Mobile Robotics Assignment 1

Team ID 24

Roll No. 2019702002 Nivedita Rufus

Roll No. 2019702008 Sravya Vardhani S

Q1) Lidar-Camera fusion

Approach:

The Lidar points are given it its frame. Our task was to map those points on to the image. This can be achieved in two steps:

- Get the points given in the lidar frame to its corresponding coordinates into camera frame.
- Map the points onto the image plane.

The first step can be achieved through the relationship given between the axes of the lidar and camera frame, i.e. **90**^{0} clockwise rotation about the z-axis of lidar, followed by **90**^{0} clockwise rotation about the x-axis, the translational distance is already given.

$$ {}^{A}P = {}^{A}_{B}R\ {}^{B}P + {}^{A}P_{BORG}. $$

Using the relation given above, we can deduce the points in the camera frame.

Code snippet :

```
def get_points_camera_frame(lidar_points,rot):
      cam_pts = []
      for point in lidar_points:
            p = np.reshape(point,(3,1))
            p = np.dot(rot,p)
```

```
            p = p + np.array([[0.27],[0.06],[-.08]])
            p = p.ravel()
            cam_pts.append(p)
      cam_pts = np.asarray(cam_pts)
      return cam_pts
```

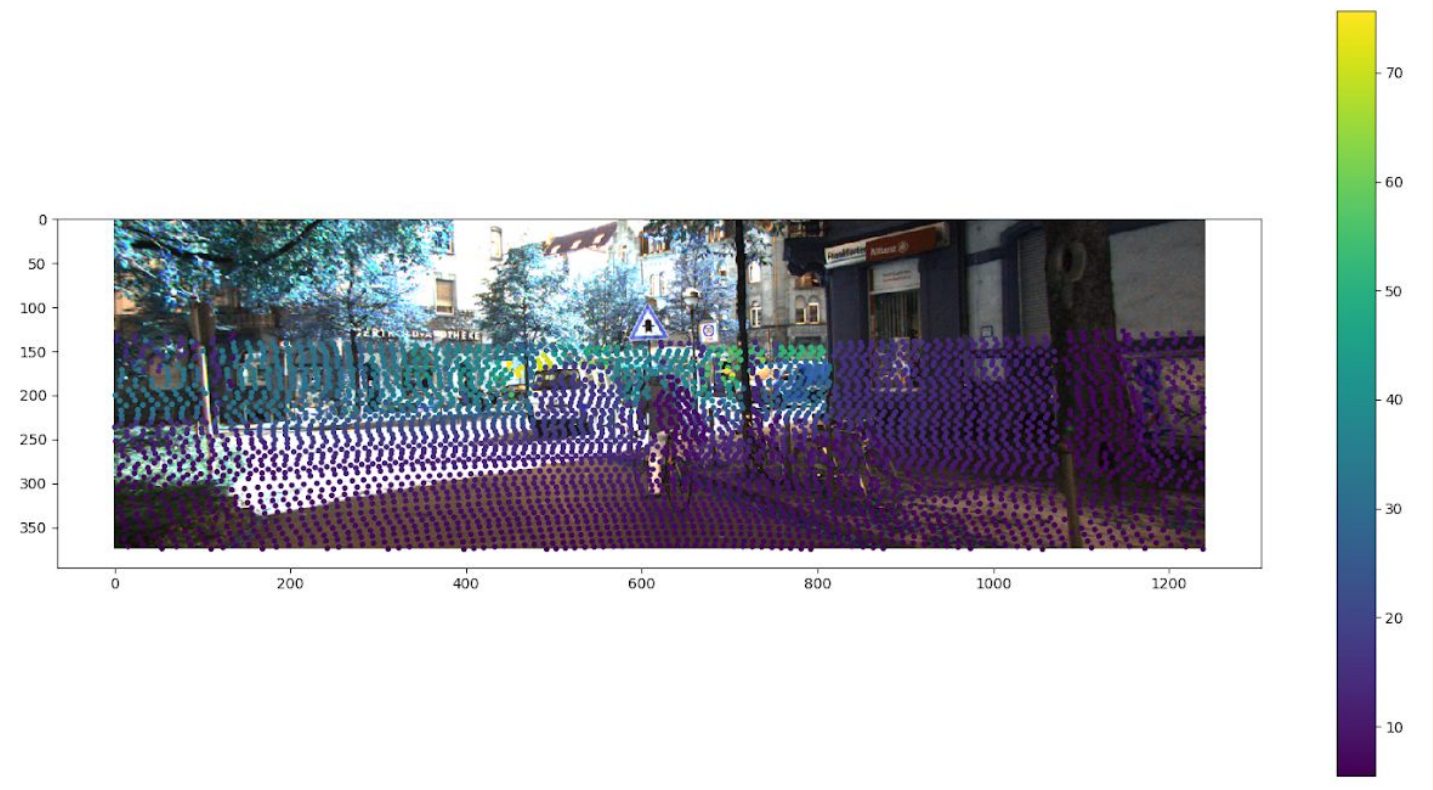
To achieve the second step, we use the following relation using the K matrix :

