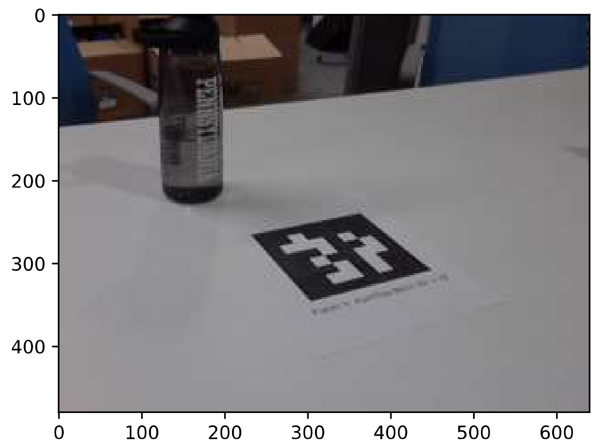
In [1]:

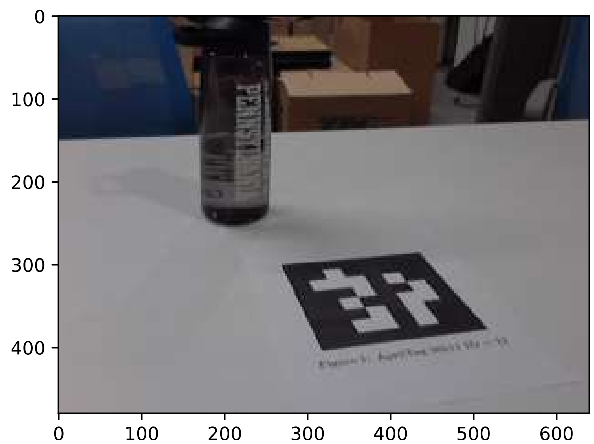
```
%matplotlib inline

""" Forces colab to use the correct version of opencv, sets up matlab, imports
!pip install opencv-contrib-python==4.3.0.38
"""

import matplotlib
import matplotlib.pyplot as plt
import cv2
import numpy as np
from numpy.linalg import svd as svd
from numpy.linalg import norm as norm
from numpy.linalg import inv as inv
from numpy.linalg import det as det
from numpy.linalg import pinv as pinv
```

In [2]:

```
"""  This loads the images.  You must first upload the images to your colab
session.  Each time you start a new session, you will need to upload them again.
"""
im_left = cv2.imread('../imgs/set3_1.jpg')
print("image shape:", im_left.shape)
plt.imshow(im_left[:, :, ::-1])
plt.figure()
im_right = cv2.imread('../imgs/set3_2.jpg')
plt.imshow(im_right[:, :, ::-1])
images = [im_left, im_right]
```

```
image shape: (480, 640, 3)
```

