**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Теория игр и исследование операций»**

Тема: Релаксация линейного программирования.

|  |  |  |
| --- | --- | --- |
| Студент гр. 5381 |  | Кобылянский А.В. |
| Преподаватель |  | Шолохов А.В. |

Санкт-Петербург

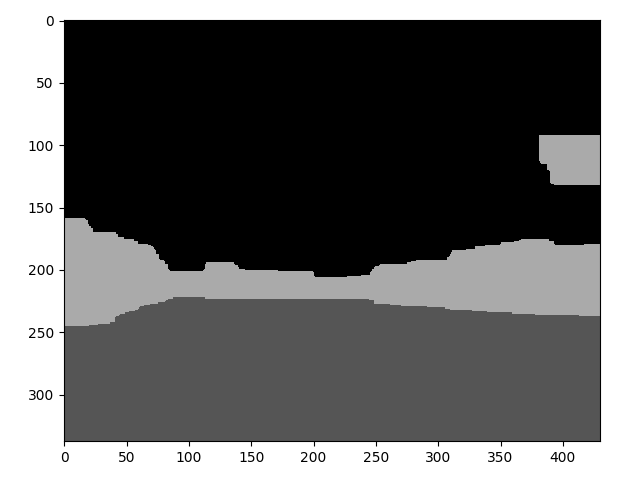
2018

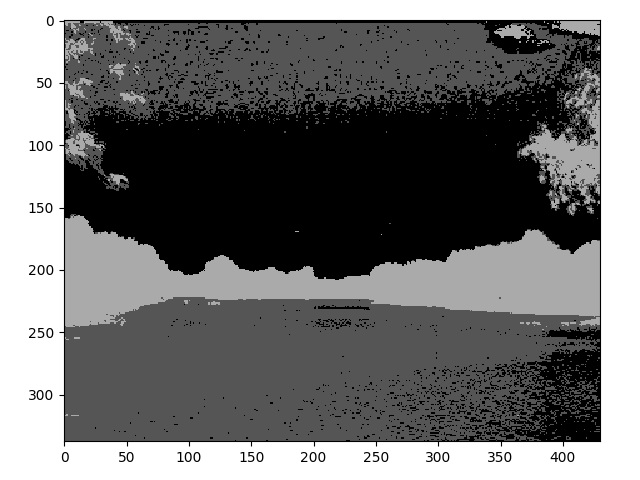
Задание 1. Сегментация изображения.

Код для решения задачи:

|  |
| --- |
| **import** numpy **as** np  **from** pystruct.inference **import** inference\_dispatch  **from** pystruct.utils **import** make\_grid\_edges  **from** scipy.io **import** loadmat  **import** matplotlib.pyplot **as** plt  **def** task1():  data = loadmat(**'data/mrf\_potentials\_segmentation.mat'**)  unary\_potentials = data.get(**'unary\_potentials'**) *# type: np.ndarray (338, 430, 3)*  pairwise\_potentials\_vert = data.get(**'pairwise\_potentials\_vert'**) *# type: np.ndarray (3, 3)*  pairwise\_potentials\_horz = data.get(**'pairwise\_potentials\_horz'**) *# type: np.ndarray (3, 3)*    edges = make\_grid\_edges(unary\_potentials) *# (289912, 2)*  height, width, n\_states = unary\_potentials.shape  n\_edges = edges.shape[0]  pairwise\_potentials = np.zeros(shape=(n\_edges, n\_states, n\_states))  **for** i, edge **in** enumerate(edges):  vert\_1, vert\_2 = edge  y1, x1 = divmod(vert\_1, width)  y2, x2 = divmod(vert\_2, width)    **if** abs(y1 - y2) == 1:  *# vertical edge*  pairwise\_potentials[i, :, :] = pairwise\_potentials\_vert  **elif** abs(x1 - x2) == 1:  *# horisontal edge*  pairwise\_potentials[i, :, :] = pairwise\_potentials\_horz    *# QPBO because max-product works slow.*  result = inference\_dispatch(-1.0\*unary\_potentials, -1.0\*pairwise\_potentials, edges, inference\_method=**'qpbo'**)    picture = result.reshape(height, width)  plt.imshow(picture, vmin=0, vmax=n\_states, cmap=**'gray'**)  plt.show()  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  task1() |

Результат:

Результат работы программы с использование только унарных потенциалов (*inference\_method*=“*unary*” в функции *inference\_dispatch*).