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Code snippet:

```
def get_image_points(points, K):
      img_pts =[]
      z=[]
      for point in points:
            p = np.reshape(point,(3,1))
            p = np.dot(K,p)
            p[0]= p[0]/p[2]
            p[1]= p[1]/p[2]
            x = p[2]
            p = p.ravel()
            if p[0]<=1240 and p[1]<=375 and p[0]>=0:
                  img_pts.append(p)
                  # print(x)
                  z.append(x[0])
      img_pts = np.asarray(img_pts)
      return img_pts,list(z)
```

Output :



Q2) Drawing 3D bounding boxes

Approach:

In the question we have been given, the K matrix of the camera, the approximate dimensions of the car. We have also been given the approximate height of the car on which the camera is placed.

One corner of the wheel of the car is located at approximately [u, v] = [810,300] location(pixels).

By using projection equations given below we estimate the translation along the Xand Z axes respectively.

$$u = u_0 + k_u x \Rightarrow u = u_0 + \frac{k_u f X_c}{Z_c}$$
$$v = v_0 + k_v y \Rightarrow v = v_0 + \frac{k_v f Y_c}{Z_c}$$

Where, u_0 and v_0 are the optical center of the camera C_x and C_y.

Code snippet :

```
u,v = (810,300)
f = 7.2153e+02
cx = 6.0955e+02
cy = 1.7285e+02
Y = 1.65
Z = f*Y/( v-cy )
X = (u-cx)*Z/f
print(X,Y,Z)
```

Next we find the coordinates of all the points of the car. As we know the approximate dimensions of the car, this will be our world frame points. We also define our RT matrix with a slight rotation of 5 degrees and translation with the values as computed above , and multiply the same with the points of the car to obtain the points in the camera frame:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

To obtain the coordinates in the image plane we use the formula, where X_c, Y_c, Z_c are the respective points in the camera frame:

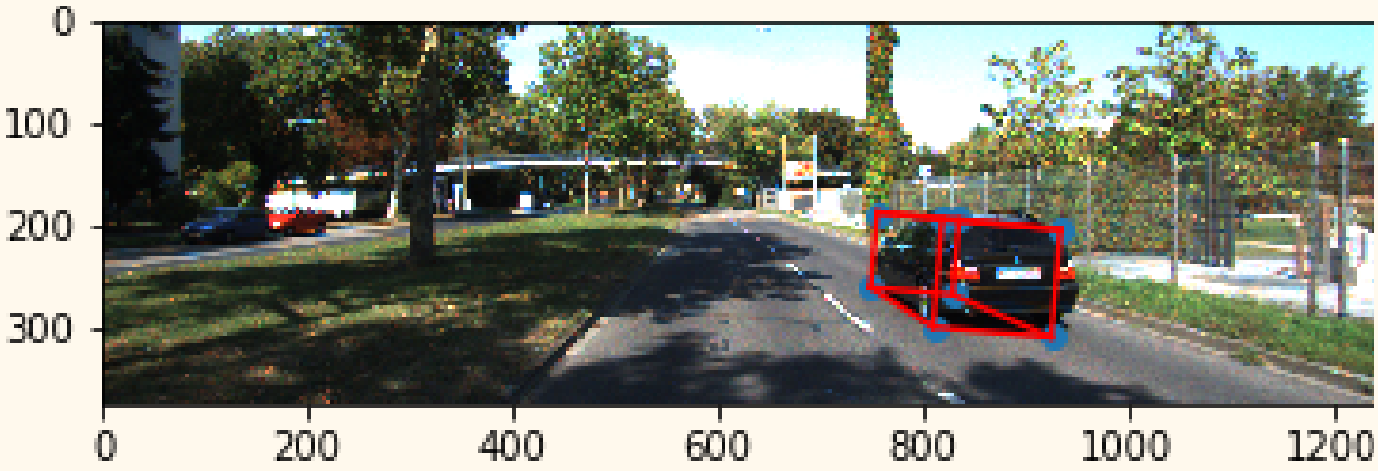
$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Code snippet :

```
#generating the other points using this point
car = np.array([[0,0,0,1],
                [1.510,0,0,1],
                [1.510,-1.380,0,1],
                [0,-1.380,0,1],
                [0,0,4.1,1],
                [1.510,0,4.1,1],
                [1.510,-1.380,4.1,1],
                [0,-1.380,4.1,1]])
Rt = [[0.99,-0.087,0,X],[0.087,0.99,0,Y],[0,0,1,Z]]
result = np.zeros([8, 3])
for i in range(8):
    temp = np.matmul(K,Rt)
    result[i] = np.matmul(temp,car[i])
print(result)
```

The obtained points are then normalised and plotted on the image to show the bounding box.

Output :



Q3) How do AprilTags work

Approach:

Our task was to find the camera pose from the given April Tag image. Since the image is a planar one, estimating the homography matrix using the given data will help us in estimation of the pose of the camera.

As mentioned above, the homography matrix is estimated using SVD (singular value decomposition).

Code snippet :

Estimation of Homography matrix:

```
def get_homography(uv, w):
      A = []
      for i in range(4):
            x,y = w[i][0], w[i][1]
            u,v = uv[i][0], uv[i][1]
            A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
            A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
      A = np.asarray(A)
```

```
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1,:] / Vh[-1,-1]
    H = L.reshape(3, 3)
    return H
```

Decomposition of the homography matrix:

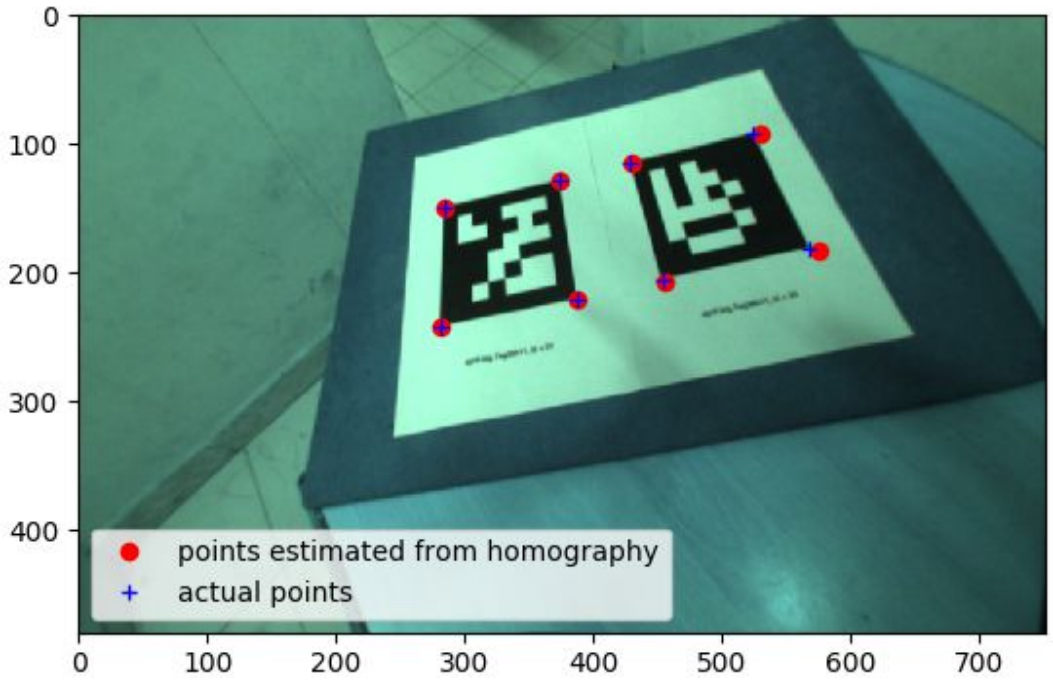
```
def PoseFromHomography(H):
    H1 = H[:, 0]
    H2 = H[:, 1]
    H3 = np.cross(H1, H2)

    norm1 = np.linalg.norm(H1)
    norm2 = np.linalg.norm(H2)
    tnorm = (norm1 + norm2) / 2.0;

    T = H[:, 2] / tnorm
    return np.mat([H1, H2, H3, T])
```

Output :

```
Camera Pose:
[[ 1.66551840e-03 -1.84565973e-04 -7.95062035e-07
-1.17050219e+02]
 [-3.26420691e-04 -1.74139676e-03 -1.91702550e-06
-1.37259309e+02]
 [-2.35911530e-04  1.17715106e-03 -2.96057449e-06
5.83593463e+02]]
```



BONUS Question:

Get position and orientation of theApril tag

To get the position of the tag, we use the following relation:

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = -R^T T + \lambda R^T K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

K, u, v are given to us, R, T, λ , was computed earlier.

Code snippet :

```
def getTagPose(rt, K, lam,uv):
     r = (rt.T)[:,0:3]
     t = (rt.T)[:,3]
     temp = np.matmul(r,t)
     temp1 = lam * np.matmul(r,np.linalg.inv(K))
     temp1 = np.matmul(temp1,uv)

     return(-temp + temp1.T)
```

To compute the orientation, we computed the plane equation of the plane passing through corner points of the tags. We used this equation to compute the normal to the plane.

Code snippet :

```
def mag(V):
  return math.sqrt(sum([x*x for x in V]))

def n(V):
  v_m = mag(V)
  return [ vi/v_m for vi in V]

def equation_plane(x1, y1, z1, x2, y2, z2, x3, y3, z3):
    a1 = x2 - x1
    b1 = y2 - y1
    c1 = z2 - z1
    a2 = x3 - x1
    b2 = y3 - y1
    c2 = z3 - z1
    a = b1 * c2 - b2 * c1
    b = a2 * c1 - a1 * c2
```

```
    c = a1 * b2 - b1 * a2
    d = (- a * x1 - b * y1 - c * z1)
    return (a,b,c)
```

Output :

```
Position:
[[ 0.16978845 -0.27563864  0.13545671]
 [ 0.17016117 -0.27563533  0.1353546 ]
 [ 0.17015946 -0.27604172  0.1356189 ]
 [ 0.16978673 -0.27604504  0.135721  ]
 [ 0.17037922 -0.27563023  0.13529283]
 [ 0.17042461 -0.27602219  0.13553458]
 [ 0.17073421 -0.27602669  0.13545504]
 [ 0.17065081 -0.27569331  0.13526143]]

orientation of Tag is along the following unit vector:
[-0.21944283246269997, -0.5312450369079108, -0.8183052939102006]
```

Roles :

Nivedita Rufus : Q3, Q1

Sravya Vardhani S: Q2, report

NOTE: The roles are not clear distinction, we had discussed everything together before execution.