HorizonNet

AI 융합학부 - 길다영, 김현우

Contents

1. 수직 보정 전처리
2. 1D Layout 표현
3. 특징 추출기(Feature Extractor)
4. 순환 신경망(Recurrent Neural Network for Capturing Global Information)
5. Post-processing
6. Pano Stretch 데이터 확장(Data Augmentation)

Preprocess.py → model.py / inference.py → post_proc.py → layout_viewer.py

1 수직 보정 전처리

Preprocess.py

소실점과 line_Segment를 찾아서 정렬을 하고, 이미지를 저장.

```python
# Process each input
for i_path in paths:
    print('Processing', i_path, flush=True)

    # Load and cat input images
    img_ori = np.array(Image.open(i_path).resize((1024, 512), Image.BICUBIC))[..., :3]

    # VP detection and line segment extraction
    _, vp, _, _, panoEdge, _, _ = panoEdgeDetection(img_ori,
                                                    qError=args.q_error,
                                                    refineIter=args.refine_iter)
    # panoEdge : image for visualize line segments
    panoEdge = (panoEdge > 0)

    # Align images with VP
    i_img = rotatePanorama(img_ori / 255.0, vp[2::-1])
    l_img = rotatePanorama(panoEdge.astype(np.float32), vp[2::-1])

    # Dump results
    basename = os.path.splitext(os.path.basename(i_path))[0]
    if args.rgbonly:
        path = os.path.join(args.output_dir, '%s.png' % basename)
        Image.fromarray((i_img * 255).astype(np.uint8)).save(path)
    else:
        path_VP = os.path.join(args.output_dir, '%s_VP.txt' % basename)
        path_i_img = os.path.join(args.output_dir, '%s_aligned_rgb.png' % basename)
        path_l_img = os.path.join(args.output_dir, '%s_aligned_line.png' % basename)

        Image.fromarray((i_img * 255).astype(np.uint8)).save(path_i_img)
        Image.fromarray((l_img * 255).astype(np.uint8)).save(path_l_img)
```

파노라마 사진에서 edge와 소실점을 찾음

소실점 가진 이미지를 align
(구부러진 사진이 펴지는 것 의미)

저장된 이미지를 다른 파일로 넘기는 부분

구부러진 사진을 바로 세움

소실점 찾음.

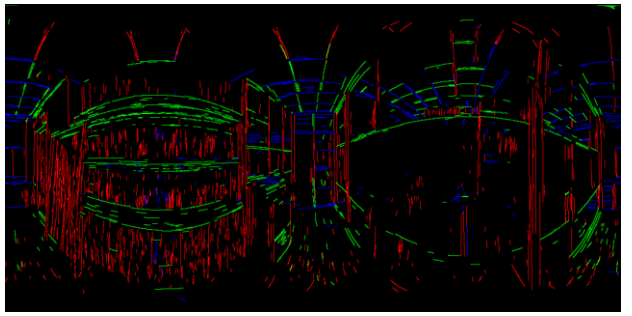
찾은 edge

결과물

Demo424_aligned_rgb.raw.png

```
-0.000345 0.002584 0.999997
0.000766 0.999993 -0.003688
1.000000 -0.000765 0.000347
```

Demo424_VP.txt



Demo424_aligned_line.png

2 1D Layout 표현

수직 보정 전처리

Yc : 천장-벽 경계

Yf : 바닥-벽 경계

Yw : 벽-벽 경계

Inference.py

Preprocess된 이미지 불러옴.

