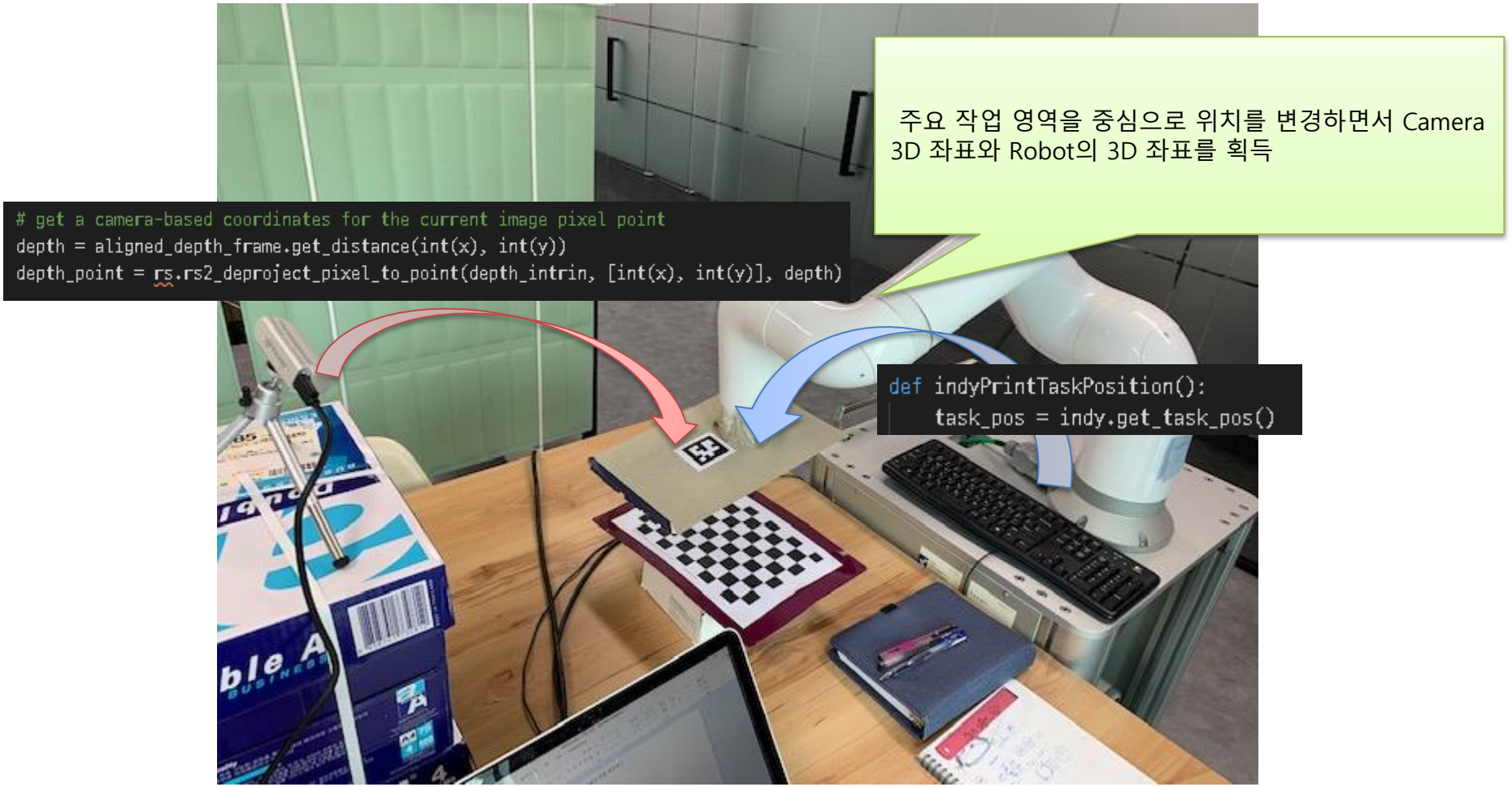
Weekly Report #1

#1 SVD를 이용한 변환 매트릭스 획득



#1 SVD를 이용한 변환 매트릭스 획득

참고: https://math.stackexchange.com/questions/222113/given-3-points-of-a-rigid-body-in-space-how-do-i-find-the-corresponding-orienta/222170#222170

OpenCV의 solve 함수(DECOMP_SVD)를 통해서 선형방정식의 해(변환 매트리스) 획득

AX=B

A: Camera 3D 좌표 n개 샘플 (n by 4)