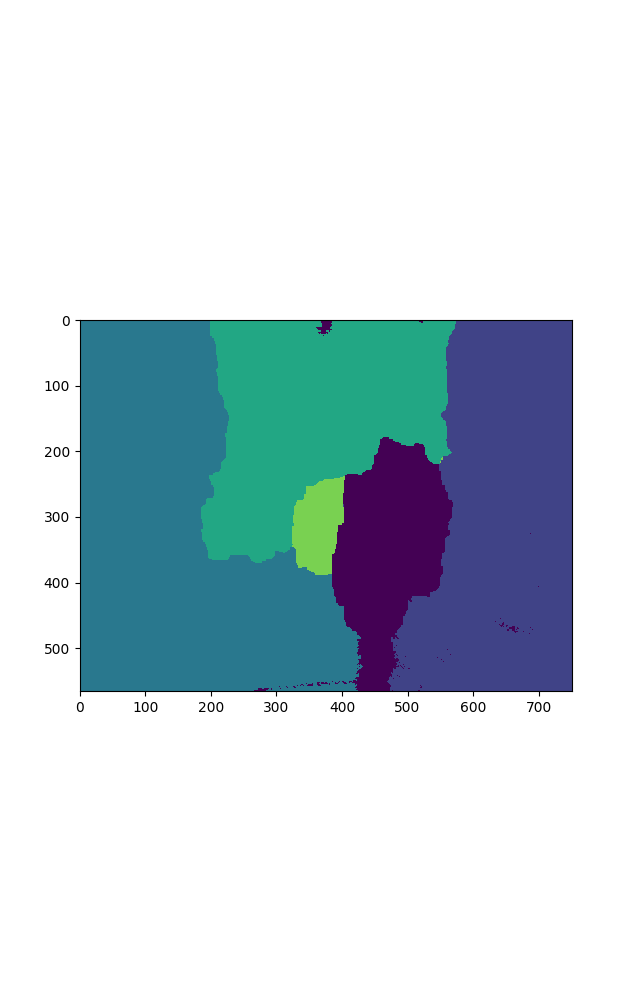
Задание 2. Склеивание изображений.

Код программы:

|  |
| --- |
| **import** imageio  **import** matplotlib.pyplot **as** plt  **import** numpy **as** np  **import** visvis **as** vv  **from** numpy.linalg **import** norm  **from** pystruct.inference **import** inference\_dispatch  **from** pystruct.utils **import** make\_grid\_edges  **class** ImagesData:  **def** \_\_init\_\_(self, images\_names, markup\_name=**None**):  self.shape = **None**  self.images = []  self.markup = **None**  self.\_\_load\_images(images\_names)  **if** markup\_name **is not None**:  self.\_\_load\_markup(markup\_name)  **def** \_\_check\_shape(self, image):  **if** self.shape **is None**:  self.shape = image.shape  **else**:  **if** self.shape != image.shape:  print(**'images shapes are not equile'**)  print(self.shape, image.shape)  exit(1)  **def** \_\_load\_images(self, images\_names):  self.images = []  **for** image\_name **in** images\_names:  self.images.append(imageio.imread(image\_name))  self.\_\_check\_shape(self.images[-1])  **def** \_\_load\_markup(self, markup\_name):  self.markup = imageio.imread(markup\_name) // 20  self.\_\_check\_shape(self.markup)  **def** \_\_scale\_image(self, image, scale):  height, width, colors = self.shape  new\_height, new\_width = height // scale, width // scale  new\_image = np.zeros(shape=(new\_height, new\_width, colors))  **for** x **in** range(new\_width):  **for** y **in** range(new\_height):  square = image[scale \* y:scale \* (y + 1), scale \* x:scale \* (x + 1)]  new\_image[y, x] = np.rint(np.mean(square, axis=(0, 1)))  **return** new\_image  **def** \_\_scale\_markup(self, markup, scale):  height, width, colors = self.shape  new\_height, new\_width = height // scale, width // scale  new\_markup = np.zeros(shape=(new\_height, new\_width, colors))  n\_states = len(self.images)  **for** x **in** range(new\_width):  **for** y **in** range(new\_height):  value = 0  count = 0  **for** i **in** range(scale \* x, scale \* (x + 1)):  **for** j **in** range(scale \* y, scale \* (y + 1)):  **if** markup[j, i, 0] < n\_states:  value = markup[j, i, 0]  count += 1  **if** count >= scale \*\* 2 / 2:  new\_markup[y, x] = np.array([value, value, value])  **else**:  new\_markup[y, x] = np.array([255 // 20, 255 // 20, 255 // 20])  **return** new\_markup  **def** scale(self, scale: int):  *# scale images and markup to 1/scale times*  *# each new pixel is mean of scale x scale square in origin image*  self.images = [self.\_\_scale\_image(image, scale) **for** image **in** self.images]  **if** self.markup **is not None**:  self.markup = self.\_\_scale\_markup(self.markup, scale)  height, width, colors = self.shape  self.shape = (height // scale, width // scale, colors)  **class** Task2:  **def** \_\_init\_\_(self, images\_data: ImagesData, unary\_potentials):  self.imgs = images\_data  self.unary\_potentials = unary\_potentials  self.\_\_result = **None**  self.path\_to\_dump\_file = **None**  self.diff\_function = diff\_standart  **def** \_\_compute\_pairwise\_potentials(self, edges):  n\_states = len(self.imgs.images)  n\_edges = edges.shape[0] *# 849946*  height, width, \_ = self.imgs.shape  pairwise\_potentials = np.zeros(shape=(n\_edges, n\_states, n\_states))    **for** i, edge **in** enumerate(edges):  **if** i % 1000 == 0:  print(**'{} / {}'**.format(i // 1000, len(edges) // 1000))  vert\_1, vert\_2 = edge  y1, x1 = divmod(vert\_1, width)  y2, x2 = divmod(vert\_2, width)  **for** state\_1 **in** range(n\_states):  **for** state\_2 **in** range(n\_states):  **if** state\_1 != state\_2:  pairwise\_potentials[i, state\_1, state\_2] = self.diff\_function(self.imgs.images, state\_1,  state\_2, x1, y1, x2, y2)  **return** pairwise\_potentials  **def** \_\_load\_pairwise\_potentials(self, path):  n\_states = len(self.imgs.images)  res = np.fromfile(path)  elements\_count = res.shape[0]  **return** res.reshape((elements\_count // n\_states \*\* 2, n\_states, n\_states))  **def** \_\_get\_regions(self):  **if** self.\_\_result **is not None**:  height, width, \_ = self.imgs.shape  **return** self.\_\_result.reshape(height, width)  **def** set\_dump\_file(self, path\_to\_dump\_file: str):  self.path\_to\_dump\_file = path\_to\_dump\_file  **def** set\_diff\_function(self, diff\_function):  self.diff\_function = diff\_function  **def** draw\_regions(self):  **if** self.\_\_result **is not None**:  regions = self.