  4.31398272 19.14474783 16.38809
122
 -0.08113081]
(6, 7)
[ 3.83805887 14.59336474  4.56989646  4.31665855 19.17214542 16.33732
456
 -0.08239388]
(7, 7)
[ 3.81270639 14.40099304  4.53113497  4.2804065  18.85067981 16.09325
362
 -0.08095745]
(8, 7)
[ 3.78543913 14.19583657  4.50961504  4.26479822 18.70310382 15.92399
197
 -0.07806844]
(9, 7)
[ 3.76807136 14.06551076  4.53727788  4.29375965 18.951807   15.96108
267
 -0.0778063 ]
(10, 7)
[ 3.71929325 13.70327158  4.43620007  4.19966217 18.12689775 15.41095
147
 -0.07392706]
(11, 7)
[ 3.67947477 13.41063512  4.3413348   4.11261845 17.37744007 14.93241
964
 -0.06945607]
(12, 7)
[ 3.6669024  13.31894514  4.18739093  3.9617578  16.13614812 14.33170
56
 -0.06718969]
(13, 7)
[ 3.60332206 12.8599288   4.09124916  3.87266262 15.414831   13.76712
234
 -0.06356906]
(14, 7)
```

```
[ 3.5421779  12.42657448  4.17678608  3.96230354 16.11905051 13.85234
311
 -0.06258507]
(15, 7)
[ 3.50815297 12.18856203  4.0240047   3.81666527 14.95726044 13.21440
```

## Part 3. Predict Action (20 pts)

Complete the functions `normalized_euclidean_distance` and `predict_action` as described below.

In [14]:
```
######################################################################
# Implement any additional helper functions (if needed) here
######################################################################

######################################################################
#                              END OF YOUR CODE
######################################################################

def normalized_euclidean_distance(x, c, var):
    """
    Returns the Euclidean distance normalized with the variance of the

    Inputs:
    - x: 7 dimensional vector containing the testMoment
    - c: A matrix of shape (N, 7) containing the trainMoments
    - var: 7 dimensional vector containing the variance of each Hu mom

    Outputs:
    - dists: A matrix of shape (N, 1) containing the normalized euclid
      each trainMoment from the testMoment
    """
    N = c.shape[0]
    dists = np.zeros((N, 1), dtype="float64")
    ##################################################################
    # TODO: Add your code here
    ##################################################################
    for i in range(N):
        dists[i] = pow(np.sum((c[i] - x)**2 / var), 0.5)
    ##################################################################
    #                              END OF YOUR CODE
    ##################################################################
    return dists


def predict_action(testMoment, trainMoments, trainLabels):
    """
    Predict the action label for testMoment by using
    nearest neighbour classification on trainMoments and trainLabels.

    Steps:
    - Calculate the variance of each of the 7 Hu Moments across the en
    - Use your `normalized_euclidean_distance` function to calculate t
      between the testMoment from all the trainMoments.
```

```
    - Return the label of the training data point in trainMoments that

    Note: This function can be used to perform leave-one-out cross val

    Inputs:
    - testMoment: 7 dimensional Hu moment decriptor representing the T
    - trainMoments: A matrix of shape (N, 7) containing Hu moment desc
    - trainLabels: A vector of shape (N, 1) containing the action cate