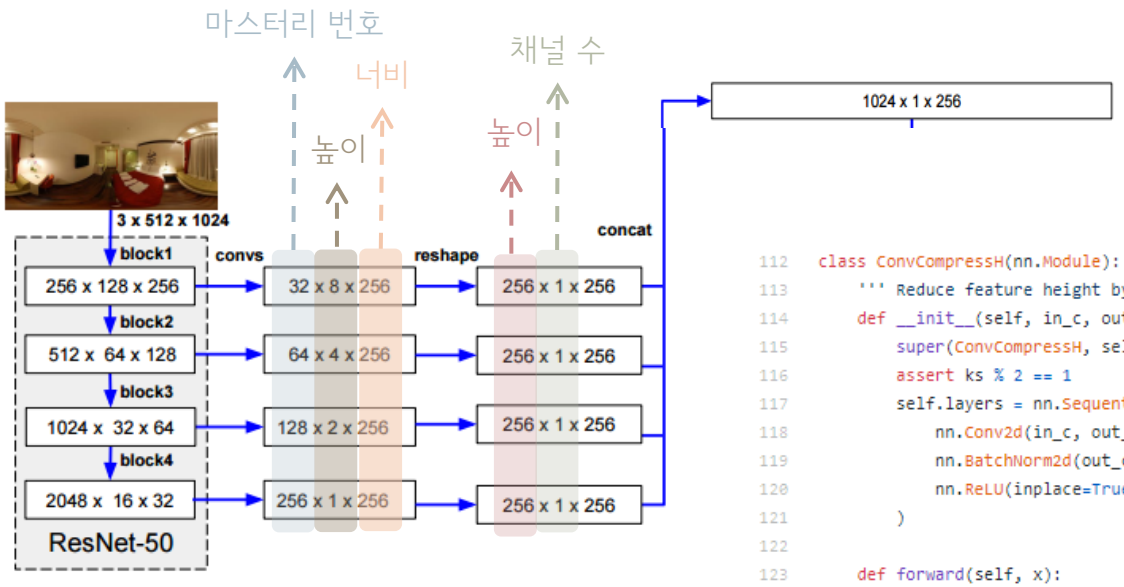
```python
if __name__ == '__main__':
    # Prepare image to processed
    paths = sorted(glob.glob(args.img_glob))
    if len(paths) == 0:
        print('no images found')
    for path in paths:
        assert os.path.isfile(path), '%s not found' % path

    # Check target directory
    if not os.path.isdir(args.output_dir):
        print('Output directory %s not existed. Create one.' % args.output_dir)
        os.makedirs(args.output_dir)
    device = torch.device('cpu' if args.no_cuda else 'cuda')

    # Loaded trained model
    net = utils.load_trained_model(HorizonNet, args.pth).to(device)
    net.eval()
```

Trained model(pth 파일) 불러옴

Inference()로 넘어감.

```python
# Inferencing
with torch.no_grad():
    for i_path in tqdm(paths, desc='Inferencing'):
        k = os.path.split(i_path)[-1][:-4]

        # Load image
        img_pil = Image.open(i_path)
        if img_pil.size != (1024, 512):
            img_pil = img_pil.resize((1024, 512), Image.BICUBIC)
        img_ori = np.array(img_pil)[..., :3].transpose([2, 0, 1]).copy()
        x = torch.FloatTensor([img_ori / 255])

        # Inferenceing corners
        cor_id, z0, z1, vis_out = inference(net, x, device,
                                            args.flip, args.rotate,
                                            args.visualize,
                                            args.force_cuboid,
                                            args.min_v, args.r)

        # Output result
        with open(os.path.join(args.output_dir, k + '.json'), 'w') as f:
            json.dump({
                'z0': float(z0),
                'z1': float(z1),
                'uv': [[float(u), float(v)] for u, v in cor_id],
            }, f)

        if vis_out is not None:
            vis_path = os.path.join(args.output_dir, k + '.raw.png')
            vh, vw = vis_out.shape[:2]
            Image.fromarray(vis_out)\
                 .resize((vw//2, vh//2), Image.LANCZOS)\
                 .save(vis_path)
```

3 | 특징 추출기(Feature Extractor)



