Assignment 1

1) LiDAR Feature Coordinate Transformation

1. Provide the full transformation matrix $\mathbf{T}_{BL}$. Please round values to 3 decimal points. The elementary rotation matrices are provided below. (14 pts)

$$\mathbf{C}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \mathbf{C}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \mathbf{C}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The Tait-Bryan extrinsic rotation matrix is the mulitpliction of the following elementary rotation matrices; C_z*C_y*C_x.

C_z(-90*pi/180) = [[6.123234e-17 1.000000e+00 0.000000e+00]

[-1.000000e+00 6.123234e-17 0.000000e+00]

[0.000000e+00 0.000000e+00 1.000000e+00]]

C_y(-23) = [[0.92050485 0. -0.39073113]

[0. 1. 0.]

[0.39073113 0. 0.92050485]]

C_x(-10) = [[1. 0. 0.]

[0. 0.98480775 0.17364818]

[0. -0.17364818 0.98480775]]

Therefore, C_{BL} = C_z*C_y*C_x = [[0. 0.985 0.174]

[-0.921 -0.068 0.385]

[0.391 -0.16 0.907]]

Putting things together, T_{BL} = [[0. 0.985 0.174 2.57]

[-0.921 -0.068 0.385 -0.52]

[0.391 -0.16 0.907 1.32]

[0. 0. 0. 1.]]

2. The LiDAR has detected a feature of interest at (3.64, 8.30, 2.45). This point is expressed in the LiDAR's *local* reference frame. Express this point in the body reference frame to 3 decimal places. (3 pts)

The point transformed in the body frame is calculated using the following formula: T_{BL}*pL, where
P_L = [[3.64]

[8.30]

[2.45]]

p_B = [[11.1718]

[-3.49359]

[3.63739]]

3. The transformation matrix from the body frame to the LiDAR frame is denoted as $\mathbf{T}_{LB}$. Given $\mathbf{T}_{LB}$ below and $\mathbf{T}_{BL}$ that you previously found, what do you notice about the relationship between these two matrices? How is this relationship useful in robotics applications? (3 pts)

$$\mathbf{T}_{LB} = \begin{bmatrix} 0 & -0.920 & 0.391 & -0.994 \\ 0.985 & -0.068 & -0.160 & -2.355 \\ 0.174 & 0.385 & 0.907 & -1.443 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_LB is the matrix inverse of T_BL calculated in answer 1. This property helps in simplifying calculations. If this property didn't exist, calculations would be expensive due to the trignometry functions in the calculations.

2) Camera Projections

1. Provide the full transformation matrix $\mathbf{T}_{BC}$. Please round values to 3 decimal points. (7 pts)

$C_{BC} = C_y(90*pi/180)$ = [[6.123234e-17 0.000000e+00 1.000000e+00]

[0.000000e+00 1.000000e+00 0.000000e+00]

[-1.000000e+00 0.000000e+00 6.123234e-17]]

T_{BC} = [[0. 0. 1. 2.82]

[0. 1. 0. 0.11]

[-1. 0. 0. 1.06]

[0. 0. 0. 1.]]

2. Assume that the LiDAR has found another point of interest. After using the transformation from Section 1, the feature is expressed in the body frame as (4.47, -0.206, 0.731). Project this point to the camera's image plane using the pinhole camera model and plumb bob distortion model using the parameters provided in Table 2 and 3. Provide the pixel location of the point, rounded to the nearest full pixel and show all calculations. *Hint*: we suggest not to round intermediate calculations, as any rounding errors will compound (especially with the distortion model) (12 pts)

p_B = [[4.47]

[-0.206]

[0.731]]

$T_{CB} = T_{BC}^{-1}$ = [[0. 0. -1. 1.06]

[0. 1. 0. -0.11]

[1. 0. 0. -2.82]

[0. 0. 0. 1.]]

$p_C = T_{CB} * p_B = T_{BC}^{-1} * p_B$ = [[0.329]

[-0.316]

[1.65]]

Normalized pixel, p_N = [[0.19939394]

[-0.19151515]

[1]]

Accounting for distortion,

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \left(1 + \underbrace{\kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6}_{\text{radial distortion}}\right) \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \underbrace{\begin{bmatrix} 2\tau_1 x_n y_n + \tau_2 \left(r^2 + 2x_n^2\right) \\ 2\tau_2 x_n y_n + \tau_1 \left(r^2 + 2y_n^2\right) \end{bmatrix}}_{\text{tangential distortion}}$$

k_1 = -0.369,k_2 = 0.197, k_3 = 0.00135, T_1 = 0.000568, T_2 = -0.068, r = sqrt(x_n^2 + y_n^2)

p_D = [[0.18335156]

[-0.18105552]

[1]]

Point in the image plane in pixels,

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}$$

f_x = 959.79, f_y = 956.93, c_x = 696.02, c_y = 224.18.