    Returns:
    - predictedLabel: An integer from 0 to 4 denoting the predicted ac
    """
    predictedLabel = 0
    ##################################################################
    # TODO: Using nearest neighbours predict the action label for test
    ##################################################################
    n = len(testMoment)
    data = np.concatenate((np.reshape(testMoment, (1, n)), trainMoment
    var = variance(data)
    dists = normalized_euclidean_distance(testMoment, trainMoments, va
    predictedLabel = np.argmin(dists)
    ##################################################################
    #                              END OF YOUR CODE
    ##################################################################
    return predictedLabel
```

## Part 4. Show Nearest MHIs (20 pts)

Write a script below that displays the top K most similar Motion History Images to an input test example, based on the normalized Euclidean distance between their associated Hu moment descriptors. (Note that you display MHIs but still refer to distance in terms of the videos' Hu moment vectors.)

Also, display the results for two selected test examples (again, from different action categories), for K = 4.

**Deliverables:**

- ☐ Display 2 different test examples alongwith the top 4 most similar MHIs to each of them in your answer sheet.

In [15]:
```
import math
```

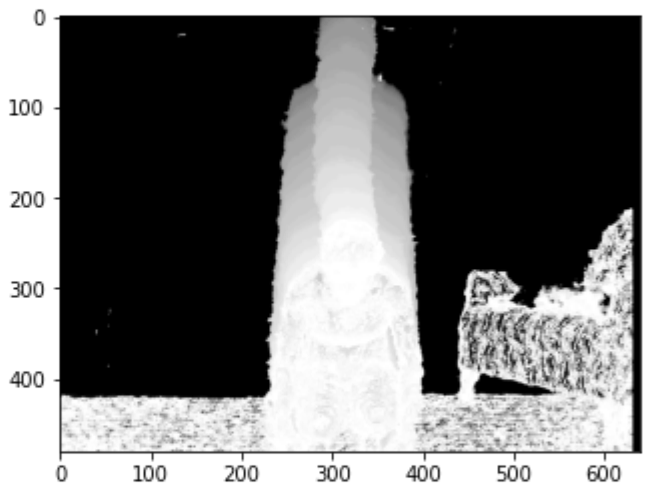
In [16]:
```
def normalizedED(testMoments, trainMoments, trainLabels):
    distances = []