model.py

```python
class ConvCompressH(nn.Module):
    ''' Reduce feature height by factor of two '''
    def __init__(self, in_c, out_c, ks=3):
        super(ConvCompressH, self).__init__()
        assert ks % 2 == 1
        self.layers = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=ks, stride=(2, 1), padding=ks//2),
            nn.BatchNorm2d(out_c),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.layers(x)
```

```python

class GlobalHeightConv(nn.Module):
    def __init__(self, in_c, out_c):
        super(GlobalHeightConv, self).__init__()
        self.layer = nn.Sequential(
            ConvCompressH(in_c, in_c//2),
            ConvCompressH(in_c//2, in_c//2),
            ConvCompressH(in_c//2, in_c//4),
            ConvCompressH(in_c//4, out_c),
        )

    def forward(self, x, out_w):
        x = self.layer(x)

        assert out_w % x.shape[3] == 0
        factor = out_w // x.shape[3]
        x = torch.cat([x[..., -1:], x, x[..., :1]], 3)
        x = F.interpolate(x, size=(x.shape[2], out_w + 2 * factor), mode='bilinear', align_corners=False)
        x = x[..., factor:-factor]
        return x
```

```python

class GlobalHeightStage(nn.Module):
    def __init__(self, c1, c2, c3, c4, out_scale=8):
        ''' Process 4 blocks from encoder to single multiscale features '''
        super(GlobalHeightStage, self).__init__()
        self.cs = c1, c2, c3, c4
        self.out_scale = out_scale
        self.ghc_lst = nn.ModuleList([
            GlobalHeightConv(c1, c1//out_scale),
            GlobalHeightConv(c2, c2//out_scale),
            GlobalHeightConv(c3, c3//out_scale),
            GlobalHeightConv(c4, c4//out_scale),
        ])

    def forward(self, conv_list, out_w):
        assert len(conv_list) == 4
        bs = conv_list[0].shape[0]
        feature = torch.cat([
            f(x, out_w).reshape(bs, -1, out_w)
            for f, x, out_c in zip(self.ghc_lst, conv_list, self.cs)
        ], dim=1)
        return feature
```

ResNet-50의 각 block

: 3개의 convolution Layer을 포함하고 특정 피라미드를 출력 함.

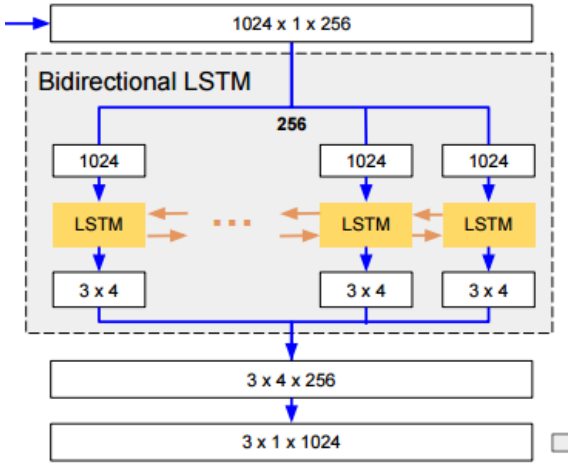
: Convolution 커널의 높이는 4, 2, 2 / 너비는 1 / 채널 수는 원래의 절반 (%2)

Convs : 3개의 Convolution 후 마스터리 번호는 /8(2*2*2), 높이 /16(4*2*2), 너비 = 256이 되도록 업 샘플링 함.

Reshape : 높이가 256이 되도록 모양을 변경하면 채널 수가 1이 된다.

Concat : 4ro의 기능 맵을 연결하여 크기를 만든다. (2014 * 1 * 256)

4 순환 신경망 (Recurrent Neural Network for Capturing Global Information)

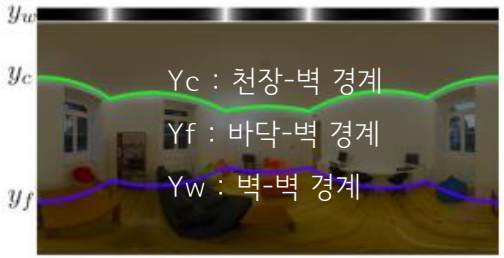


model.py

```python
    def forward(self, x):
        if x.shape[2] != 512 or x.shape[3] != 1024:
            raise NotImplementedError()
        x = self._prepare_x(x)
        conv_list = self.feature_extractor(x)
        feature = self.reduce_height_module(conv_list, x.shape[3]//self.step_cols)

        # rnn
        if self.use_rnn:
            feature = feature.permute(2, 0, 1)  # [w, b, c*h]
            output, hidden = self.bi_rnn(feature)  # [seq_len, b, num_directions * hidden_size]
            output = self.drop_out(output)
            output = self.linear(output)  # [seq_len, b, 3 * step_cols]
            output = output.view(output.shape[0], output.shape[1], 3, self.step_cols)  # [seq_len, b, 3, step_cols]
            output = output.permute(1, 2, 0, 3)  # [b, 3, seq_len, step_cols]
            output = output.contiguous().view(output.shape[0], 3, -1)  # [b, 3, seq_len*step_cols]
        else:
            feature = feature.permute(0, 2, 1)  # [b, w, c*h]
            output = self.linear(feature)  # [b, w, 3 * step_cols]
            output = output.view(output.shape[0], output.shape[1], 3, self.step_cols)  # [b, w, 3, step_cols]
            output = output.permute(0, 2, 1, 3)  # [b, 3, w, step_cols]
            output = output.contiguous().view(output.shape[0], 3, -1)  # [b, 3, w*step_cols]

        # output.shape => B x 3 x W
        cor = output[:, :1]  # B x 1 x W
        bon = output[:, 1:]  # B x 2 x W

        return bon, cor
```