\_\_get\_regions()  n\_states = len(self.imgs.images)  plt.imshow(regions, vmin=0, vmax=n\_states)  plt.show()  **def** draw\_picture(self):  **if** self.\_\_result **is not None**:  regions = self.\_\_get\_regions()  height, width = regions.shape  picture = np.zeros(shape=(height, width, 3))  **for** x **in** range(width):  **for** y **in** range(height):  region = regions[y, x]  picture[y, x] = self.imgs.images[region][y, x]  vv.imshow(picture)  **def** get\_pairwise\_potentials(self, edges, from\_file, save\_to\_file):  **if** (from\_file **or** save\_to\_file) **and** self.path\_to\_dump\_file **is None**:  **raise** Exception(**'Path to dump file is not set'**)  **if** from\_file:  pairwise\_potentials = self.\_\_load\_pairwise\_potentials(self.path\_to\_dump\_file)  **else**:  pairwise\_potentials = self.\_\_compute\_pairwise\_potentials(edges)  **if** save\_to\_file:  pairwise\_potentials.tofile(self.path\_to\_dump\_file)  **return** pairwise\_potentials  **def** compute(self, eta, from\_file=**True**, save\_to\_file=**False**):  unary\_potentials = self.unary\_potentials  edges = make\_grid\_edges(unary\_potentials)  pairwise\_potentials = self.get\_pairwise\_potentials(edges, from\_file, save\_to\_file)  print(**'start computing'**)  self.\_\_result = inference\_dispatch(-unary\_potentials, -eta \* pairwise\_potentials, edges,  inference\_method=**'qpbo'**)  print(**'stop comuting'**)  **def** diff\_standart(images, state\_1, state\_2, x1, y1, x2, y2):  d1 = images[state\_1][y1, x1] - images[state\_2][y1, x1]  d2 = images[state\_1][y2, x2] - images[state\_2][y2, x2]  **return** norm(d1) + norm(d2)  **def** diff\_alternative(images, state\_1, state\_2, x1, y1, x2, y2):  d1 = images[state\_1][y1, x1] - images[state\_2][y1, x1]  d2 = images[state\_1][y2, x2] - images[state\_2][y2, x2]  d3 = images[state\_1][y1, x1] - images[state\_1][y2, x2]  d4 = images[state\_2][y1, x1] - images[state\_2][y2, x2]  **return** (norm(d1) + norm(d2)) / (norm(d3) + norm(d4) + 1)  **def** compute\_unary\_potentials\_family(images\_data: ImagesData):  INF = 10 \*\* 5  height, width, \_ = images\_data.shape  n\_states = len(images\_data.images)  markup = images\_data.markup  unary\_potentials = np.zeros(shape=(height, width, n\_states))  **for** y **in** range(height):  **for** x **in** range(width):  **if** markup[y, x, 0] < n\_states:  **for** state **in** range(n\_states):  **if** state != markup[y, x, 0]:  unary\_potentials[y, x, state] = INF  **return** unary\_potentials  **def** compute\_unary\_potentials\_pano(images\_data: ImagesData):  INF = 10 \*\* 5  height, width, \_ = images\_data.shape  n\_states = len(images\_data.images)  unary\_potentials = np.zeros(shape=(height, width, n\_states))  **for** y **in** range(height):  **for** x **in** range(width):  **for** state, image **in** enumerate(images\_data.images):  **if** np.all(image[y, x] == 0):  unary\_potentials[y, x, state] = INF  **return** unary\_potentials  **def** task2\_family():  images\_names = [  **'data/family/small\_DSC\_0168.png'**,  **'data/family/small\_DSC\_0170.png'**,  **'data/family/small\_DSC\_0173.png'**,  **'data/family/small\_DSC\_0174.png'**,  **'data/family/small\_DSC\_0176.png'**,  ]  markup\_name = **'data/family/familydatacost.png'**  images\_data = ImagesData(images\_names, markup\_name)  images\_data.scale(2)  unary\_potentials = compute\_unary\_potentials\_family(images\_data)  task2 = Task2(images\_data, unary\_potentials)  task2.set\_dump\_file(**'data/family\_dump'**)  task2.compute(eta=0.5, from\_file=**False**, save\_to\_file=**True**)  task2.draw\_regions()  task2.draw\_picture()  input(**'Press any key'**)  **def** task2\_pano():  images\_names = [  **'data/pano/NQIMG\_0257.PNG'**,  **'data/pano/NQIMG\_0258.PNG'**,  **'data/pano/NQIMG\_0259.PNG'**,  **'data/pano/NQIMG\_0260.PNG'**,  **'data/pano/NQIMG\_0261.PNG'**,  **'data/pano/NQIMG\_0263.PNG'**,  **'data/pano/NQIMG\_0264.PNG'**,  ]  images\_data = ImagesData(images\_names)  images\_data.scale(4)  unary\_potentials = compute\_unary\_potentials\_pano(images\_data)  task2 = Task2(images\_data, unary\_potentials)  task2.set\_dump\_file(**'data/pano\_dump2'**)  task2.set\_diff\_function(diff\_alternative)  task2.compute(eta=0.5, from\_file=**False**, save\_to\_file=**True**)  task2.draw\_regions()  task2.draw\_picture()  *# input('Press any key')*  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  *# task2\_family()*  task2\_pano() |