X: 변환 매트릭스 (4 by 4)

B: Robot 3D 좌표 n개 샘플 (n by 4)

```python
def calculateTransformMatrix(srcPoints, dstPoints):
    assert(len(srcPoints) == len(dstPoints))

    p = np.ones([len(srcPoints), 4])
    p_prime = np.ones([len(dstPoints), 4])
    for idx in range(len(srcPoints)):
        p[idx][0] = srcPoints[idx][0]
        p[idx][1] = srcPoints[idx][1]
        p[idx][2] = srcPoints[idx][2]

        p_prime[idx][0] = dstPoints[idx][0]
        p_prime[idx][1] = dstPoints[idx][1]
        p_prime[idx][2] = dstPoints[idx][2]

    trMatrix = cv2.solve(p, p_prime, flags=cv2.DECOMP_SVD)
    return trMatrix
```

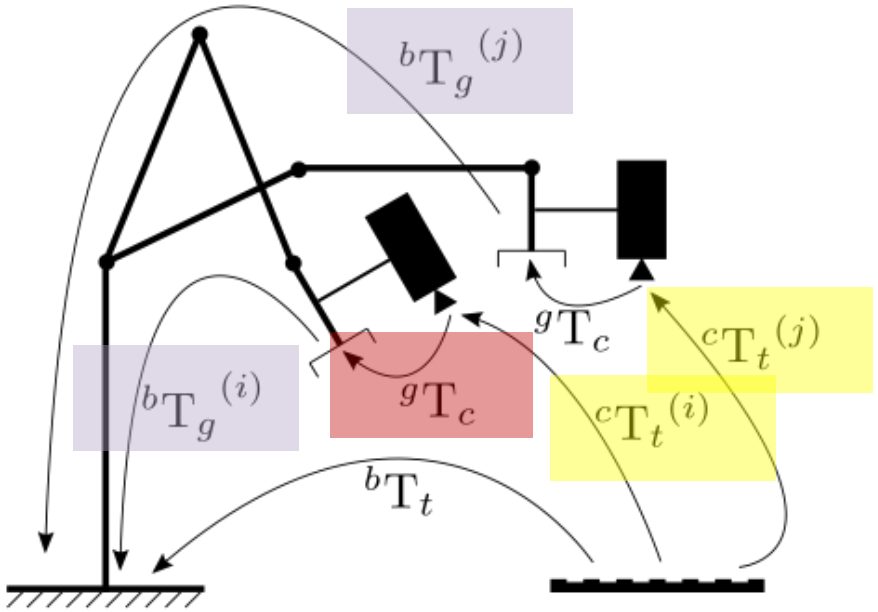
#2 OpenCV calibrateHandeye 적용

calibrateHandEye()

```
void cv::calibrateHandEye ( InputArrayOfArrays  R_gripper2base,
                            InputArrayOfArrays  t_gripper2base,
                            InputArrayOfArrays  R_target2cam,
                            InputArrayOfArrays  t_target2cam,
                            OutputArray         R_cam2gripper,
                            OutputArray         t_cam2gripper,
                            HandEyeCalibrationMethod method = CALIB_HAND_EYE_TSAI
                          )
```

Python:

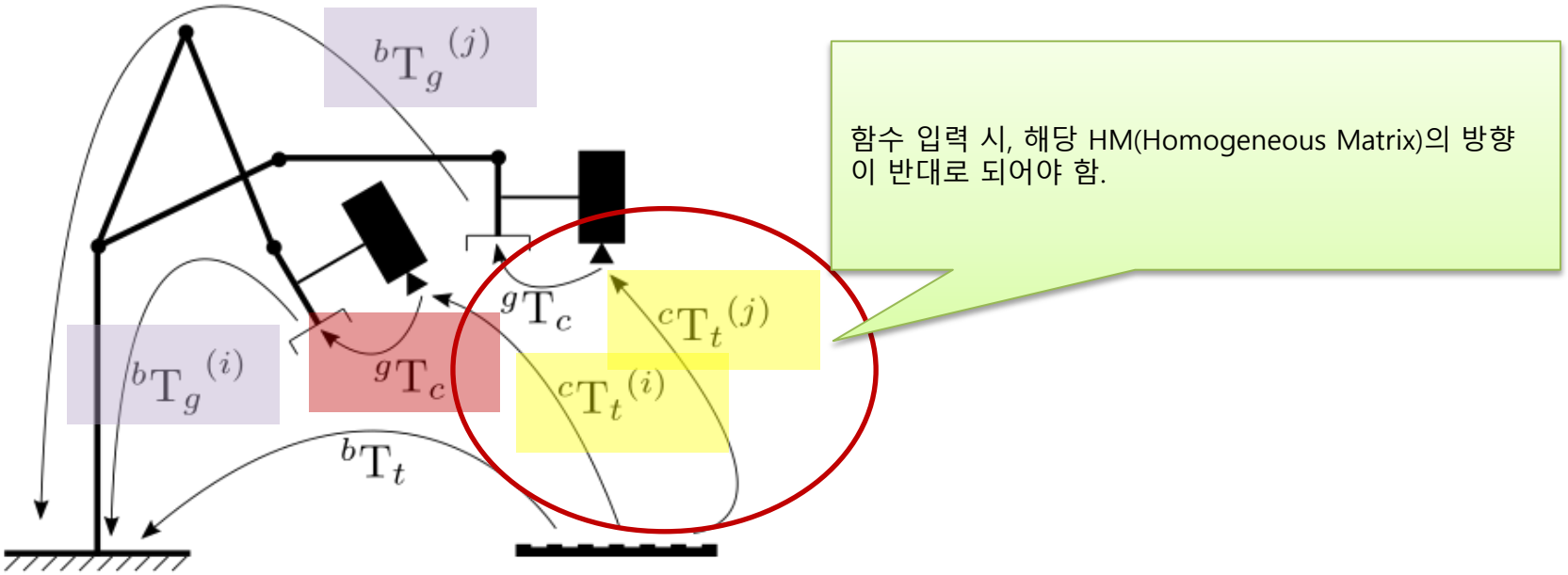
```
R_cam2gripper, t_cam2gripper = cv.calibrateHandEye( R_gripper2base, t_gripper2base, R_target2cam, t_target2cam[, R_cam2gripper[, t_cam2gripper[, method]]] )
```



#2 OpenCV calibrateHandeye 적용

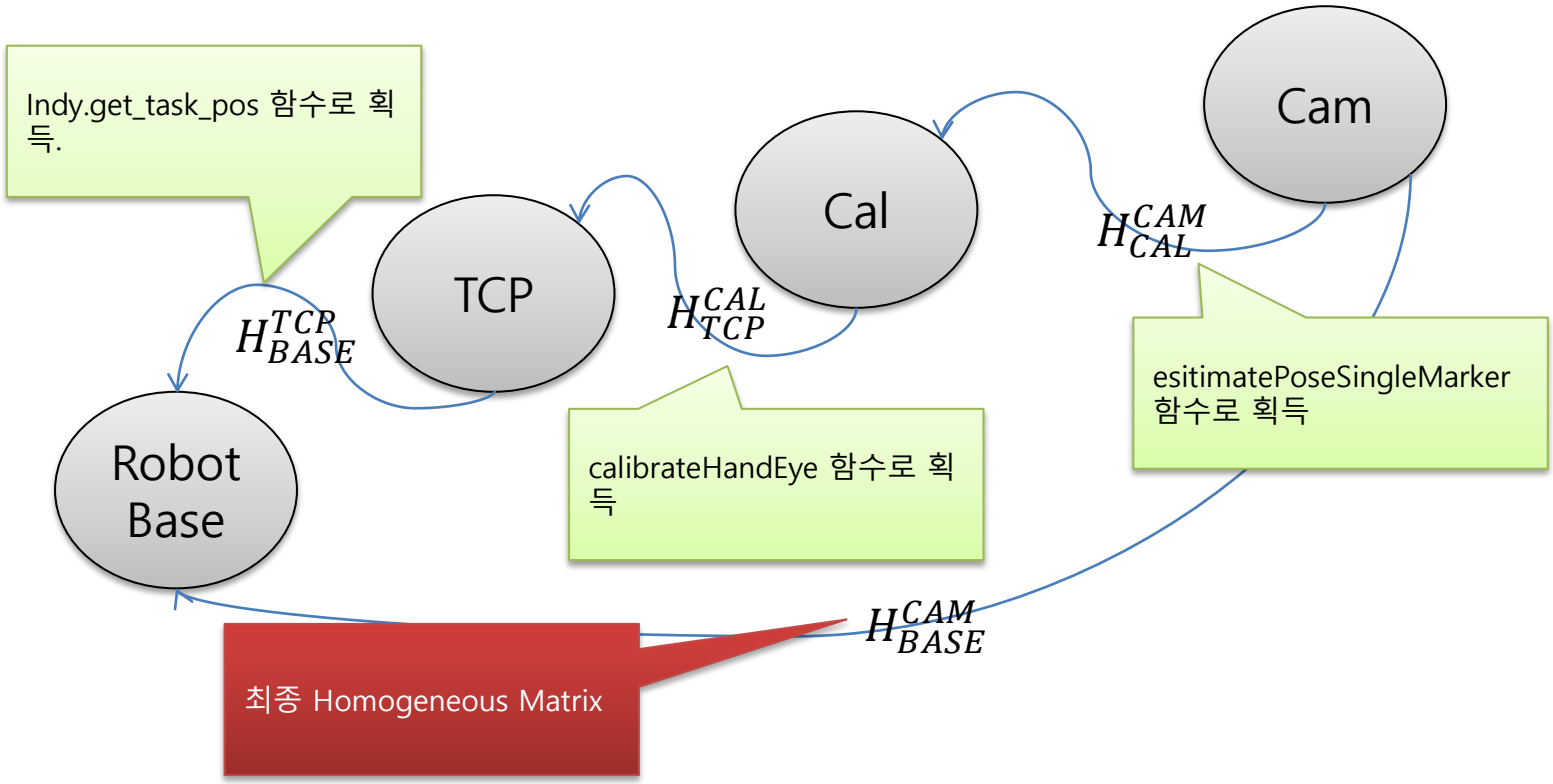
OpenCV calibrateHandEye 함수를 Hand-to-Eye에 적용하기 위해서는 Camera의 위치와 Calibration Panel이 위치가 변경되어야 함.