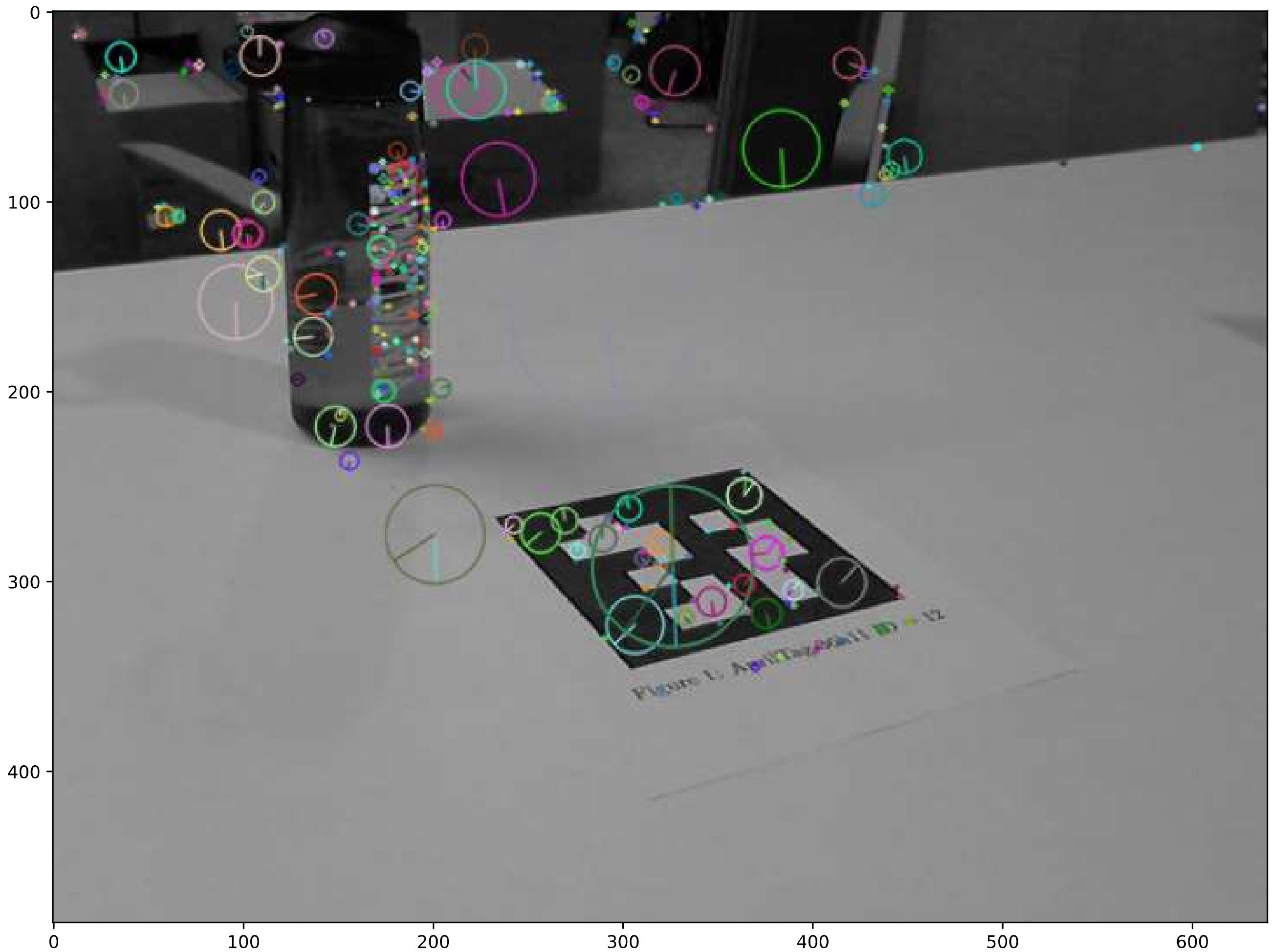


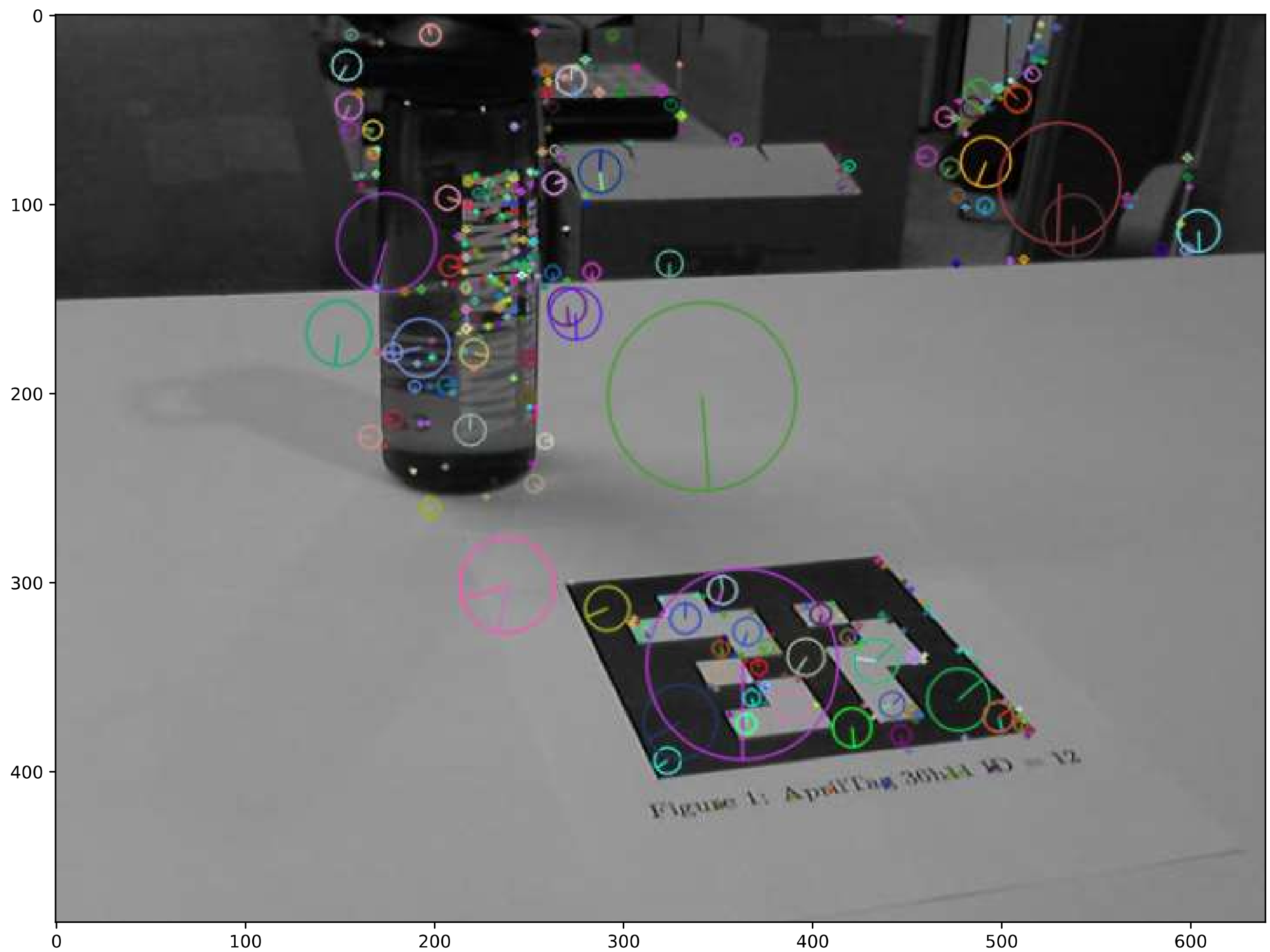


In [3]:

```
""" Detects SIFT features in all of the images
"""
keypoints = []
descriptions = []
for im in images:
  gray= cv2.cvtColor(im,cv2.COLOR_RGB2GRAY)

  sift = cv2.xfeatures2d.SIFT_create()
  kp, des = sift.detectAndCompute(gray,None)

  keypoints.append(kp)
  descriptions.append(des)

  plt.figure(figsize=(6.4*2, 4.8*2))
  out_im = cv2.drawKeypoints(gray,kp, gray, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