    variances = np.var(trainMoments, axis=0)
    for j in range(trainMoments.shape[0]):
        distance = 0
        for i in range(trainMoments.shape[1]):
            distance += ((testMoments[i] - trainMoments[j][i]) ** 2) /
```

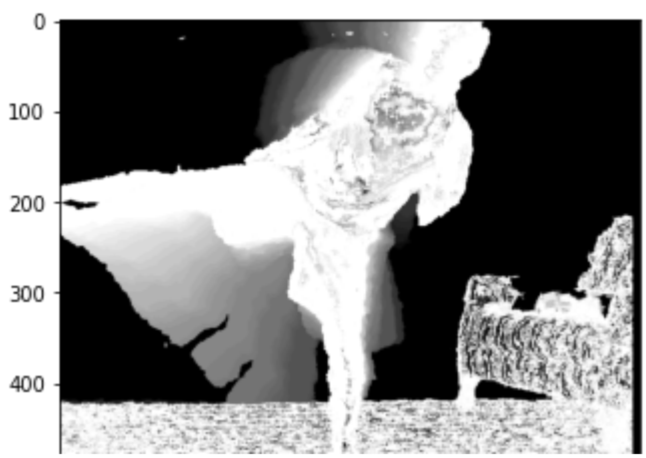
```
            distances.append((math.sqrt(distance), j))
    distances.sort(key=lambda x: x[0])
    temp = []
    for dist in distances:
        temp.append(dist[1])
#     print("st", temp, trainLabels)
    return distances
```

In [17]:

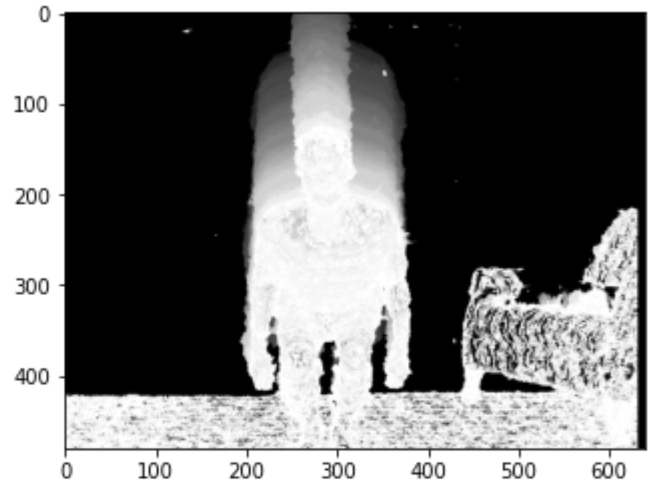
```
def showNearestMHI(testMoments, trainMoments, trainLabels, allMHIs, k,
    distances = normalizedED(testMoments, trainMoments, trainLabels)
#     print(distances)
    for i in range(k):
        idx = distances[i][1]
        img = allMHIs[:,:,idx]
        plt.imshow(img, cmap='gray')
        name = "./" + str(temp) + "_" + str(i) + ".png"
        temp = img * 255
        temp = temp.astype("uint8")
        imsave(name, temp)
        plt.show()
```

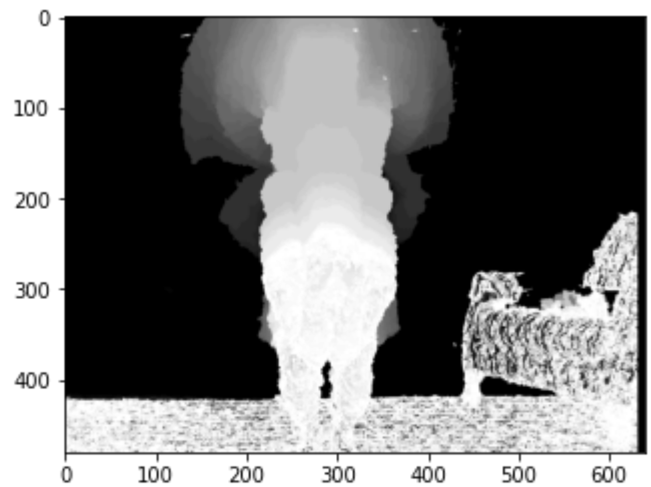
In [18]:
```
##################################################################
# TODO: Compute the top K most similar MHIs for an input test example.
#       Display the results for 2 test examples from different action
#       categories and K = 4.
##################################################################
actions = ['botharms', 'crouch', 'leftarmup', 'punch', 'rightkick']
trainLabelsAll = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
