Lab assignment 7

ETH Zurich, Computer Science Department

# Computer vision

# Report
# Structure from Motion and RANSAC

*Author*
Emilia Szymańska, 22-945-547

December 2022

The objective of this lab assignment was to implement a Structure from Motion pipeline and RANSAC line fitting.

# 1 Structure from Motion

## 1.1 Essential matrix estimation

The first step is to lift the keypoints to the normalized image plane first – I make the points homogeneous by adding ones as the last coordinates, calculate the inverse matrix of K and multiply it by homogeneous keypoints (transposed to match the dimensions of K matrix for multiplication, then I transpose normalized keypoints back to their initial dimensions). The next step is to create a constraint matrix to reformulate the constraint $\hat{x}_1^T E \hat{x}_2 = 0$. I do that by adding rows corresponding to each keypoint match in a form of $[x_2x_1, x_2y_1, x_2z_1, y_2x_1, y_2y_1, y_2z_1, x_1z_2, y_1z_2, z_1z_2]$, where in fact Z coordinates should be equal to 1. Then, the Singular Value Decomposition is applied to the constraint matrix and the last row of obtained $V^H$ matrix contains elements of $\hat{E}$. This vector has to be reshaped to be a matrix 3x3. We need to fulfill the essential matrix constraints that the third singular value is equal to 0 and the first to are equal and greater than 0 (in our case we can assume them as 1). Therefore, SVD needs to be applied to $\hat{E}$ and obtained $U$ and $V^H$ elements are multiplied by $S$, which is a diagonal matrix with 1, 1, 0 on its diagonal.

The code implementation of this part is presented below:

```
def EstimateEssentialMatrix(K, im1, im2, matches):
  K_inv = np.linalg.inv(K)

  kpts1 = MakeHomogeneous(im1.kps, ax=1)
  kpts2 = MakeHomogeneous(im2.kps, ax=1)

  normalized_kps1 = K_inv @ kpts1.T
  normalized_kps2 = K_inv @ kpts2.T
  normalized_kps1 = normalized_kps1.T
  normalized_kps2 = normalized_kps2.T

  constraint_matrix = np.zeros((matches.shape[0], 9))
  for i in range(matches.shape[0]):
    x1,y1,z1 = normalized_kps1[i][0],normalized_kps1[i][1],normalized_kps1[i][2]
    x2,y2,z2 = normalized_kps2[i][0],normalized_kps2[i][1],normalized_kps2[i][2]
    A_i = np.array([x2*x1,x2*y1,x2*z1,y2*x1,y2*y1,y2*z1,x1*z2,y1*z2,z1*z2])
    constraint_matrix[i] = A_i

  _, _, vh = np.linalg.svd(constraint_matrix)
  vectorized_E_hat = vh[-1,:]

  E_hat = vectorized_E_hat.reshape((3,3))

  U_hat, _, VH_hat = np.linalg.svd(E_hat)
  s = np.array([1, 1, 0])
  S_hat = np.diag(s)
  E = U_hat @ S_hat @ VH_hat

  return E
```

## 1.2 Next steps

I did not manage to overcome some of the problems and therefore did not continue on Structure from Motion task.

# 2 RANSAC

## 2.1 Least-squares solution

To implement the least square solution, I first (as suggested by the linalg.lstsq documentation) rewrite the problem from $y = kx + b$ to $y = Ap$, where $A = [[x1]]$ and $p = [[k], [b]]$. Then I use the lstsq to solve for p as presented in the code:

```
def least_square(x,y):
    A = np.vstack([x, np.ones(len(x))]).T
    k, b = np.linalg.lstsq(A, y, rcond=None)[0]
    return k, b
```

## 2.2 RANSAC implementation

First I use the np.choice function to randomly choose a subset of both x and y point sets. Then I compute the least-squares solution for this subset and with obtained $k, b$ I calculate the number of inliers and the mask that denotes the indices of inliers (a point is an inlier if its distance to the line smaller than thres_dist, which is compared in num_inlier function). If the number of inliers is larger than the current best result, I update the parameters k_ransac, b_ransac, best_inliers and inlier_mask. The code corresponding to this part can be found below:

```
def num_inlier(x,y,k,b,n_samples,thres_dist):
    num = 0
    mask = np.zeros(x.shape, dtype=bool)
    for i in range(n_samples):
        dist = abs(k*x[i] - y[i] + b)/math.sqrt(k*k + 1)
        if dist < thres_dist:
            num += 1
            mask[i] = True

    return num, mask

def ransac(x,y,iter,n_samples,thres_dist,num_subset):
    k_ransac = None
    b_ransac = None
    inlier_mask = None
    best_inliers = 0

    for _ in range(iter):
        x_subset = np.random.choice(x, size=num_subset)
        y_subset = np.random.choice(y, size=num_subset)
        k, b = least_square(x_subset,y_subset)
        num_inl, mask = num_inlier(x, y, k, b, n_samples, thres_dist)
        if num_inl > best_inliers:
            best_inliers = num_inl
            inlier_mask = mask
            k_ransac = k
            b_ransac = b

    return k_ransac, b_ransac, inlier_mask
```

## 2.3 Results

After running RANSAC, the results presented in Fig.1 are obtained. The plot corresonds to:

- true coefficients: $k = 1$, $b = 10$,
- linear regression coefficients: $k = 0.615965$, $b = 8.961727$,
- RANSAC coefficients: $k = 0.936744$, $b = 9.860836$.

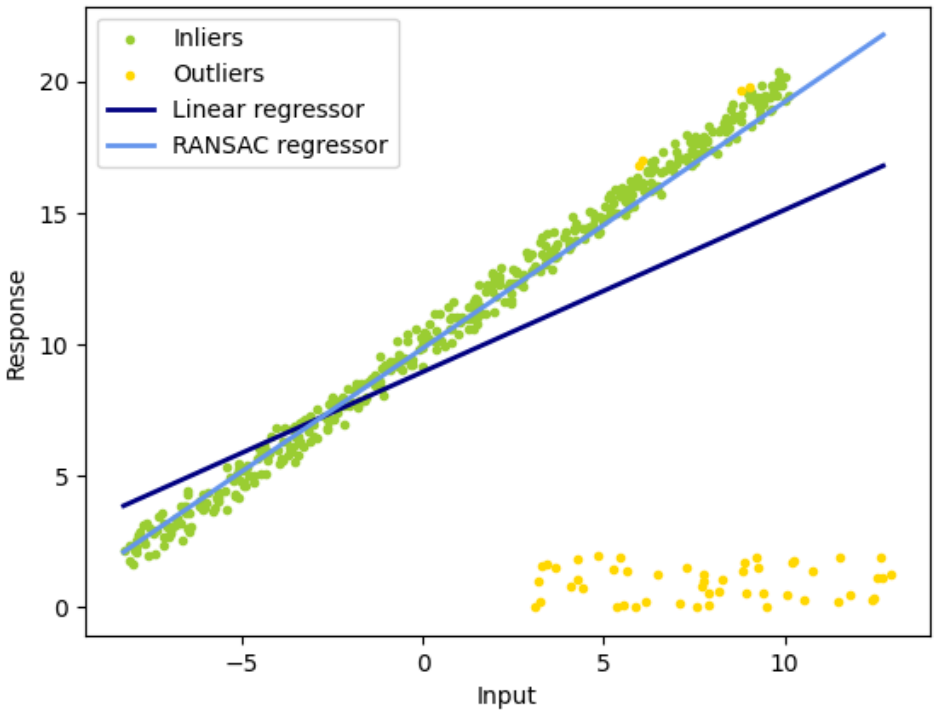


Figure 1: The result of running RANSAC for line fitting.