또한, Hand-in-Eye와 달리, 최종적으로는 Robot Base와 Camera 간의 변환 매트릭스 획득이 필요.



#2 OpenCV calibrateHandeye 적용

OpenCV calibrateHandEye 함수의 결과값을 Homogeneous 변환 에 따라 최종적으로 Base에서 Cam으로의 Homogeneous Matrix 획득.




```
R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(R_gripper2base, t_gripper2base, R_target2cam, t_target2cam,
hmT2G = UtilHM.makeHM(R_cam2gripper, t_cam2gripper.T)
hmG2B = UtilHM.makeHM(R_gripper2base[idx], t_gripper2base[idx].reshape(1,3))
hmC2T = UtilHM.makeHM(R_target2cam[idx], t_target2cam[idx].reshape(1,3))
hmTransform = np.dot(hmG2B, hmT2G)
hmTransform = np.dot(hmTransform2, hmC2T)
print(hmTransform)
```

TODO List

- #1 방식과 #2 방식의 차이점 및 성능 확인 필요. (이론적으로는 #2 방식이 탄탄함.)
- #2 방식에서 5개의 알고리즘 중 적합한 알고리즘 선택 필요
- 개선된 환경에서 에러율 검토 필요.
  (참고: https://blog.zivid.com/importance-of-3d-hand-eye-calibration)
- Camera의 depth를 통해서 획득한 3D 좌표와 OpenCV 함수를 통해서 획득한 3D 좌표에 차이 발생 원인 확인

Weekly Report #2

Input Values(Translation) 검증

동일한 Marker Center 위치를 타겟으로 측정.

내부 Camera Matrix 사용									
	Realsense SDK의 depth 정보 사용			OpenCV estimatePoseSingleMarkers 함수			두 함수 간의 오차		
	X	Y	Z	X	Y	Z	X	Y	Z
MIN & MAX	0.079856403	0.169344187	0.645000041	0.08090296	0.17161119	0.65295498	0.00061395	0.0013023	0.00495498
	0.080400683	0.170394376	0.649000049	0.08125392	0.17250175	0.65700563	0.00136061	0.0028896	0.0110056
	(MAX - MIN)								
	0.00054428	**0.00105019**	**0.00400001**	**0.00035096**	**0.00089056**	**0.00405065**	**0.0007467**	**0.00159**	**0.006051**

생성된 Camera Matrix 사용									
	Realsense SDK의 depth 정보 사용			OpenCV estimatePoseSingleMarkers 함수			두 함수 간의 오차		
	X	Y	Z	X	Y	Z	X	Y	Z
MIN & MAX	0.079863541	0.169161946	0.645000041	0.08048165	0.16998658	0.64819995	2.4827E-05	5.244E-05	0.00019995
	0.080590069	0.17040208	0.649000049	0.08182668	0.17321007	0.66033415	0.00177626	0.0037599	0.01433412
	(MAX - MIN)								
	0.000726528	**0.00124013**	**0.00400001**	**0.00134503**	**0.00322349**	**0.0121342**	**0.0017514**	**0.00371**	**0.014134**

고정된 Marker를 측정해도 xyz 각각 0.7mm, 1.2mm, 0.4mm의 오차 측정.