huVectors = np.load("hu_Vectors.npy")
overallMHIs = np.load("all_MHIs.npy")
for idx in [5, 16]:
    img = overallMHIs[:,:,idx]
    plt.imshow(img, cmap='gray')
    name = "./" + str(idx) + ".png"
    temp = img * 255
    temp = temp.astype("uint8")
    imsave(name, temp)
    plt.show()
print("starting")
for i in [5, 16]:
    trainMoments = np.delete(huVectors, i, 0)
    trainLabels = np.delete(trainLabelsAll, i, 0)
    testMoments = huVectors[i]
    allMHIs = np.delete(overallMHIs, i, 2)
    showNearestMHI(testMoments, trainMoments, trainLabels, allMHIs, 4,
##################################################################
#                              END OF YOUR CODE
##################################################################
```

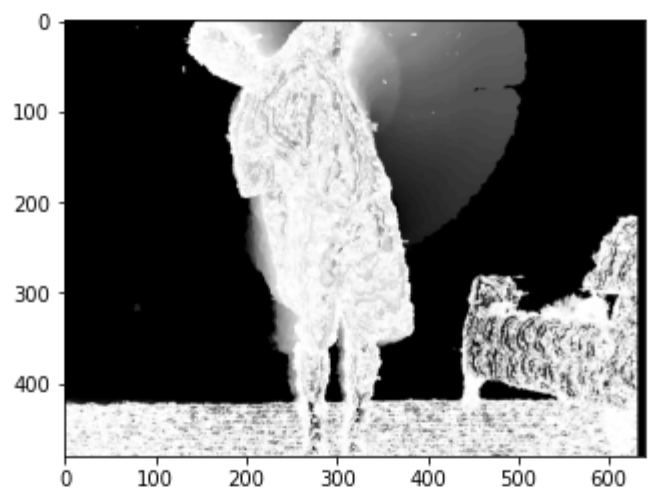


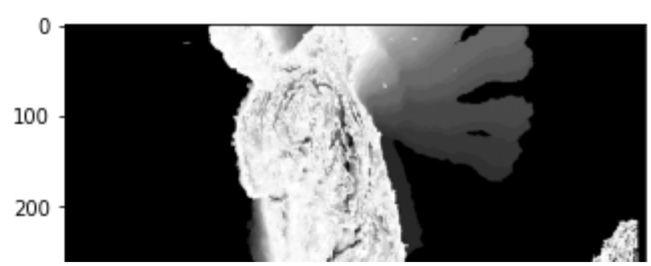



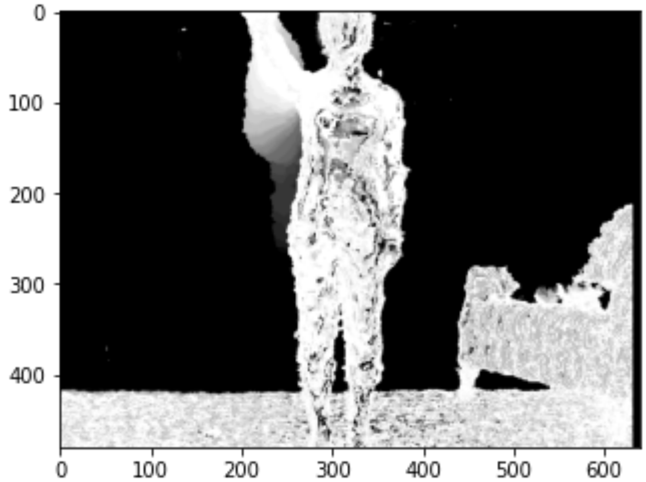
```
starting
```

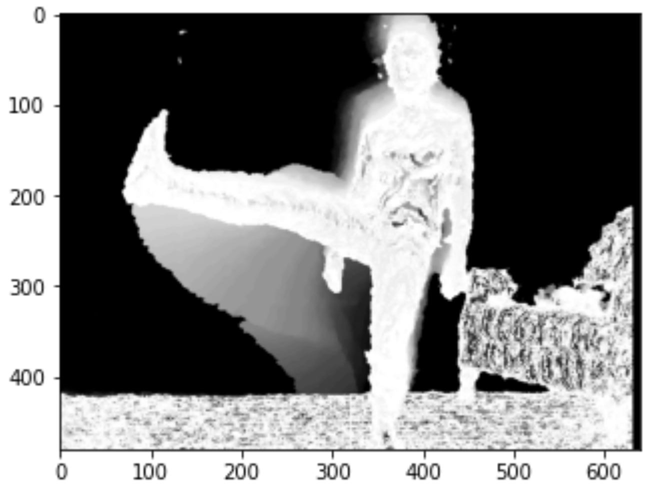


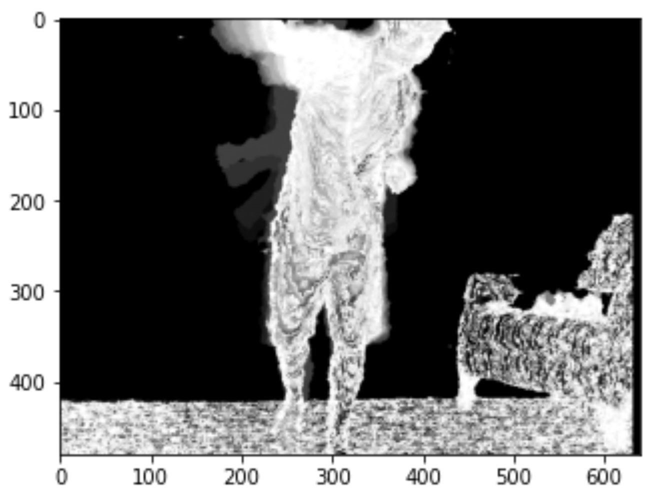


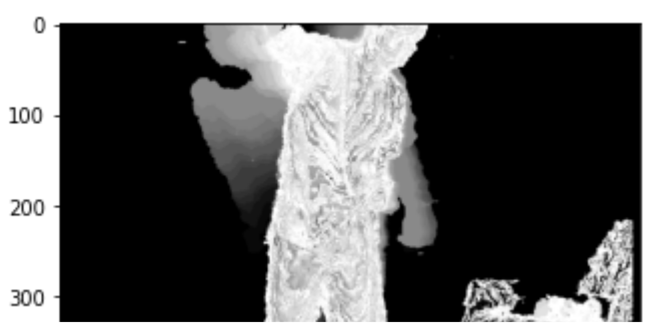





0
100
200
300
400
0 100 200 300 400 500 600


0
100
200
300
400
0 100 200 300 400 500 600


0
100
200
300
400
0 100 200 300 400 500 600


0
100
200
300

# Part 5. Classify All Actions (15 pts)

Use the nearest neighbour recognition function `predict_action` to perform *leave-one-out cross validation* for all the provided videos.

Also, calculate the Overall recognition rate, Mean recognition rate per class and display a 5x5 confusion matrix.

**Deliverables:**

- [ ] Report the following in your answer sheet:

```
1. Overall Recognition Rate
2. Mean Recognition Rate Per Class
3. Display a 5x5 Confusion matrix
4. Discuss the performance and the most confused classes
```

In [19]:

```
import statistics as st
```

In [20]:

```
def predictAction(testMoments, trainMoments, trainLabels):
    return trainLabels[normalizedED(testMoments, trainMoments, trainLa
```

In [45]:

```
###########################################################################
# TODO: Perform leave-one-out cross validation on all the provided seq
#       Report the overall recognition rate, mean recognition rate per
#       and display a 5x5 confusion matrix.
###########################################################################
actions = ['botharms', 'crouch', 'leftarmup', 'punch', 'rightkick']
trainLabelsAll = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
means = {}
total = {}
for trainLabel in trainLabelsAll:
    means[trainLabel] = 0
    if trainLabel not in total:
        total[trainLabel] = 1
    else:
        total[trainLabel] += 1
huVectors = np.load("hu_Vectors.npy")
confusionMat = np.zeros((len(actions), len(actions)), dtype=int)
for i in range(huVectors.shape[0]):
    trainMoments = np.delete(huVectors, i, 0)
    trainLabels = np.delete(trainLabelsAll, i, 0)
    testMoments = huVectors[i]
    res = predictAction(testMoments, trainMoments, trainLabels)
    # res = predictActionEC(testMoments, trainMoments, trainLabels)
    # res = predictActionMean(testMoments, trainMoments, trainLabels)
#     res = predictActionMode(testMoments, trainMoments, trainLabels)
#     print(res)
    confusionMat[trainLabelsAll[i]-1][res-1] += 1
    if res == trainLabelsAll[i]:
```

```
        means[res] += 1

s = 0
print("Mean Recognition Rate Per Class")
for key in means.keys():
    means[key] /= total[key]
    print(actions[key-1], " : ", means[key])
    s = s + means[key]
print("Overall Recognition Rate")
print(s/5)
fig, axs =plt.subplots()
axs.axis('tight')
axs.axis('off')
the_table = axs.table(cellText=confusionMat,colLabels=actions, rowLabe
plt.show()

######################################################################
#                              END OF YOUR CODE
######################################################################
```

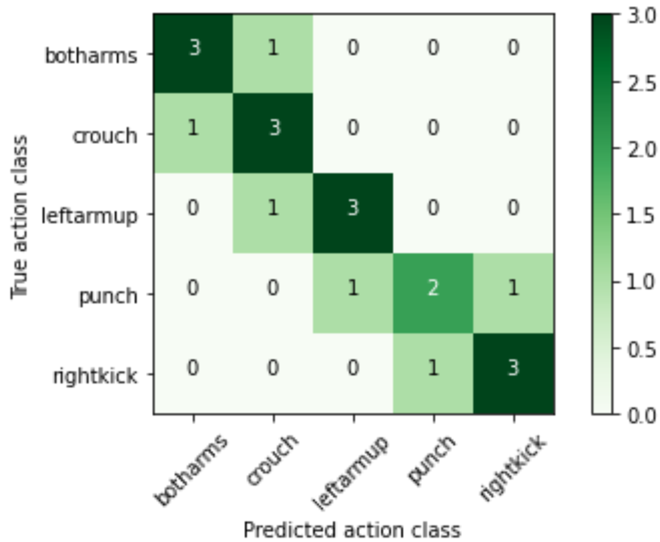
```
Mean Recognition Rate Per Class
botharms  :  0.75
crouch  :  0.75
leftarmup  :  1.0
punch  :  0.5
rightkick  :  0.75
Overall Recognition Rate
0.75
```

| | botharms | crouch | leftarmup | punch | rightkick |
|---|---|---|---|---|---|
| botharms | 3 | 1 | 0 | 0 | 0 |
| crouch | 1 | 3 | 0 | 0 | 0 |
| leftarmup | 0 | 0 | 4 | 0 | 0 |
| punch | 0 | 0 | 1 | 2 | 1 |
| rightkick | 0 | 0 | 0 | 1 | 3 |

In [46]:
```
import itertools
```

In [61]:
```
plt.imshow(confusionMat, interpolation='nearest', cmap = plt.cm.Greens
# plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, actions, rotation=45)
plt.yticks(tick_marks, actions)

fmt = '.2f' if normalize else 'd'
t = confusionMat.max() / 2.
for i, j in itertools.product(range(confusionMat.shape[0]), range(conf
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
```

```
                color="white" if confusionMat[i, j] > t else "black")

plt.ylabel('True action class')
plt.xlabel('Predicted action class')
plt.tight_layout()
plt.show()
```




# OPTIONAL: Extra credit (max 20 points)

Using ideas from any previous lectures, enhance the approach to try and improve the recognition results. For example, you might incorporate a different classifier, distance function, or descriptor. You might exploit the depth map for more than simply background subtraction, enhance the silhouette computation,...

**Deliverables:**

- Report the following in your answer sheet:

```
 1. How have you extended the method?
 2. What results did you get?
 3. How are they different from the results you got befor
e? Why are they different?
```

In [73]:
```
def normalizedEDEC(testMoments, trainMoments, trainLabels):
    distances = []
    variances = np.square(np.var(trainMoments, axis=0))
    for j in range(trainMoments.shape[0]):
        distance = 0
        for i in range(trainMoments.shape[1]):
            distance += ((testMoments[i] - trainMoments[j][i]) ** 2) /
        distances.append((math.sqrt(distance), j))
    distances.sort(key=lambda x: x[0])
    temp = []
    for dist in distances:
```

```
        temp.append(dist[1])
#     print("st", temp, trainLabels)
    return distances
```

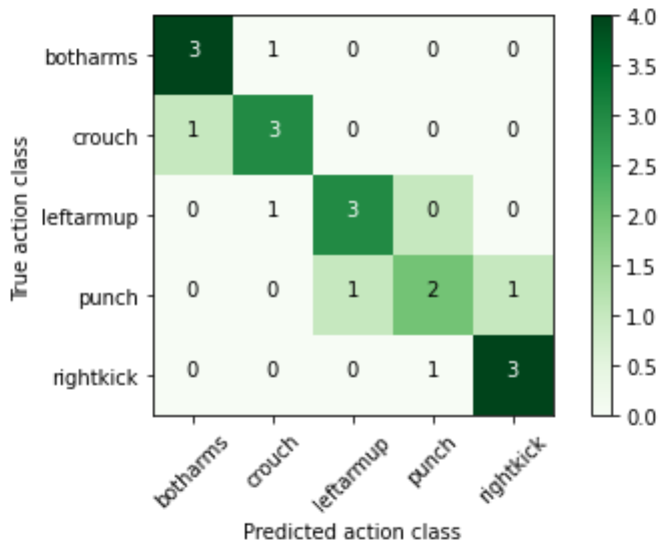
In [21]:
```
def predictActionEC(testMoments, trainMoments, trainLabels):
    return trainLabels[normalizedEDEC(testMoments, trainMoments, train
```

In [74]:
```
actions = ['botharms', 'crouch', 'leftarmup', 'punch', 'rightkick']
trainLabelsAll = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
means = {}
total = {}
for trainLabel in trainLabelsAll:
    means[trainLabel] = 0
    if trainLabel not in total:
        total[trainLabel] = 1
    else:
        total[trainLabel] += 1
huVectors = np.load("hu_Vectors.npy")
confusionMat = np.zeros((len(actions), len(actions)), dtype=int)
for i in range(huVectors.shape[0]):
    trainMoments = np.delete(huVectors, i, 0)
    trainLabels = np.delete(trainLabelsAll, i, 0)
    testMoments = huVectors[i]
#     res = predictAction(testMoments, trainMoments, trainLabels)
    res = predictActionEC(testMoments, trainMoments, trainLabels)
    # res = predictActionMean(testMoments, trainMoments, trainLabels)
#     res = predictActionMode(testMoments, trainMoments, trainLabels)
#     print(res)
    confusionMat[trainLabelsAll[i]-1][res-1] += 1
    if res == trainLabelsAll[i]:
        means[res] += 1

s = 0
print("Mean Recognition Rate Per Class")
for key in means.keys():
    means[key] /= total[key]
    print(actions[key-1], " : ", means[key])
    s = s + means[key]
print("")
print("Overall Recognition Rate")
print(s/5)
fig, axs =plt.subplots()
axs.axis('tight')
axs.axis('off')
the_table = axs.table(cellText=confusionMat,colLabels=actions, rowLabe
plt.show()

##########################################################################
#                                END OF YOUR CODE
##########################################################################
```

```
Mean Recognition Rate Per Class
botharms  :  1.0
```

| | botharms | crouch | leftarmup | punch | rightkick |
|---|---|---|---|---|---|
| botharms | 4 | 0 | 0 | 0 | 0 |
| crouch | 1 | 3 | 0 | 0 | 0 |
| leftarmup | 0 | 0 | 3 | 1 | 0 |
| punch | 0 | 0 | 1 | 2 | 1 |
| rightkick | 0 | 0 | 0 | 0 | 4 |

In [75]:
```python
plt.imshow(confusionMat, interpolation='nearest', cmap = plt.cm.Greens
# plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, actions, rotation=45)
plt.yticks(tick_marks, actions)

fmt = '.2f' if normalize else 'd'
t = confusionMat.max() / 2.
for i, j in itertools.product(range(confusionMat.shape[0]), range(conf
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusionMat[i, j] > t else "black")

plt.ylabel('True action class')
plt.xlabel('Predicted action class')
plt.tight_layout()
plt.show()
```




In [72]:
```python
def predictActionMean(testMoments, trainMoments, trainLabels):
    k = 4
```