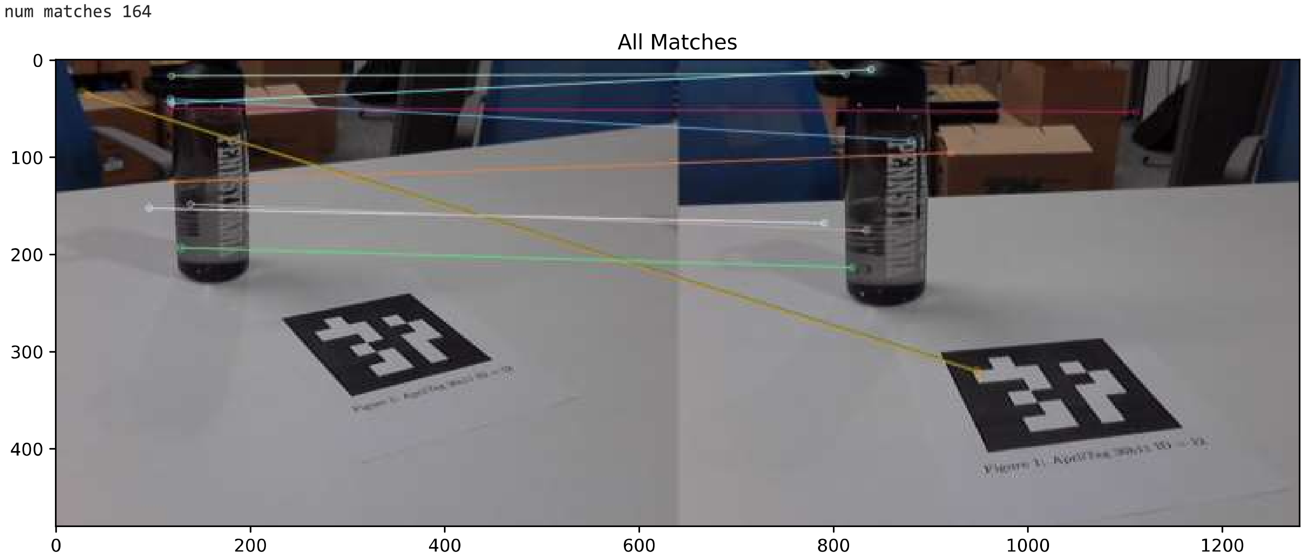
```
plt.imshow(out_im)
```





In [4]:
```python
""" Matches the detected keypoints between the images
"""

bf = cv2.BFMatcher(crossCheck=True)
matches = bf.match(descriptions[0], descriptions[1])

print("num matches", len(matches))

matched_image = cv2.drawMatches(images[0][:, :, ::-1], keypoints[0], images[1][:, :, ::-1], keypoints[1], matches[:10], None, flags=2)
plt.figure(figsize=(6.4*2, 4.8*2))
plt.title("All Matches")
plt.imshow(matched_image)
plt.show()
```

```
num matches 164
```



In [5]:
```python
""" Compute calibrated coordinates
f = 552
```

```
u0 = 307.5
v0 = 205
"""
f = 823.8
u0 = 304.8
v0 = 236.3

K = np.array([[f, 0, u0],
              [0, f, v0],
              [0, 0, 1]])


uncalibrated_1 = [[keypoints[0][match.queryIdx].pt[0], keypoints[0][match.queryIdx].pt[1], 1] for match in matches]
uncalibrated_2 = [[keypoints[1][match.trainIdx].pt[0], keypoints[1][match.trainIdx].pt[1], 1] for match in matches]

uncalibrated_1 = np.array(uncalibrated_1).T
uncalibrated_2 = np.array(uncalibrated_2).T

k_inv = np.linalg.inv(K)

calibrated_1 = np.matmul(k_inv, uncalibrated_1).T
calibrated_2 = np.matmul(k_inv, uncalibrated_2).T
```

In [6]:

```
def least_squares_estimation(X1, X2):
  """ YOUR CODE HERE
  """
  A = X1[:, 0] * X2.T
  B = X1[:, 1] * X2.T
  C = X1[:, 2] * X2.T

  mat = np.hstack((A.T, B.T, C.T))
  U, S, V = svd(mat)

  E = (V.T)[:, -1].reshape(3, 3)

  U, S, V = svd(E)
  d = np.diag([1, 1, 0])
  E = (U @ d @ V).T

  """ END YOUR CODE
  """
  return E

E_least = least_squares_estimation(calibrated_1, calibrated_2)
print("E least")
print(np.around(E_least, 2))
```

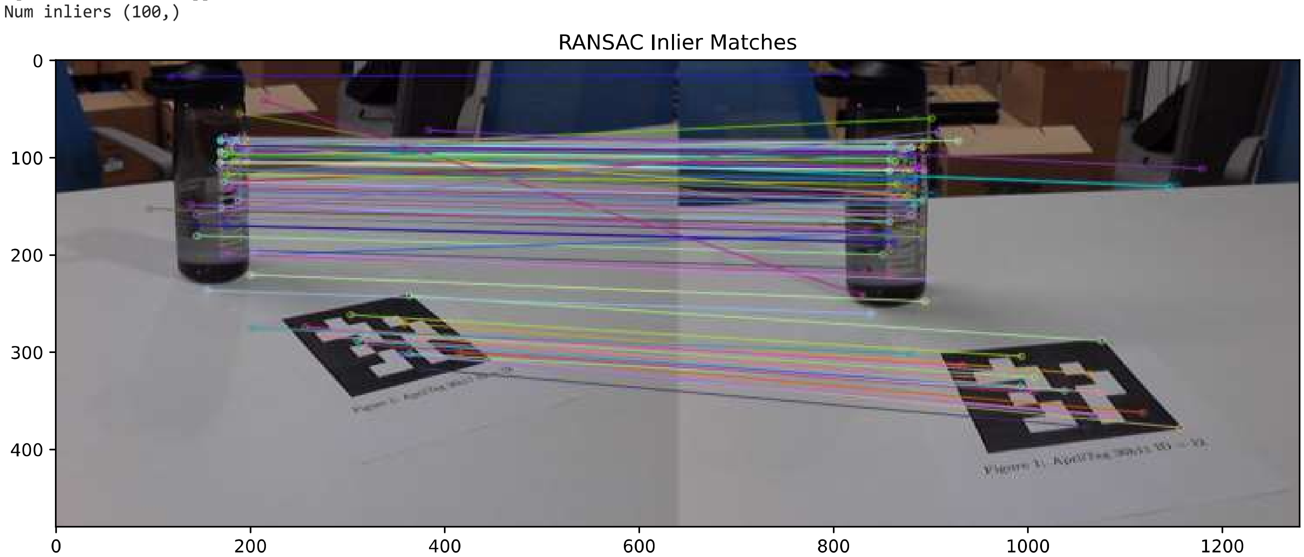
```
E least
[[-0.08  0.97 -0.04]
 [-0.93 -0.13 -0.27]
 [-0.21  0.2  -0.06]]
```

In [7]:

```
def ransac_estimator(X1, X2):
  num_iterations = 80000 # 20000
  sample_size = 8

  eps = 10**-4

  best_num_inliers = -1
  best_inliers = None
  best_E = None

  for _ in range(num_iterations):
    permuted_indices = np.random.permutation(np.arange(X1.shape[0]))
    sample_indices = permuted_indices[:sample_size]
    test_indices = permuted_indices[sample_size:]

    """ YOUR CODE HERE
    """
    inlier_arr = []

    x1_sample = X1[sample_indices]
    x2_sample = X2[sample_indices]

    x1_test = X1[test_indices].T
    x2_test = X2[test_indices].T

    E = least_squares_estimation(x1_sample, x2_sample)

    for i in range(len(test_indices)):
      residual_arr = []

      E_X1 = E @ x1_test[:, i]
      residual_num = (x2_test[:, i] @ E).T @ x1_test[:, i]
      residual_den = norm(E_X1[:2])
      residual_arr.append(residual_num**2 / residual_den)

      E_X2 = E.T @ x2_test[:, i]
      residual_num = (x1_test[:, i].T @ E.T) @ x2_test[:, i]
```

```
      residual_den = norm(E_X2[:2])
      residual_arr.append(residual_num**2 / residual_den)

      if sum(residual_arr) < eps: inlier_arr.append(test_indices[i])

    inliers = np.array(inlier_arr).T

    """ END YOUR CODE
    """
    if inliers.shape[0] > best_num_inliers:
      best_num_inliers = inliers.shape[0]
      best_E = E
      best_inliers = inliers


  return best_E, best_inliers
E_ransac, inliers = ransac_estimator(calibrated_1, calibrated_2)
print("E_ransac")
print(np.around(E_ransac, 2))
print("Num inliers", inliers.shape)

inlier_matches = [matches[i] for i in inliers]

matched_image = cv2.drawMatches(images[0][:, :, ::-1],
                                keypoints[0],
                                images[1][:, :, ::-1],
                                keypoints[1],
                                inlier_matches, None, flags=2)
plt.figure(figsize=(6.4*2, 4.8*2))
plt.title("RANSAC Inlier Matches")
plt.imshow(matched_image)
plt.show()
```

```
E_ransac
[[ 0.09  0.98  0.17]
 [-0.98  0.1  -0.12]
 [-0.13  0.09 -0.   ]]
Num inliers (100,)
```



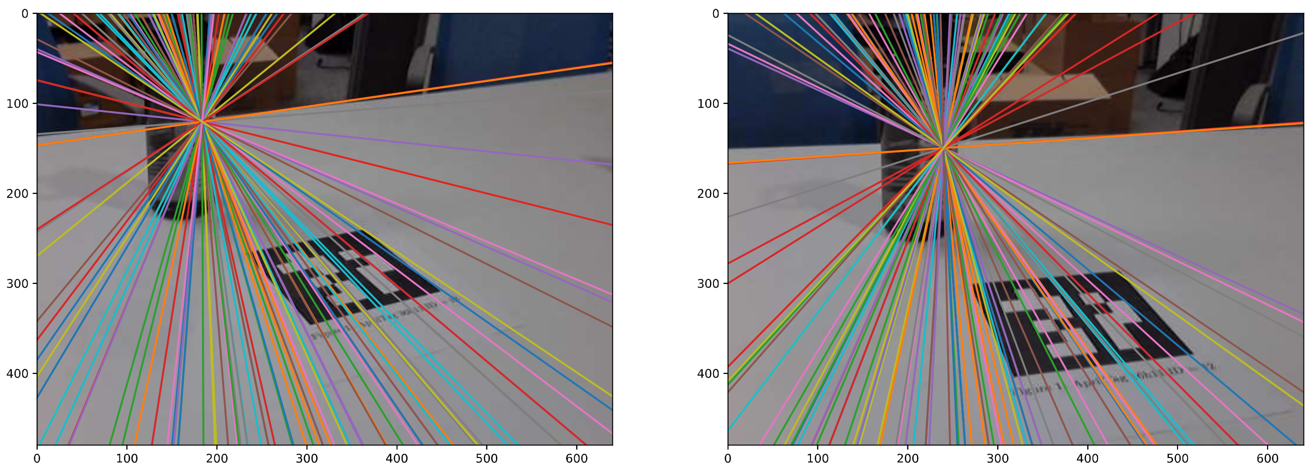
In [113…

```
def plot_lines(lines, h, w):
  """ Utility function to plot lines
  """

  for i in range(lines.shape[1]):
    if abs(lines[0, i] / lines[1, i]) < 1:
      y0 = -lines[2, i] / lines[1, i]
      yw = y0 - w * lines[0, i] / lines[1, i]
      plt.plot([0, w], [y0, yw])
    else:
      x0 = -lines[2, i] / lines[0, i]
      xh = x0 - h * lines[1, i] / lines[0, i]
      plt.plot([x0, xh], [0, h])
```

In [114…

```
def plot_epipolar_lines(image1, image2, uncalibrated_1, uncalibrated_2, E, K):
  """ Plots the epipolar lines on the images
  """

  """ YOUR CODE HERE
  """
  fundamental_mat = inv(K).T @ E @ inv(K)
  epipolar_lines_in_1 = fundamental_mat.T @ uncalibrated_2
  epipolar_lines_in_2 = fundamental_mat @ uncalibrated_1
  """ END YOUR CODE
  """
```

```
  plt.figure(figsize=(6.4*3, 4.8*3))
  ax = plt.subplot(1, 2, 1)
  ax.set_xlim([0, image1.shape[1]])
  ax.set_ylim([image1.shape[0], 0])
  plt.imshow(image1[:, :, ::-1])
  plot_lines(epipolar_lines_in_1, image1.shape[0], image1.shape[1])

  ax = plt.subplot(1, 2, 2)
  ax.set_xlim([0, image1.shape[1]])
  ax.set_ylim([image1.shape[0], 0])
  plt.imshow(image2[:, :, ::-1])
  plot_lines(epipolar_lines_in_2, image2.shape[0], image2.shape[1])
  plt.show()


uncalibrated_inliers_1 = [[keypoints[0][match.queryIdx].pt[0], keypoints[0][match.queryIdx].pt[1], 1] for match in inlier_matches]
uncalibrated_inliers_2 = [[keypoints[1][match.trainIdx].pt[0], keypoints[1][match.trainIdx].pt[1], 1] for match in inlier_matches]
uncalibrated_inliers_1 = np.array(uncalibrated_inliers_1).T
uncalibrated_inliers_2 = np.array(uncalibrated_inliers_2).T

plot_epipolar_lines(images[0], images[1], uncalibrated_inliers_1, uncalibrated_inliers_2, E_ransac, K)
```



In [115…

```
def pose_canidates_from_E(E):
  transform_canidates = []
  """ YOUR CODE HERE
  """
  rot_array = []
  trans_array = []

  ## First Pose Candidate
  U, S, V = svd(E)
  positive_Rz = np.diag([0, 0, 1])
  positive_Rz[0, 1] = -1
  positive_Rz[1, 0] = 1

  rotation = U @ positive_Rz.T @ V
  if det(rotation) < 0:
    positive_Rz[2, 2] = det(U @ V)
    rotation = U @ positive_Rz.T @ V

  rot_array.append(rotation)

  translation = U[:, 2]
  trans_array.append(translation)

  ## Second Pose Candidate
  negative_Rz = np.diag([0, 0, 1])
  negative_Rz[0, 1] = 1
  negative_Rz[1, 0] = -1

  rotation = U @ negative_Rz.T @ V
  if det(rotation) < 0:
    negative_Rz[2, 2] = det(U @ V)
    rotation = U @ negative_Rz.T @ V

  rot_array.append(rotation)

  translation = -U[:, 2]
  trans_array.append(translation)

  ## Third Pose Candidate
  U, S, V = svd(-E)
  positive_Rz = np.diag([0, 0, 1])
  positive_Rz[0, 1] = -1
  positive_Rz[1, 0] = 1
```

```
    rotation = U @ positive_Rz.T @ V
    if det(rotation) < 0:
      positive_Rz[2, 2] = det(U @ V)
      rotation = U @ positive_Rz.T @ V

    rot_array.append(rotation)

    translation = U[:, 2]
    trans_array.append(translation)

    ## Fourth Pose Candidate
    negative_Rz = np.diag([0, 0, 1])
    negative_Rz[0, 1] = 1
    negative_Rz[1, 0] = -1

    rotation = U @ negative_Rz.T @ V
    if det(rotation) < 0:
      negative_Rz[2, 2] = det(U @ V)
      rotation = U @ negative_Rz.T @ V

    rot_array.append(rotation)

    translation = -U[:, 2]
    trans_array.append(translation)

    ## Create dictionary of pose candidates
    for i in range(4):
      candidate = {}
      candidate['R'] = rot_array[i]
      candidate['T'] = trans_array[i]
      transform_canidates.append(candidate)

    """ END YOUR CODE
    """
    return transform_canidates

transform_canidates = pose_canidates_from_E(E_ransac)
print("transform_canidates")
print(transform_canidates)
```

```
transform_canidates
[{'R': array([[-0.94035121,  0.11729137, -0.31934672],
       [-0.03284848, -0.96560586, -0.25792694],
       [-0.33861567, -0.23205186,  0.91186148]]), 'T': array([-0.12147593, -0.09899498,  0.98764548])}, {'R': array([[ 0.99305971, -
0.08137266,  0.08491707],
       [ 0.07580248,  0.99487727,  0.06688202],
       [-0.08992443, -0.05998091,  0.99414078]]), 'T': array([ 0.12147593,  0.09899498, -0.98764548])}, {'R': array([[ 0.99305971, -
0.08137266,  0.08491707],
       [ 0.07580248,  0.99487727,  0.06688202],
       [-0.08992443, -0.05998091,  0.99414078]]), 'T': array([-0.12147593, -0.09899498,  0.98764548])}, {'R': array([[-0.94035121,
0.11729137, -0.31934672],
       [-0.03284848, -0.96560586, -0.25792694],
       [-0.33861567, -0.23205186,  0.91186148]]), 'T': array([ 0.12147593,  0.09899498, -0.98764548])}]
```

In [116…

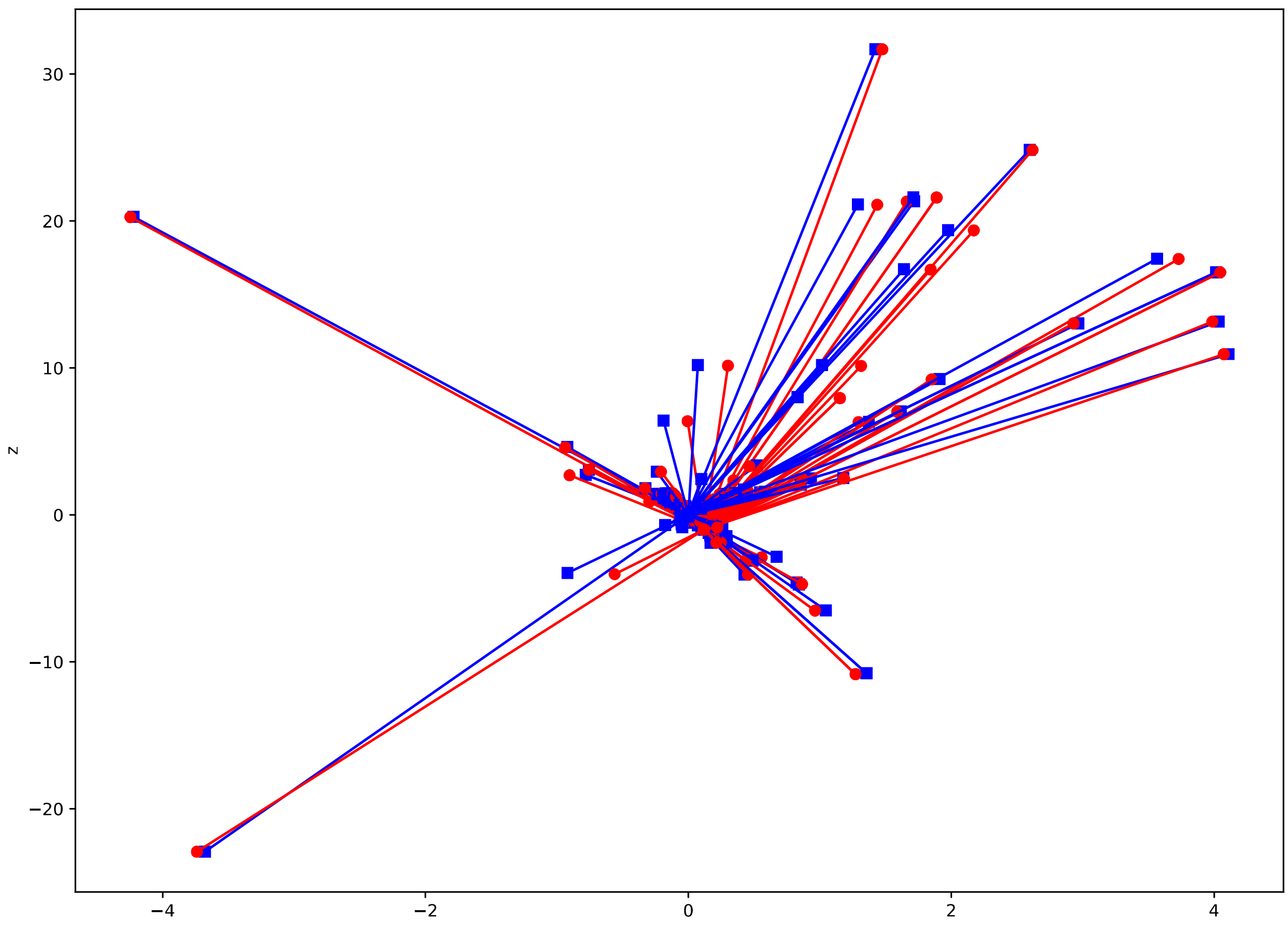
```
def plot_reconstruction(P1, P2, T, R):
    P1trans = (R @ P1.T).T + T

    plt.figure(figsize=(6.4*2, 4.8*2))
    ax = plt.axes()
    ax.set_xlabel('x')
    ax.set_ylabel('z')
    plt.plot([0], [0], 'bs')
    plt.plot([T[0]], [T[2]], 'ro')

    for i in range(P1.shape[0]):
      plt.plot([0, P2[i, 0]], [0, P2[i, 2]], 'bs-')
      plt.plot([T[0], P1trans[i, 0]], [T[2], P1trans[i, 2]], 'ro-')
```

In [118…

```
def reconstruct3D(transform_canidates, calibrated_1, calibrated_2):
    """This functions selects (T,R) among the 4 candidates transform_canidates
    such that all triangulated points are in front of both cameras.
    """

    best_num_front = -1
    best_canidate = None
    best_lambdas = None
    for canidate in transform_canidates:
      R = canidate['R']
      T = canidate['T']

      lambdas = np.zeros((2, calibrated_1.shape[0]))
      """ YOUR CODE HERE
      """
      for i in range(calibrated_1.shape[0]):
        calibrated_rot = -R @ calibrated_1[i]
        lambda_i = pinv(np.vstack((calibrated_2[i], calibrated_rot)).T) @ T
        lambdas[:, i] = lambda_i

      """ END YOUR CODE
```

```
    """
    num_front = np.sum(np.logical_and(lambdas[0]>0, lambdas[1]>0))

    if num_front > best_num_front:
      best_num_front = num_front
      best_canidate = canidate
      best_lambdas = lambdas
      print("best", num_front, best_lambdas[0].shape)
    else:
      print("not best", num_front)


  P1 = best_lambdas[1].reshape(-1, 1) * calibrated_1
  P2 = best_lambdas[0].reshape(-1, 1) * calibrated_2
  T = best_canidate['T']
  R = best_canidate['R']
  return P1, P2, T, R


P1, P2, T, R = reconstruct3D(transform_canidates, calibrated_1, calibrated_2)


plot_reconstruction(P1, P2, T, R)
```

```
best 4 (164,)
best 101 (164,)
not best 29
not best 8
```



In [17]:

```
def show_reprojections(image1, image2, uncalibrated_1, uncalibrated_2, P1, P2, K, T, R):

  """ YOUR CODE HERE
  """
  P1proj = np.zeros((3, P1.shape[0]))
  P2proj = np.zeros(P1proj.shape)

  for i in range(P1.shape[0]):
    P1proj[:, i] = K @ ((R @ P1[i].T) + T )
    P2proj[:, i] = K @ ((R @ P2[i].T) + T )

  P1proj = P1proj.T
  P2proj = P2proj.T
  """ END YOUR CODE
  """
```

```python
    plt.figure(figsize=(6.4*3, 4.8*3))
    ax = plt.subplot(1, 2, 1)
    ax.set_xlim([0, image1.shape[1]])
    ax.set_ylim([image1.shape[0], 0])
    plt.imshow(image1[:, :, ::-1])
    plt.plot(P2proj[:, 0] / P2proj[:, 2],
             P2proj[:, 1] / P2proj[:, 2], 'bs')
    plt.plot(uncalibrated_1[0, :], uncalibrated_1[1, :], 'ro')

    ax = plt.subplot(1, 2, 2)
    ax.set_xlim([0, image1.shape[1]])
    ax.set_ylim([image1.shape[0], 0])
    plt.imshow(image2[:, :, ::-1])
    plt.plot(P1proj[:, 0] / P1proj[:, 2],
             P1proj[:, 1] / P1proj[:, 2], 'bs')
    plt.plot(uncalibrated_2[0, :], uncalibrated_2[1, :], 'ro')

show_reprojections(images[0], images[1], uncalibrated_1, uncalibrated_2, P1, P2, K, T, R)
```