Aruco Marker의 축이 고정될 경우에도 흔들리는 현상으로 인해서 발생되었을 것으로 예상됨.

1. 내부의 Camera Matrix를 사용하는 것이 체스보드 혹은 Aruco 보드를 통해서 생성된 Camera Matrix 보다 값의 편차도 적고 3D Depth에 의해 측정된 값에 근접함.
(Translation은 SDK에 의해서 획득되는 좌표값을 사용하는 것이 유리 할 것으로 판단.)

2 Camera Calibration의 시험에 따라서 측정되는 결과값이 매번 상이. → 다음 장표 참조.

Camera Calibration 검증

- Test Case 마다 조금씩 다른 결과값을 보여줌.
- Chessboard가 아닌 Arucoboard를 사용해도 측정 시마다 상이한 값을 보여줌.
- Camera Matrix의 값에 따라서 estimatePose 함수에 의해서 측정되는 tvec과 rvec 값이 달라짐. ← Hand Eye Calibration에 발생하는 오차값의 주요 원인이 될 것으로 예상됨.
(참고: https://answers.opencv.org/question/216633/calibratehandeye-precision/)
- 시험 데이터 참고

calibrateHandEye 함수 결과 검증

cv2.CALIB_HAND_EYE_TSAI
cv2.CALIB_HAND_EYE_PARK
cv2.CALIB_HAND_EYE_HORAUD
cv2.CALIB_HAND_EYE_ANDREFF
cv2.CALIB_HAND_EYE_DANIILIDIS

Method 0
[[-0.53111808 -0.66809495 0.5211168]
[-0.67767341 0.7041279 0.212044]
[-0.5085984 -0.2405266 -0.82672524]]
[[-0.22988355]
[-0.30189875]
[0.01499742]]

Distance: 0.379755

Method 1
[[-0.74201907 -0.66968512 0.03048841]
[-0.67034704 0.7416587 -0.02402535]
[-0.00652258 -0.03826508 -0.99924634]]
[[-0.032312]
[-0.13177883]
[0.00663744]]

Distance: 0.135845

Method 2
[[-0.74223362 -0.6694401 0.03064636]
[-0.67010743 0.74187888 -0.02391137]
[-0.00672865 -0.03828418 -0.99924424]]
[[-0.03235716]
[-0.13190149]
[0.00660721]]

Distance: 0.135973

Method 3
[[-0.75403104 -0.65672513 0.01221899]
[-0.65669077 0.75333574 -0.03524907]
[0.01394395 -0.03460299 -0.99930386]]
[[-0.1073896]
[-0.01718936]
[-0.23714391]]

Distance: 0.260893

Method 4
[[-0.73287686 -0.67946701 0.03487258]
[-0.68030057 0.73253307 -0.02421645]
[-0.00909104 -0.04147152 -0.99909833]]
[[-0.0371705]
[-0.13063565]
[0.00827459]]

Distance: 0.136073

---

1. 동일한 시험 데이터를 이용해서 계산 결과 TSI와 ANDREFF의 방식의 경우 비정상인 matrix를 반환.
2. 현재는 Method 2(Horaud 방식을 채용)를 통해서 검증 진행
3. 1번의 이슈가 Camera Matrix 등의 다른 문제가 없는 지 검증 필요.
4. 시험 측정 샘플에 따라서 다른 값을 보이는 것으로 예상되며 최적의 샘플 측정 방법에 대한 고찰 필요

TODO List

- 측정 침을 이용한 정확도 측정 및 시험 결과 도출

- Robot XYZUVW에 대한 움직임에 대한 특이점 여부 등의 사전 유효성 검사 방법 (문의 진행중): http://forum.neuromeka.com/topic/51/indydcp-task_move_to-%ED%95%A8%EC%88%98-%EC%82%AC%EC%9A%A9-%EA%B4%80%EB%A0%A8-%EB%AC%B8%EC%9D%98
  → Linux ROS 환경 구축(필요성?)

- Cognex 카메라 관련 기능 검증 및 현재 진행 중인 정확도 측정에 비교군으로 확인.