```
    distances = normalizedED(testMoments, trainMoments, trainLabels)
    mean = 0
    labels = []
    for i in range(k):
        mean+= trainLabels[distances[i][1]]
        labels.append(trainLabels[distances[i][1]])
#     print(labels)
    mean = int(round(mean/k))
    if mean > 5:
        return 5
    elif mean < 1:
        return 1
    else:
        return mean
```

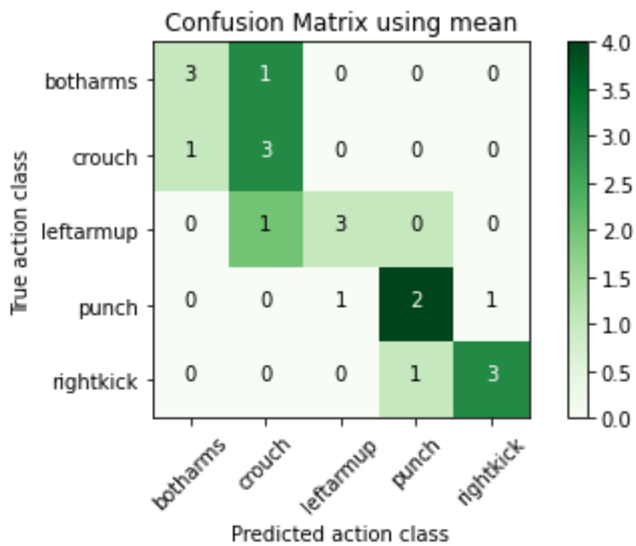
In [76]:

```
actions = ['botharms', 'crouch', 'leftarmup', 'punch', 'rightkick']
trainLabelsAll = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
means = {}
total = {}
for trainLabel in trainLabelsAll:
    means[trainLabel] = 0
    if trainLabel not in total:
        total[trainLabel] = 1
    else:
        total[trainLabel] += 1
huVectors = np.load("hu_Vectors.npy")
confusionMat = np.zeros((len(actions), len(actions)), dtype=int)
for i in range(huVectors.shape[0]):
    trainMoments = np.delete(huVectors, i, 0)
    trainLabels = np.delete(trainLabelsAll, i, 0)
    testMoments = huVectors[i]
#     res = predictAction(testMoments, trainMoments, trainLabels)
#     res = predictActionEC(testMoments, trainMoments, trainLabels)
    res = predictActionMean(testMoments, trainMoments, trainLabels)
#     res = predictActionMode(testMoments, trainMoments, trainLabels)
#     print(res)
    confusionMat[trainLabelsAll[i]-1][res-1] += 1
    if res == trainLabelsAll[i]:
        means[res] += 1

s = 0
print("Mean Recognition Rate Per Class")
for key in means.keys():
    means[key] /= total[key]
    print(actions[key-1], " : ", means[key])
    s = s + means[key]
print("Overall Recognition Rate")
print(s/5)
fig, axs =plt.subplots()
axs.axis('tight')
axs.axis('off')
the_table = axs.table(cellText=confusionMat,colLabels=actions, rowLabe
plt.show()

######################################################################
#                          END OF YOUR CODE
######################################################################
```

```
Mean Recognition Rate Per Class
botharms  :  0.25
crouch  :  0.75
leftarmup  :  0.25
punch  :  1.0
rightkick  :  0.75
Overall Recognition Rate
0.6
```

| | botharms | crouch | leftarmup | punch | rightkick |
|---|---|---|---|---|---|
| botharms | 1 | 3 | 0 | 0 | 0 |
| crouch | 1 | 3 | 0 | 0 | 0 |
| leftarmup | 0 | 2 | 1 | 1 | 0 |
| punch | 0 | 0 | 0 | 4 | 0 |
| rightkick | 0 | 0 | 0 | 1 | 3 |

In [77]:
```
plt.imshow(confusionMat, interpolation='nearest', cmap = plt.cm.Greens
# plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, actions, rotation=45)
plt.yticks(tick_marks, actions)

fmt = '.2f' if normalize else 'd'
t = confusionMat.max() / 2.
for i, j in itertools.product(range(confusionMat.shape[0]), range(conf
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusionMat[i, j] > t else "black")