Семья.

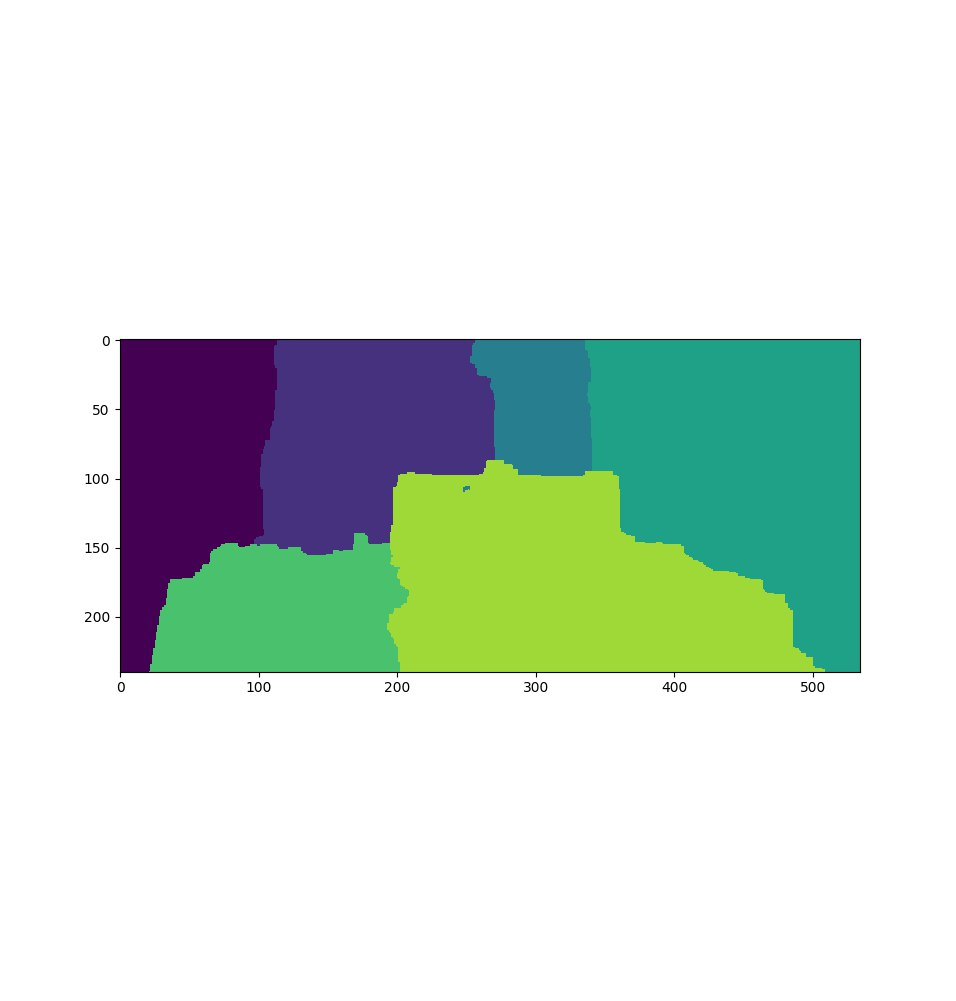
При различных разбиения различаются, но картинка всегда почти одинаковая, без особых артифактов. Приведуем рещультат для

Разбиение:

Результат:

В формуле могло быть деление на ноль, так что я изменил её, добавив +1 в знаменатиле. Результат при не особо отличается от первого.

