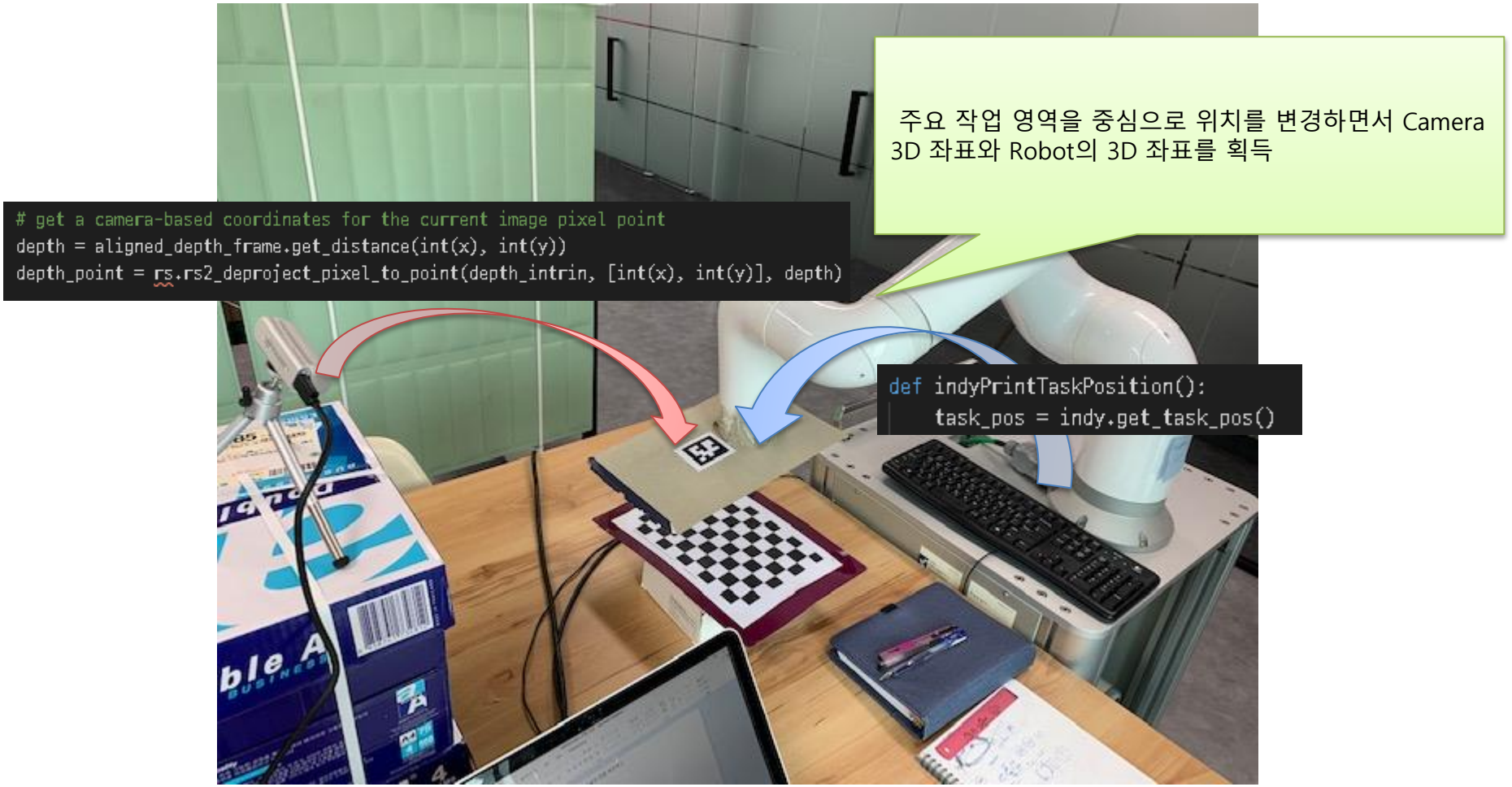
#1 SVD를 이용한 변환 매트릭스 획득



#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
```

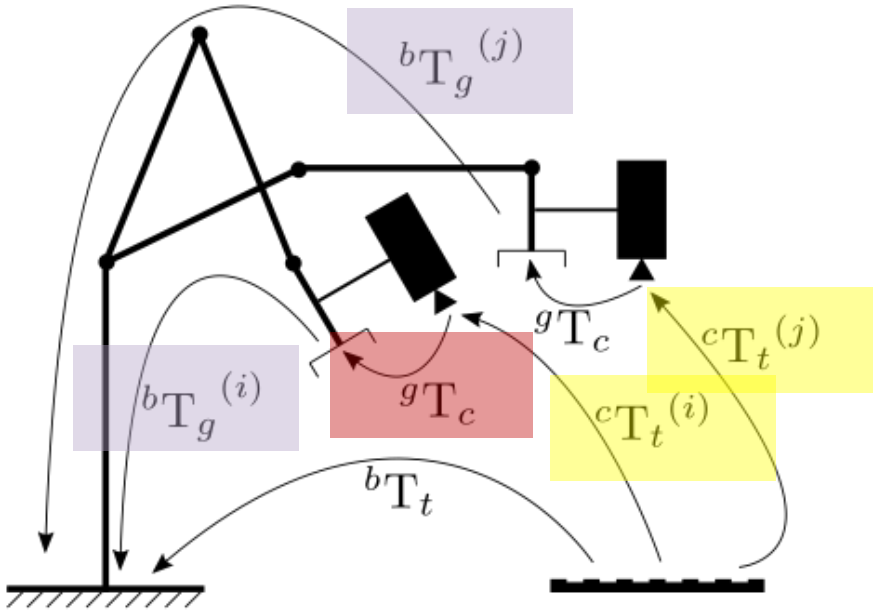
#2 OpenCV calibrateHandeye 적용

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:

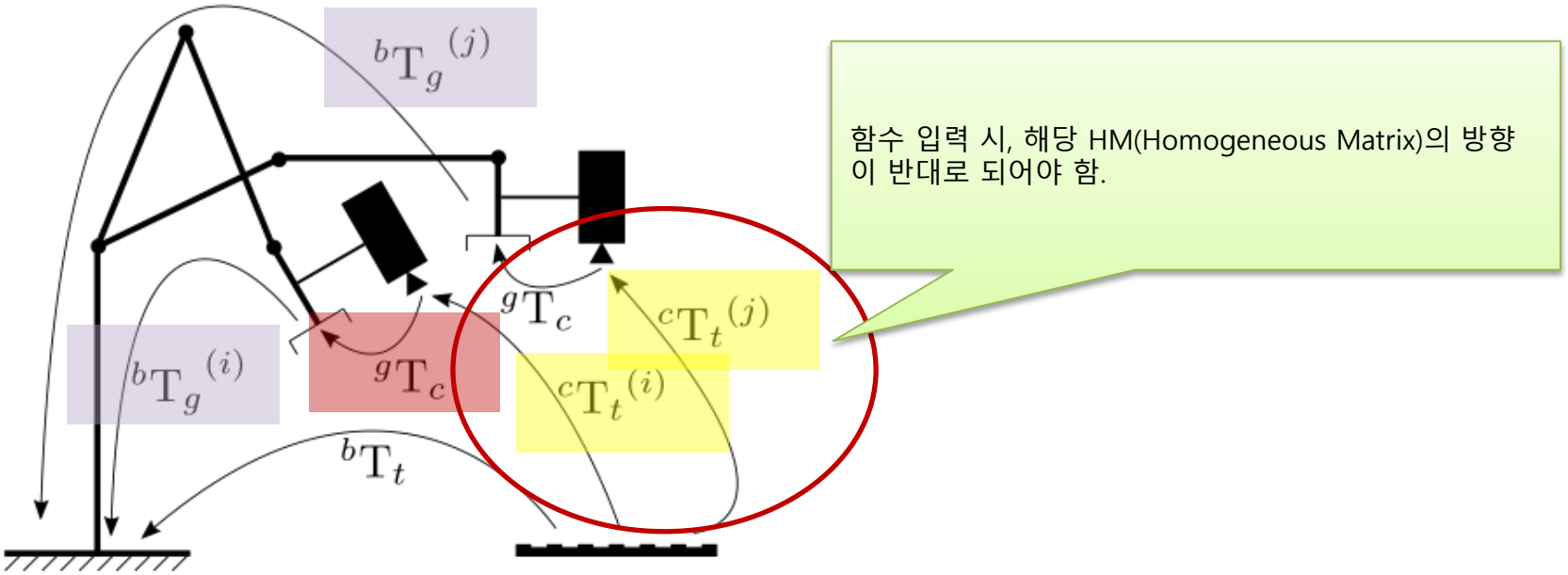
```
R_cam2gripper, t_cam2gripper = cv.calibrateHandEye( R_gripper2base, t_gripper2base, R_target2cam, t_target2cam[, R_cam2gripper[, t_cam2gripper[, method]]] )
```



#2 OpenCV calibrateHandeye 적용

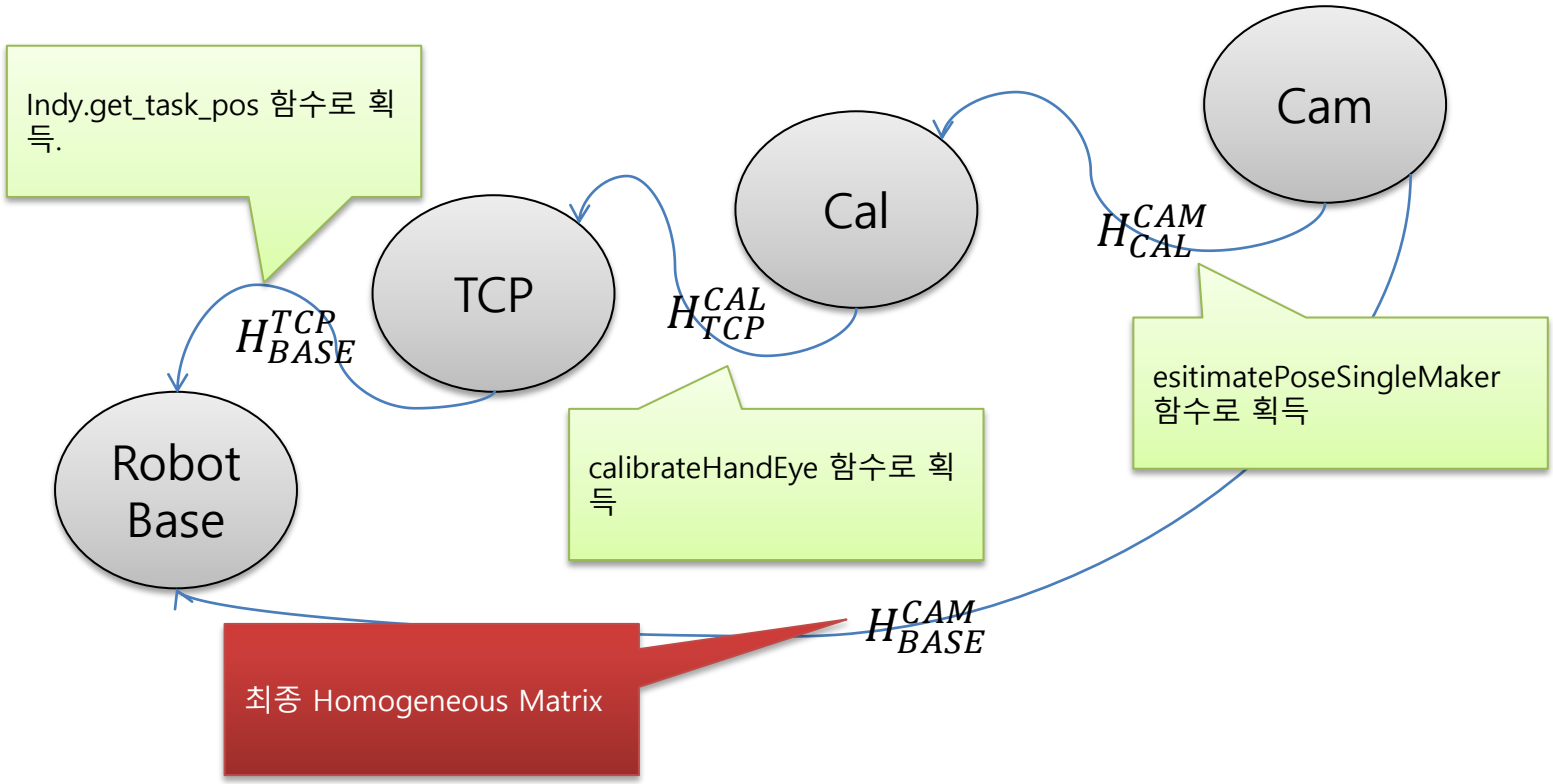
OpenCV calibrateHandEye 함수를 Hand-to-Eye에 적용하기 위해서는 Camera의 위치와 Calibration Panel이 위치가 변경되어야 함.

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