기하학적으로 말하면 방의 모든 구석은 다른 구석의 위치에서 대략적으로 유추 할 수 있으므로 RNN은 전역 정보 및 장기 종속성을 캡처하는 데 사용됩니다. LSTM의 셀 상태에있는 다른 영역의 정보를 저장합니다. RNN은 열별로 열을 예측합니다. $와이_{바}$, $와이_{에프}$, $와이_{w}$ 즉, RNN의 시퀀스 길이는 이미지 너비에 비례합니다. 이 논문의 방법에서 RNN은 시간 단계 당 1 개의 열이 아닌 4 개의 열을 예측하며 정확도에 영향을주지 않고 계산 시간이 더 적게 걸립니다. 파노라마의 왼쪽과 오른쪽이 연속적이기 때문에 **양방향 LSTM이 사용됩니다** .

5 • Post-processing



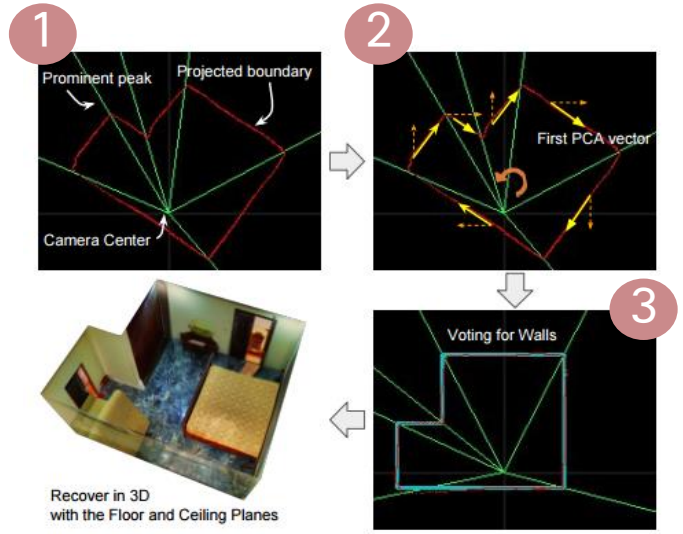
가정
I. 교차 벽은 서로 수직이다(Manhattan world가정).
II. 바닥과 천장이 서로 평행한 모든 객실의 1층 1층 배치
III. 카메라 높이는 1.6미터
IV. 전처리 단계에서 바닥을 y축에 직교하도록 올바르게 정렬합니다.

misc/post_proc.py

1. 바닥 및 천장 평면 복구
천장-바닥 거리 투표하기 위해 yf(바닥-벽), yc(천장-벽)에서 상응하는 값을 사용할 수 있다.
가정된 카메라 높이를 기반으로 이미지에서 3D XYZ 위치로 바닥- 벽 경계인 yf를 투영할 수 있습니다(모두 동일한 Y를 공유함).
천장-벽 경계 yc는 동일한 3D X, Z 위치를 동일한 영상 열에 있는 yf와 공유합니다.
그러므로 바닥과 천장 사이의 거리는 계산될 수 있다.

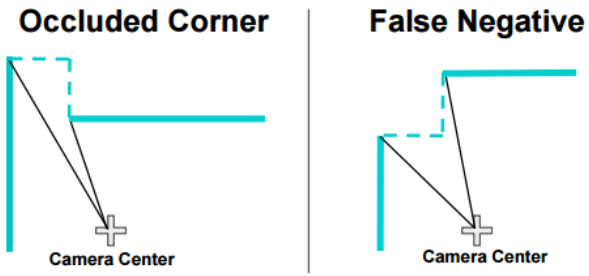
$$s_i = \frac{-1.6}{tan(\phi_i^f)}$$
$$H_i^c = s_i \times tan(\phi_i^c)$$
$$H = H_f + \text{mean}_i(H_i^c)$$

2. 벽면 복구
빨간 선 : Yc(천장-벽)
1 녹색 선 : 눈에 띄는 봉우리는 여러 부분으로 나눔
2 각 부품에서 PCA를 수행하여 부품의 주 방향 벡터를 얻음. 각 부품의 방향 각도를 평균화하고 전체 평면을 회전함.
3 즉, 벽은 XZ 축과 정렬되어야 하고 인접한 벽은 수직이며 마지막으로 벽이 선택됨



(a) Depicting how we recover the wall planes from our model output.

```python
def gen_ww_cuboid(xy, gpid, tol):
    xy_cor = []
    assert len(np.unique(gpid)) == 4

    # For each part seperated by wall-wall peak, voting for a wall
    for j in range(4):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score})
```

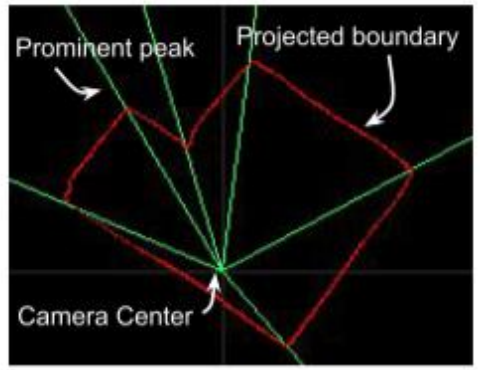


Gen_ww_Cuboid 자체에는 문제가 있음. 두 경우 모두 벽 대신 모서리를 추가함.
(벽에 투표하는 대신, 두개의 두드러진 봉우리와 두 개의 벽의 위치에 따라 코너를 추가함)

5 • Post-processing

```python
# Detech wall-wall peaks
if min_v is None:
    min_v = 0 if force_cuboid else 0.05
r = int(round(W * r / 2))
N = 4 if force_cuboid else None
xs_ = find_N_peaks(y_cor_, r=r, min_v=min_v, N=N)[0]

# Generate wall-walls
cor, xy_cor = post_proc.gen_ww(xs_, y_bon_[0], z0, tol=abs(0.16 * z1 / 1.6), force_cuboid=force_cuboid)
if not force_cuboid:
    # Check valid (for fear self-intersection)
    xy2d = np.zeros((len(xy_cor), 2), np.float32)
    for i in range(len(xy_cor)):
        xy2d[i, xy_cor[i]['type']] = xy_cor[i]['val']
        xy2d[i, xy_cor[i-1]['type']] = xy_cor[i-1]['val']
    if not Polygon(xy2d).is_valid:
        print(
            'Fail to generate valid general layout!! '
            'Generate cuboid as fallback.',
            file=sys.stderr)
        xs_ = find_N_peaks(y_cor_, r=r, min_v=0, N=4)[0]
        cor, xy_cor = post_proc.gen_ww(xs_, y_bon_[0], z0, tol=abs(0.16 * z1 / 1.6),


# Expand with btn coory
cor = np.hstack([cor, post_proc.infer_coory(cor[:, 1], z1 - z0, z0)[:, None]])

# Collect corner position in equirectangular
cor_id = np.zeros((len(cor)*2, 2), np.float32)
for j in range(len(cor)):
    cor_id[j*2] = cor[j, 0], cor[j, 1]
    cor_id[j*2 + 1] = cor[j, 0], cor[j, 2]

# Normalized to [0, 1]
cor_id[:, 0] /= W
cor_id[:, 1] /= H

return cor_id, z0, z1, vis_out
```

Find_N_peaks()
: prominent peak를 찾는 과정



Gen_ww()
: cuboid, non-cuboid 구분
: gen_ww_cuboid() -> vote 함수
: gen_ww_general() -> vote 함수

For문의 개수 차이
Cuboid : 4개
Non-cuboid : 모서리 개수

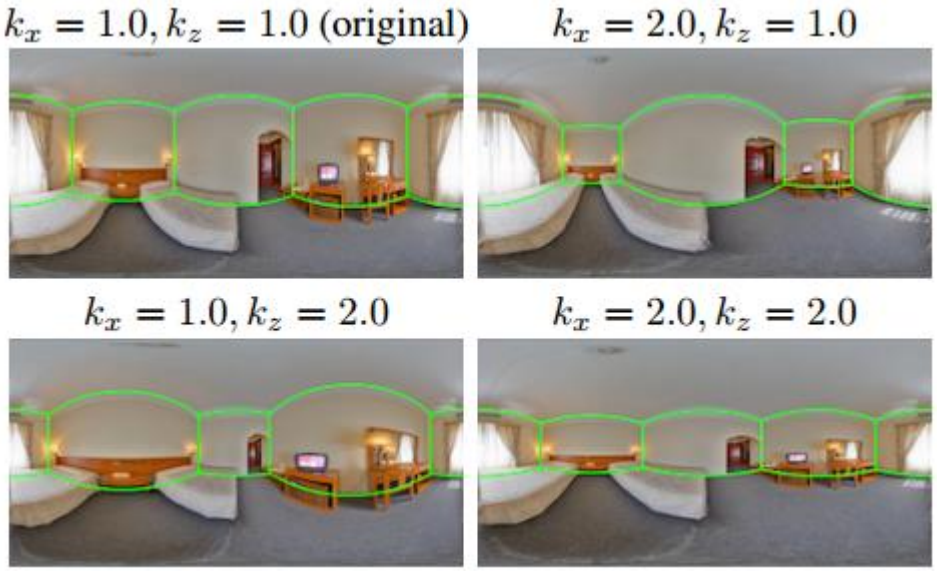
misc/post_proc.py

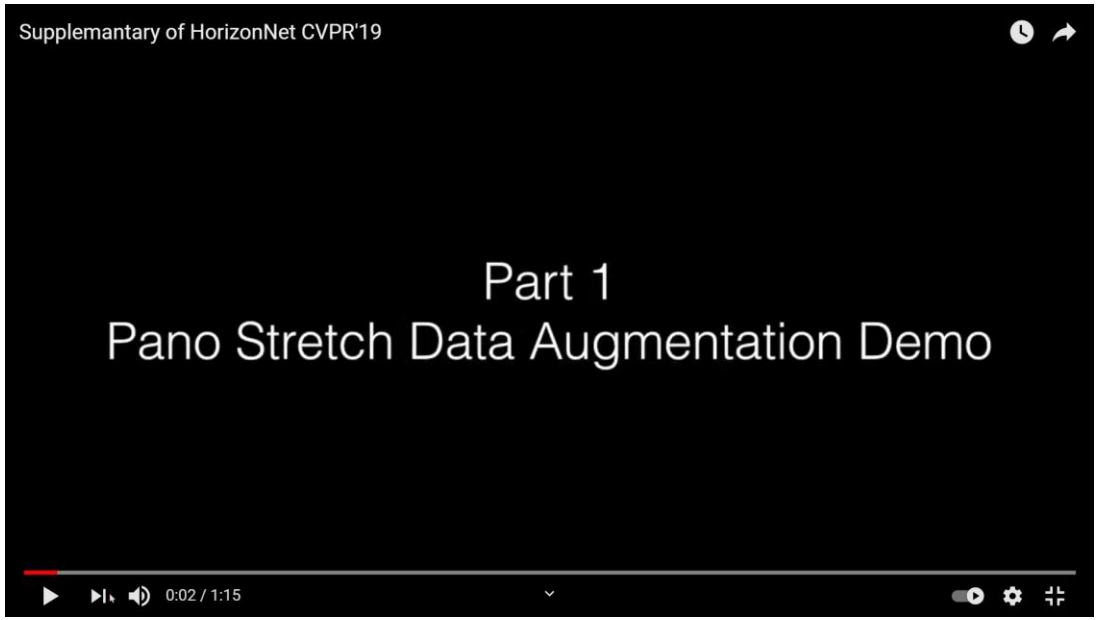
```python
def gen_ww_cuboid(xy, gpid, tol):
    xy_cor = []
    assert len(np.unique(gpid)) == 4

    # For each part seperated by wall-wall peak, voting for a wall
    for j in range(4):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score})
```

```python
def gen_ww_general(init_coorx, xy, gpid, tol):
    xy_cor = []
    assert len(init_coorx) == len(np.unique(gpid))

    # Candidate for each part seperated by wall-wall boundary
    for j in range(len(init_coorx)):
        now_x = xy[gpid == j, 0]
        now_y = xy[gpid == j, 1]
        new_x, x_score, x_l1 = vote(now_x, tol)
        new_y, y_score, y_l1 = vote(now_y, tol)
        u0 = np_coorx2u(init_coorx[(j - 1 + len(init_coorx)) % len(init_coorx)])
        u1 = np_coorx2u(init_coorx[j])
        if (x_score, -x_l1) > (y_score, -y_l1):
            xy_cor.append({'type': 0, 'val': new_x, 'score': x_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
        else:
            xy_cor.append({'type': 1, 'val': new_y, 'score': y_score, 'action': 'ori', 'gpid': j, 'u0': u0, 'u1': u1, 'tbd': True})
```

```python
def gen_ww(init_coorx, coory, z=50, coorW=1024, coorH=512, floorW=1024, floorH=512, tol=3, force_cuboid=True):
    gpid = get_gpid(init_coorx, coorW)
    coor = np.hstack([np.arange(coorW)[:, None], coory[:, None]])
    xy = np_coor2xy(coor, z, coorW, coorH, floorW, floorH)

    # Generate wall-wall
    if force_cuboid:
        xy_cor = gen_ww_cuboid(xy, gpid, tol)
    else:
        xy_cor = gen_ww_general(init_coorx, xy, gpid, tol)

    # Ceiling view to normal view
    cor = []
    for j in range(len(xy_cor)):
        next_j = (j + 1) % len(xy_cor)
        if xy_cor[j]['type'] == 1:
            cor.append((xy_cor[next_j]['val'], xy_cor[j]['val']))
        else:
            cor.append((xy_cor[j]['val'], xy_cor[next_j]['val']))
    cor = np_xy2coor(np.array(cor), z, coorW, coorH, floorW, floorH)
    cor = np.roll(cor, -2 * cor[::2, 0].argmin(), axis=0)

    return cor, xy_cor
```

6 Pano Stretch 데이터 확장(Data Augmentation)





녹색선 : 이미지와 ground truth 배치

녹색 선은 x또는 z축을 따라 늘어난다.

→ 룸의 길이와 너비를 변경하여 데이터를 늘릴 수 있다.

이 증강 전략은 모든 실험 설정에서 정량적 결과를 개선한다.(정확도 높이는데 사용)

This augmentation strategy improves our quantitative results under all experiment settings .

6 Pano Stretch 데이터 확장(Data Augmentation)

inference.py

```python
def inference(net, x, device, flip=False, rotate=[], visualize=False,
              force_cuboid=True, min_v=None, r=0.05):
    '''
    net   : the trained HorizonNet
    x     : tensor in shape [1, 3, 512, 1024]
    flip  : fliping testing augmentation
    rotate: horizontal rotation testing augmentation
    '''

    H, W = tuple(x.shape[2:])

    # Network feedforward (with testing augmentation)
    x, aug_type = augment(x, flip, rotate)
    y_bon_, y_cor_ = net(x.to(device))
    y_bon_ = augment_undo(y_bon_.cpu(), aug_type).mean(0)
    y_cor_ = augment_undo(torch.sigmoid(y_cor_).cpu(), aug_type).mean(0)

    # Visualize raw model output
    if visualize:
        vis_out = visualize_a_data(x[0],
                                   torch.FloatTensor(y_bon_[0]),
                                   torch.FloatTensor(y_cor_[0]))
    else:
        vis_out = None

    y_bon_ = (y_bon_[0] / np.pi + 0.5) * H - 0.5
    y_cor_ = y_cor_[0, 0]
```

```python
def augment(x_img, flip, rotate):
    x_img = x_img.numpy()
    aug_type = ['']
    x_imgs_augmented = [x_img]
    if flip:
        aug_type.append('flip')
        x_imgs_augmented.append(np.flip(x_img, axis=-1))
    for shift_p in rotate:
        shift = int(round(shift_p * x_img.shape[-1]))
        aug_type.append('rotate %d' % shift)
        x_imgs_augmented.append(np.roll(x_img, shift, axis=-1))
    return torch.FloatTensor(np.concatenate(x_imgs_augmented, 0)), aug_type
```

Thank You !