plt.ylabel('True action class')
plt.xlabel('Predicted action class')
plt.title('Confusion Matrix using mean')
plt.tight_layout()
plt.show()
```




In [80]:
```
def predictActionMode(testMoments, trainMoments, trainLabels):
    k = 4
    distances = normalizedED(testMoments, trainMoments, trainLabels)
    labels = []
    for i in range(k):
        labels.append(trainLabels[distances[i][1]])
    try:
        label = st.mode(labels)
    except st.StatisticsError:
        label = labels[0]
    return label
```

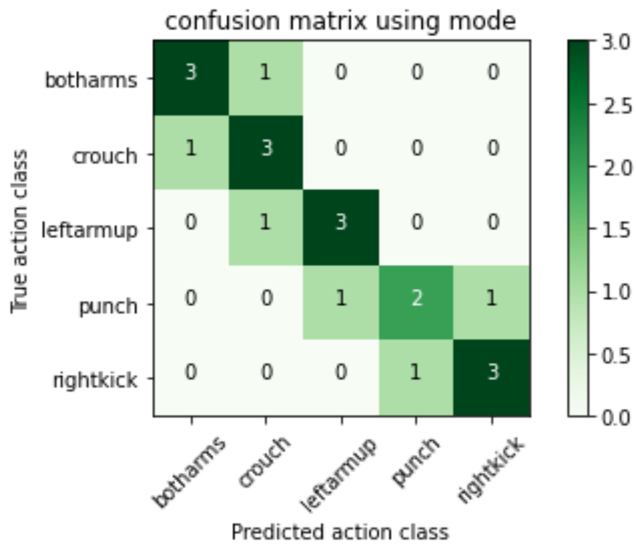
In [81]:
```
actions = ['botharms', 'crouch', 'leftarmup', 'punch', 'rightkick']
trainLabelsAll = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5
means = {}
total = {}
for trainLabel in trainLabelsAll:
    means[trainLabel] = 0
```

```
    if trainLabel not in total:
        total[trainLabel] = 1
    else:
        total[trainLabel] += 1
huVectors = np.load("hu_Vectors.npy")
confusionMat = np.zeros((len(actions), len(actions)), dtype=int)
for i in range(huVectors.shape[0]):
    trainMoments = np.delete(huVectors, i, 0)
    trainLabels = np.delete(trainLabelsAll, i, 0)
    testMoments = huVectors[i]
#     res = predictAction(testMoments, trainMoments, trainLabels)
#     res = predictActionEC(testMoments, trainMoments, trainLabels)
#     res = predictActionMean(testMoments, trainMoments, trainLabels)
    res = predictActionMode(testMoments, trainMoments, trainLabels)
#     print(res)
    confusionMat[trainLabelsAll[i]-1][res-1] += 1
    if res == trainLabelsAll[i]:
        means[res] += 1

s = 0
print("Mean Recognition Rate Per Class")
for key in means.keys():
    means[key] /= total[key]
    print(actions[key-1], " : ", means[key])
    s = s + means[key]
print("Overall Recognition Rate")
print(s/5)
fig, axs =plt.subplots()
axs.axis('tight')
axs.axis('off')
the_table = axs.table(cellText=confusionMat,colLabels=actions, rowLabe
plt.show()

######################################################################
#                              END OF YOUR CODE
######################################################################
```

```
Mean Recognition Rate Per Class
botharms  :  0.75
crouch  :  0.75
leftarmup  :  0.75
punch  :  0.5
rightkick  :  0.75
Overall Recognition Rate
0.7
```

| | botharms | crouch | leftarmup | punch | rightkick |
|---|---|---|---|---|---|
| botharms | 3 | 1 | 0 | 0 | 0 |
| crouch | 1 | 3 | 0 | 0 | 0 |
| leftarmup | 0 | 1 | 3 | 0 | 0 |
| punch | 0 | 0 | 1 | 2 | 1 |
| rightkick | 0 | 0 | 0 | 1 | 3 |

In [82]:
```
plt.imshow(confusionMat, interpolation='nearest', cmap = plt.cm.Greens
# plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, actions, rotation=45)
plt.yticks(tick_marks, actions)

fmt = '.2f' if normalize else 'd'
t = confusionMat.max() / 2.
for i, j in itertools.product(range(confusionMat.shape[0]), range(conf
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusionMat[i, j] > t else "black")

plt.ylabel('True action class')
plt.xlabel('Predicted action class')
plt.title('confusion matrix using mode')
plt.tight_layout()
plt.show()
```




In [ ]: