1/ Get the camera pose relative to the marker using the solvePnP function (code below) which returns a 3d transformation from the marker coordinate system to the camera coordinate system. It is specified by rotation and translation vectors

[cv::Mat](https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html) cameraMatrix, distCoeffs;

// You can read camera parameters from tutorial\_camera\_params.yml

readCameraParameters(cameraParamsFilename, cameraMatrix, distCoeffs); // This function is implemented in aruco\_samples\_utility.hpp

std::vector<cv::Vec3d> rvecs, tvecs;

// Set coordinate system

[cv::Mat](https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html) objPoints(4, 1, [CV\_32FC3](https://docs.opencv.org/4.x/d1/d1b/group__core__hal__interface.html#ga0610d99405b809062622588c25ed5c8f));

...

// Calculate pose for each marker

for (int i = 0; i < nMarkers; i++) {

[solvePnP](https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d)(objPoints, corners.at(i), cameraMatrix, distCoeffs, rvecs.at(i), tvecs.at(i));

}

2/ You simply reverse the pose returned by the rvecs and tvecs to get the position and orientation of the marker relative to the camera, instead of the camera relative to the marker. The code is pretty easy.

Mat R;

Rodrigues(rvecs.at(i), R);

R = R.t();

tvecs.at(i) = -R\*tvecs.at(i);

Rodrigues(R, rvecs.at(i));

import cv2

import cv2.aruco as aruco

import numpy as np

def main():

# Define Aruco dictionary and parameters

aruco\_dict = aruco.Dictionary\_get(aruco.DICT\_4X4\_50)

aruco\_params = aruco.DetectorParameters\_create()

# Load camera parameters

camera\_params\_file = "tutorial\_camera\_params.yml"

with open(camera\_params\_file, 'r') as f:

camera\_params\_data = yaml.load(f)

camera\_matrix = np.array(camera\_params\_data['camera\_matrix']['data']).reshape((3, 3))

dist\_coeffs = np.array(camera\_params\_data['distortion\_coefficients']['data'])

# Set object points for the marker

obj\_points = np.zeros((4, 3), dtype=np.float32)

obj\_points[:, :2] = np.array([[0, 0], [0, 1], [1, 1], [1, 0]], dtype=np.float32)

# Initialize video capture from the camera

cap = cv2.VideoCapture(0)

while not rospy.is\_shutdown():

# Read a frame from the camera

ret, frame = cap.read()

if not ret:

rospy.logwarn("Failed to read frame from the camera!")

break

# Detect the Aruco marker

corners, ids, \_ = aruco.detectMarkers(frame, aruco\_dict, parameters=aruco\_params)

if ids is not None and len(ids) > 0:

# Draw the detected marker on the frame

aruco.drawDetectedMarkers(frame, corners, ids)

# Calculate the pose of the marker relative to the camera

rvecs, tvecs = None, None

ret, rvecs, tvecs = cv2.solvePnP(obj\_points, corners[0], camera\_matrix, dist\_coeffs, rvecs, tvecs)

if ret:

# Reverse the pose to get the position and orientation of the marker relative to the camera

R, \_ = cv2.Rodrigues(rvecs)

R = R.T

tvecs = -R.dot(tvecs)

# Print the marker pose

rospy.loginfo("Marker pose relative to the camera:")

rospy.loginfo("Rotation: {}".format(R))

rospy.loginfo("Translation: {}".format(tvecs))

# Display the frame with detected marker

cv2.imshow("Frame", frame)

cv2.waitKey(1)

# Release the video capture and close the window

cap.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

main()

3/ Lay the (0,0,0) of the marker in the place you want the (0,0,0) of the world. Then keep that detected pose in a variable because it's the information you need to go from world (which is for this picture the same as the marker) to camera. You'll need to reverse it to go from camera to world (same method as above).

4/ Then you can take your detected pose in the live system to go marker -> camera, and use the stored to go camera -> world. That gives you a chain of marker -> world, which is what you want.