Pano

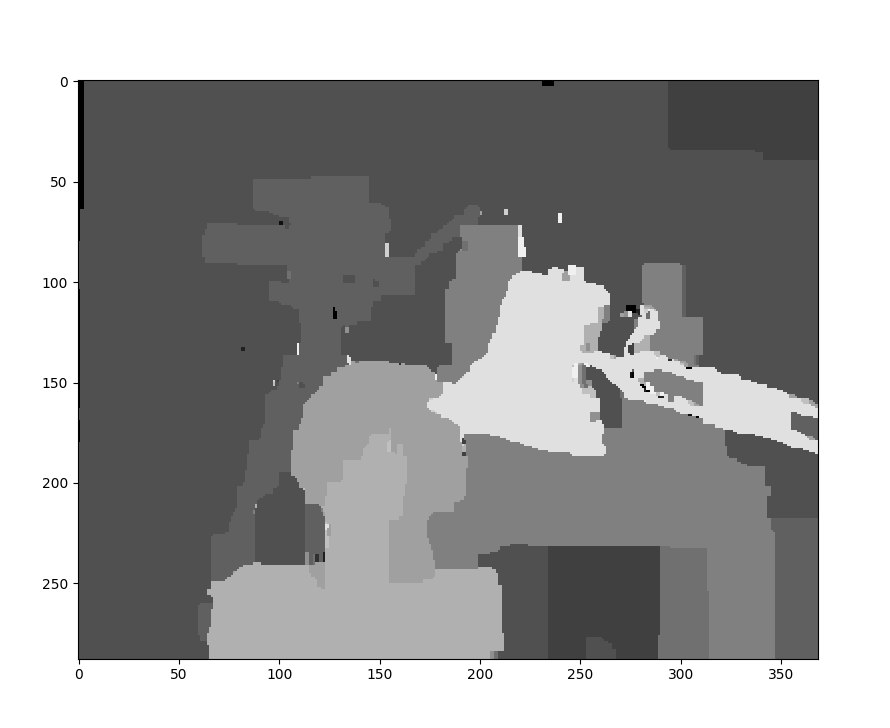
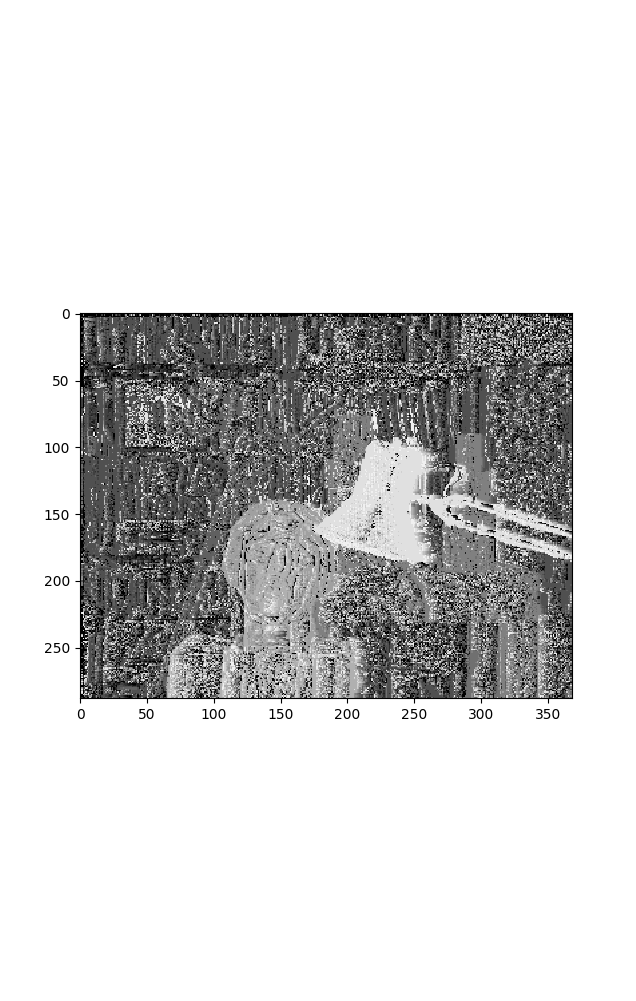
Разбиение:

Результат:  
  
Результат содержит очевидные артефакты склеивания – “обрезанных” людей на леснице. При альтернативном задании и все становится только хуже.

Задание 3. Стереозрение.

Код для решения задачи:

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  **import** numpy **as** np  **from** PIL **import** Image  **from** pystruct.utils **import** make\_grid\_edges  **from** pystruct.inference **import** inference\_dispatch  **from** utils **import** my\_inference\_dispatch  **from** skimage.color **import** rgb2yuv  **import** imageio  **import** visvis **as** vv  T = 1  W = 0.05  SIGMA = 18  K = 15  **def** load\_image(filename: str) -> np.ndarray:  img = imageio.imread(filename) *# type: Image.Image*  data = rgb2yuv(img)  **return** data  **def** task3():  n\_states = K + 1  image\_left = load\_image(**'./data/imL.png'**) *# (288, 384, 3)*  image\_right = load\_image(**'./data/imR.png'**) *# (288, 384, 3)*  **assert** image\_left.shape == image\_right.shape  height, width, \_ = image\_left.shape *# (288, 384, 3)*  cutted\_width = width - K  unary\_potentials = np.zeros( shape=(height, cutted\_width, n\_states))  **for** x **in** range(cutted\_width):  **for** y **in** range(height):  **for** state **in** range(n\_states):  diff = image\_left[y, x + state] - image\_right[y, x]  unary\_potentials[y, x, state] = min(np.linalg.norm(diff), SIGMA)    edges = make\_grid\_edges(unary\_potentials) *# (212462, 2)*  pairwise\_potentials = np.zeros(shape=(n\_states, n\_states))  **for** state\_1 **in** range(n\_states):  **for** state\_2 **in** range(n\_states):  pairwise\_potentials[state\_1, state\_2] = W\*min(abs(state\_1 - state\_2), T)    *# QPBO because max-product works slow.*  result = inference\_dispatch(-unary\_potentials, -pairwise\_potentials, edges, inference\_method=**'qpbo'**)    picture = result.reshape(height, cutted\_width)  plt.imshow(picture, vmin=0, vmax=n\_states, cmap=**'gray'**)  plt.show()  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  task3() |

Результат для = 15: , , со всеми связям  
Результат только с унарными потенциалами: