# Exercises for Probabilistic Graphical Models
# Sheet No.2

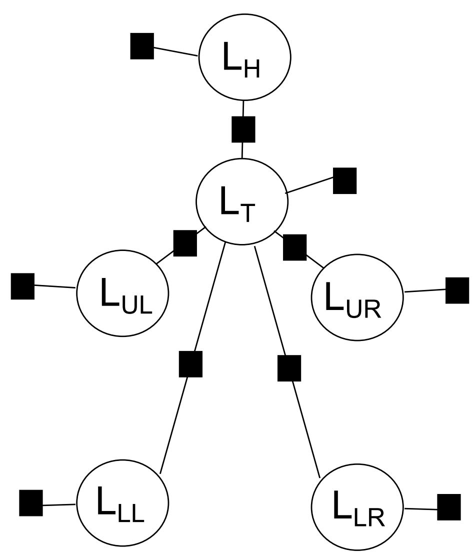
Bernt Schiele, Bjoern Andres, Eldar Insafutdinov, Evgeny Levinkov

**Due Date: December 1**
Submit to Evgeny via levinkov@mpi-inf.mpg.de.
Begin the subject of your e-mails with [PGM]. Only send the code that you have written (*.m files), no data files.

## 1 Pictorial Structures

**Points: 20**

The goal of this exercise is to implement a simplified version of the pictorial structures model, although using an efficient algorithm, and test it on a pedestrian dataset. You are going to use a star model over (upper and lower) legs, head, and torso as follows:




Each $L_i$ in this distribution is actually a two dimensional random variable representing image coordinates of the center of each body part: $L_i = (x_i, y_i)$. Scale and rotation is not considered here. Unary factors represent likelihoods

$p(e_i|L_i)$ and pairwise factors stand for kinematic priors $p(L_i, L_j)$. You will learn the priors from a training dataset. Then, you will make inference on test images, for which you are given the corresponding likelihood maps.

The first step is to download "assignment02-data.zip" from the lecture's website and import the "data.mat". Variable **likelihoods{j,i}** is the 2D likelihood map $p(e_i|L_i)$ for image number $j$. The images are to be found in directory **testset** for reference. Index $i$ is as follows:

| | |
|---|---|
| 1 | Lower leg Left |
| 2 | Upper leg Left |
| 3 | Upper leg Right |
| 4 | Lower leg Right |
| 5 | Head |
| 6 | Torso |

Variable **train** contains training data. Columns $i$ are numbered as before and every row $j$ represents a sample $L^{(j)}$.

## 2 Learning Kinematic Priors

**Points: 4**

We set the prior as a 2D Gaussian $p(L_i, L_j) \approx N(L_i - T_{ij}(L_j); 0, \Sigma_{ij})$ and we define the transformation $T_{ij}(L_j) = L_j - \mu_{ij}$. This is a reasonable approximation for small joint rotations, as it is the case for pedestrians. The parameters to be determined for each prior are thus $\Sigma_{ij}$ and $\mu_{ij}$. Write **function pairwisePotts = learnPairwisePotts(train)** which, for each body part, computes ML estimate of $\mu_{ij}$ as a row vector **pairwisePotts{i,1}** and of $\Sigma_{ij}$ as **pairwisePotts{i,2}**. Use the **mean** and **cov** matlab functions.

## 3 Maximal Marginal States

**Points: 6**

The goal is to compute maximal marginals of the model using sum-product algorithm. Write **function maxstates = sumproduct(pairwisePotts, unaryPotts)** doing this. It returns a 6x2 matrix of x,y coordinates.

- Note that the graph is not a chain but a tree, so you have to think about the correct message scheduling.
- You will need computations like $f(L_i) = \sum_{L_j} N(L_i - T_{ij}(L_j))g(L_j)$ in your code. However, summing over $L_j$ is nearly infeasible due to the huge state space. That's why one computes the sum as a convolution: $f(L_i) = \sum_s N(L_i - s)g(T_{ij}^{-1}(s)) = (N * (g \circ T_{ij}^{-1}))(L_i)$ with $s = T_{ij}(L_j)$. Moreover, one uses a diagonal covariance here so that the Gaussian is separable. Functions **fspecial**, **conv2** and the prepared file **shiftimg.m**

(it shifts a given image for a given 2D offset) will be useful here. Take care when computing messages in the opposite direction.

- You can visualize your maxima using **drawmaxima.m**. You can compare your results for the first 10 images with the official solution in directory **solution**. If your results are better than ours, let us know.
- Hint: remember that we work with $(x, y)$ vectors but Matlab indexes its matrices by (row, column).

# 4 Mode

**Points: 6**

The goal is to compute the maximum state of the joint distribution using min-sum algorithm (i.e. using negated log potentials). Write **function maxstates = minsum(pairwisePotts, unaryPotts)** doing this. It returns a 6x2 matrix of $x, y$ coordinates.

- For efficiency reasons, we compute minimizations using the generalized distance transform: $min_{L_j} -log[N(L_i - T_{ij}(L_j))] - log[g(L_j)] = min_s \delta(L_i, s) - log[g(T_{ij}^{-1}(s))] = DT(-log[g(T_{ij}^{-1}(s))])$. You can find the code in DT.m; the function takes covariance matrix of the Gaussian as the second argument.
- Unfortunately, DT doesn't give you the argmin, just the min. For this reason, you can't do backtracking and you need to implement the min-sum algorithm as in the lecture on max-sum algorithm earlier, i.e. computing all messages (that's two messages per edge) and taking a node-wise minimum. This will probably fail on potential ties (multiple modes) but that's fine in this exercise.

# 5 Evaluation

**Points: 4**

Evaluation (4 pts)

Now you are going to evaluate the model as a person detector by writing a script **evaluation.m**. To make it simple, fix a bounding box of size 80x200px around the torso (horizontally centered at it, vertically offset in 1:2 ratio). A predicted bounding box is considered correct if it overlaps more than 50% with a ground-truth bounding box, otherwise the bounding box is considered a false positive detection. Ground truth can be found as variable **GT** in the supplied mat file, each row is a rectangle $[x1, y1, w, h]$. Bounding box overlap is computed using **boxoverlap.m** function which is provided for your convinience.

Compute bounding boxes for each test image using three ways of choosing the torso:

- Torso as the result of min-sum.
- Torso as the result of max-product.
- Torso as the maximum of a torso's likelihood, which corresponds to using no model at all.

The result is the accuracy (correct:all) of each method. Note that you can visualize your bounding boxes and detections using **drawmaxima.m**.