SomaCube Software Description

총 **5**개의 folder 로 이루어짐

> Annotation
>
> Calibraiton
>
> Pose Estimation
>
> Segmentation
>
> Run

1. Annotation

1-0. description

- cocohandai550/, data4cv/: 디렉토리는 딥러닝 학습용, data4cv는 computer vision 코드의 검증용으로 사용. 세부적인 레이블링의 기준이 다를 뿐 같은 코드를 사용.
- cocohandai/annotations_train.json: training annotation 파일 (레이블)
- cocohandai/annotations_train.json: validation annotation 파일 (레이블)
- cocohandai550/train: training 이미지
- cocohandai550/val: validation 이미지
- cocohandai550/ann_train: training 이미지에서 이미지들의 annotation 조각(json파일)들의 모음
- cocohandai550/ann_val: validation 이미지에서 이미지들의 annotation 조각(json파일)들의 모음
- bbseg.py: 이미지의 annotation(segementation, bounding box)를 만들어 ann_train, ann_val에 저장함
- collect.py: 사진기와 pc를 연결해 사진을 찍고 저장함
- merge_ann_train.py: 이미지들의 annotation 조각(train)을 합침
- merge_ann_val.py: 이미지들의 annotation 조각(val)을 합침

2. Calibration

2-0. intrinsic_calibration.py : 카메라를 intrinsic calibration 하기 위해 사용

- cam, w, h, size 를 argument로 입력.
- 격자(target)을 회전시키고 이동시키면서 사진을 찍음(자동 촬영)
- results/intrinsic//mtx.pkl 에 calibration 결과 (camera_to_image matrix)가 저장됨.

2-1. make_pos_image_pairs.py : extrinsic calibration을 위한 데이터 쌍을 얻기 위해 사용

- cam, data_dir 를 argument로 입력
- 화살표와 W/S 로 wrist의 위치를 조정, Q/E/H/K/Y/I 로 wrist의 rotation을 조정하고, 스페이스바로 사진을 찍음.
- 사진을 찍으면 사진이 <data_dir>/images에 저장되고, 그 시점에서 로봇의 손목 좌표가 <data_dir>/poses에 저장됨.

2-2. static_test.py : static camera에 대해 calibration한 것을 테스트하기 위해 사용

- h, w, size, auto를 argument로 입력
- 실행 시, static camera가 target을 보고 있어야 함(손목 등에 의해 가려지면 안됨).
- static camera가 찍은 image에서 target을 찾고, 변환을 여러 번 시켜서 target의 base좌표계 좌표를 구함.
- wrist를 target의 각 좌표로 이동시킴. auto == 1 인 경우 자동으로 이동하고, auto == 0 인 경우 스페이스바를 누르면 이동함.

manual_reaching.py : 정해진 위치에 놓여진 블록들을 정해진 모양대로 조립하게끔 로봇을 움직이는 코드

2-3. Moduels

cam_tool.py : 카메라 가동 및 조작을 위한 module. 현재는 sony와 realsense 카메라 모델만 지원함.

utils.py : 각종 utility function들이 정의되어 있는 module.

2-4. Simple test codes

gripper_test.py : Robot gripper를 테스트하기 위한 코드

preview.py : 카메라 연결 및 카메라에 찍히는 화면을 테스트하기 위한 코드

take_photo.py : 카메라로 사진을 찍어서 저장하기 위한 코드

3. Pose Estimation

3-0. Execution

- segmentation/deepLab/deeplab manual.md에 따라 환경 구축해야 함.
- deeplab.run()에 학습시키고 싶은 image를 넣어주고 코드 전체를 실행하면 됨. (ex)out = deeplab.run("blocks.jpg"))
- img = cv2.imread('', cv2.IMREAD_GRAYSCALE), img_color = cv2.imread('', cv2.IMREAD_COLOR) 에도 같은 이미지를 넣어주어 시각적으로 결과를 확인 함.

3-1. Description

- pp3.py와 grippoint.py는 DeeplabModule()을 통해 7가지 block의 종류와 위치를 구하여 로봇이 block을 잡아야 할 좌표를 계산함.
- input image를 학습시켜 블록의 lable을 구분함. OpenCV의 contourArea()로 영역의 크기를 지정해서 선택된 영역에만 boundingRect()를 쳐서 block의 angle을 구함. (이때 angle은 절대값이 아닌 90이하의 값)
- 정확한 회전각을 구하기 위해서 OpenCV의 matchTemplate()을 이용('TM_CCOEFF_NORMED' 사용). template 리스트는 mk_ps_tp()에서 생성됨.

- lable_to_pose에서 lable 당 가능한 pose 후보군을 설정. boundingRect()를 통해 구한 angle을 90 간격으로 4가지 후보군 설정.
- 위에 대한 결과로 얻은 결과로 회전된 절대각을 구하여 pose1(),pose2(),pose3(),pose4(),pose5(),pose6(),pose7(),pose8()에서 잡아야 할 좌표를 계산해줌. pose가 같더라도 block 당 잡아야할 위치가 다르기 때문에 lable에 따라 다르게 계산되도록 함.
- pp3의 출력값은 template 매칭된 결과의 대각선 좌표들이며, grippoint의 결과는 잡아야 할 좌표임.

4. Segmentation

4-1. DeepLab

0. environment 구성

> deeplab 코드는 다음의 github을 참고하였고 environment 구성은 anaconda로 설명되어 있는 그대로 구성하였음
>
> https://github.com/jfzhang95/pytorch-deeplab-xception?source=post_page----------
>
> 단, pycocotools는 다음 링크를 참조하여 깔아야 문제 없이 python에서 pycocotools를 사용할 수 있을 것
>
> https://github.com/philferriere/cocoapi

1. code 실행

- for training: 'pytorch-deeplab-xception' 에 들어가서 train_coco.sh 실행
- for valdating: 'pytorch-deeplab-xception' 에 들어가서 val_coco.sh 실행

> run 디렉토리의 최신 experiment 폴더에 저장될 것임
>
> training 시에 조금 시간이 지난 후에 model parameter saving이 됨
>
> tensorboard 켜서 groundtruth, prediction 확인 가능

2. custom dataset configuration

- COCO dataset 형식에 따라 dataset을 만들었음

> train data 500개, val data 50개로 구성handai_annotations*[train/val].json 에 annotation 데이터 저장handai_dataset_img*[train/val] 에 img 데이터 저장

- cube dataset 은 dataset 디렉토리에 있음

> handai_annotations_train.json: annotation json file for training
>
> handai_annotations_val.json: annotation json file for validation
>
> handai_dataset_img_train: img files for training
>
> handai_dataset_img_val: img files for validation

3. Dataset Preprocessing 관련

- *'pytorch-deeplab-xception/datasloaders/datasets/coco.py'* 코드를 읽어보면 됨
- *line 16*: NUM_CLASSES = background + cube 7 종류 = 8 카테고리
- *line 18*: 기존에는 COCO dataset에서 사용할 카테고리 index 를 선택적을 넣었으나, 우리는 0~7 모두를 쓸 것이므로 0~7 모두 기입
- *line 31*: annotation specification json file 경로인데, split 에 따라 다른 파일 참조

  > dataset 디렉토리에 보면, 'handai_dataset_ann_train' 과 'handai_dataset_ann_val'가 그것임

- *line32*: handai_ann_ids 경로인데, deeplab 에서 필요한 파일인거 같음

  > train_coco.sh 나 val_coco.sh 돌리면 handai_ann_ids_[train|val].pth 가 만들어지는데, 해당 파일이 없을때 처음에만 processing 되므로, dataset에 image나 annotation을 추가 혹은 변경하였다면 handai_ann_ids[train|val].pth 를 삭제하고 코드를 실행시킬 것

- *line 35*: handai_dataset_img 경로이고 마찬가지로 split에 다라 다른 파일 참조

  > dataset 디렉토리에 보면, 'handai_dataset_img_train' 과 'handai_dataset_img_val'이 그것임

4. trained model

- val_coco.sh의 —*resume* 인자에 적혀있는 'run/coco/deeplab-resnet/experiment_1/checkpoint.pth.tar'가 trained model임
- val.py 는 validation img들을 masked_imgs 디렉토리에 저장하고 iou score를 iou.csv 파일에 저장
- *line 173 ~ line 192* : iou, masked img 저장 코드

4-2. MaskRCNN

기존의 MaksRCNN github repository 를 customized dataset 을 이용할 수 있도록 변형시켰음

0. dataset

- mask_rcnn/datasets 폴더 안에 handai dataset 을 위치시킴

1. run

- *mask_rcnn/samples 의 inspect_data.ipynb*: data 의 형태가 어떻게 생겼는지 확인할 수 있는 코드
- *mask_rcnn/samples 의 inspect_model.ipynb*: model 을 실제로 돌려볼 수 있음
- *handai.py*: handai dataset 에 맞는 function 들 정의

5. Run

5-1. Real

0. execution

0 - extrinsic_calibration.py : 카메라를 extrinsic calibration하기 위해 사용

- cam, w, h, size, data_dir 를 argument로 입력

- cam이 static인 경우 : 손목에 붙어있는 target을 static camera로 촬영하는 경우를 상정. camera_to_base matrix와 target_to_wrist matrix를 추정.
- cam이 wrist인 경우 : 바닥에 가만히 있는 target을 wrist camera로 촬영하는 경우를 상정. camera_to_wrist matrix와 target_to_base matrix를 추정.
- cam이 both인 경우 : target을 wrist camera와 static camera로 동시에 촬영하는 경우를 상정. static_camera_to_base matrix와 wrist_camera_to_wrist matrix를 추정.
- intrinsic_calibration.py에서 구한 카메라 정보와, make_pos_image_pairs.py 에서 만든 데이터 쌍들을 이용해 오브젝트 간의 좌표 변환 matrix를 추정
- results/extrinsic//mtx.pkl 에 결과가 저장됨.
    - cam이 static인 경우, camera_to_base matrix와 target_to_wrist matrix를 stack한 array가 저장됨.
    - cam이 wrist인 경우, camera_to_wrist matrix와 target_to_base matrix를 stack한 array가 저장됨.
    - cam이 both인 경우, static_camera_to_base matrix와 wrist_camera_to_wrist matrix를 stack한 array가 저장됨.

1- utils.py : 각종 utility function들이 정의되어 있는 module.

5-2. Webots

0. jfiddle.lua

- Webots simulation 환경에서 UR5 로봇을 이용하여 이미 놓여져 있는 cube 들을 빈 공간에 조립하는 herd coded script

1. 실행법

- terminal 상에서 ./jiddle.lua 로 실행시키면 하나의 시나리오가 실행됨

6. Data

handai dataset(총 7가지 큐브에 대한 dataset) 와 관련한 링크

6-1. template matching dataset

- template matching train/test 를 위한 customized dataset
- https://drive.google.com/file/d/1bAEr9jXUWr3vhHnhQSpes2ra1qxlBePW/view

6-2. segmentation dataset

- segmentation train/test 를 위한 customized dataset