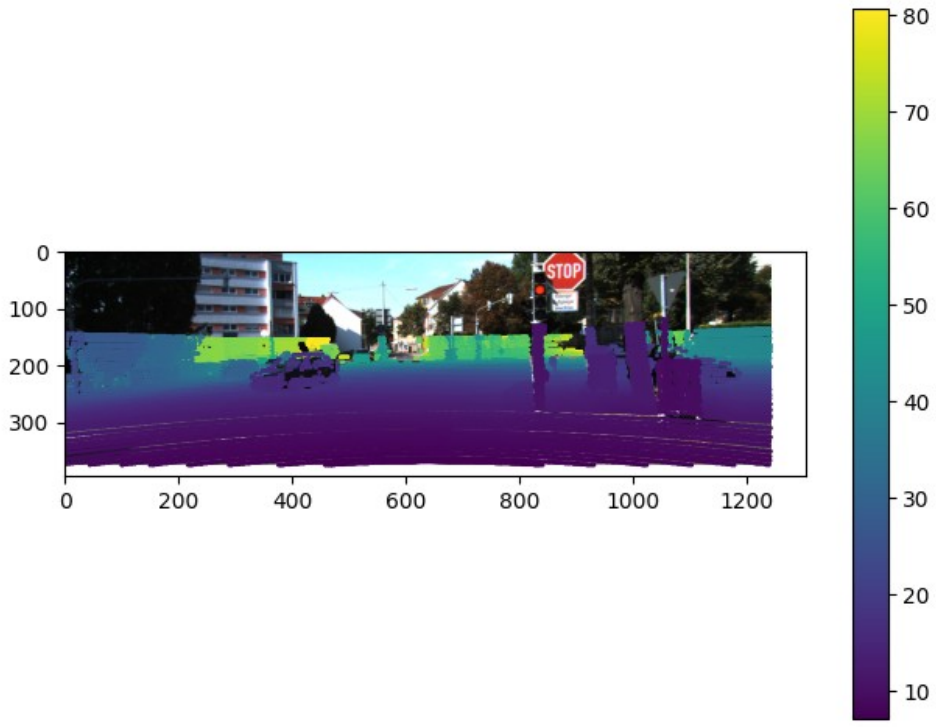
p_s = [[872.]

[51.]]

3. Open the provided image at `data/image/000000.png` (also provided below for reference) in an image editor of your choice (i.e. Inkscape, Gimp, Photoshop, MS Paint) or using Python/OpenCV. What object is at the pixel location above? (1 pt)

The pixel, [872, 51] is located at the lower half of the stop sign.

3) Putting it all Together



Code

```
#!/usr/env/bin python3

import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np
import os

from utils import *
from functions import *

'''
Starter code for loading files, calibration data, and transformations
'''

# File paths
calib_dir = os.path.abspath('./data/calib')
image_dir = os.path.abspath('./data/image')
lidar_dir = os.path.abspath('./data/velodyne')
sample = '000000'

# Load the image
image_path = os.path.join(image_dir, sample + '.png')
image = img.imread(image_path)

# Load the LiDAR points
```

```
lidar_path = os.path.join(lidar_dir, sample + '.bin')
lidar_points = load_velo_points(lidar_path)
print_func(lidar_points, "lidar_points")

# Load the body to camera and body to LiDAR transforms
body_to_lidar_calib_path = os.path.join(calib_dir, 'calib_imu_to_velo.txt')
T_lidar_body = load_calib_rigid(body_to_lidar_calib_path)
# print_func(T_lidar_body, "T_lidar_body")
T_body_lidar = np.linalg.inv(T_lidar_body)

# Load the camera calibration data
# Remember that when using the calibration data, there are 4 cameras with IDs
# 0 to 3. We will only consider images from camera 2.
lidar_to_cam_calib_path = os.path.join(calib_dir, 'calib_velo_to_cam.txt')
cam_to_cam_calib_path = os.path.join(calib_dir, 'calib_cam_to_cam.txt')
cam_calib = load_calib_cam_to_cam(lidar_to_cam_calib_path, cam_to_cam_calib_path)
intrinsics = cam_calib['K_cam2']
T_cam2_lidar = cam_calib['T_cam2_velo']
print_func(intrinsics, "intrinsics")
print_func(T_cam2_lidar, "T_cam2_lidar")

'''
For you to complete:
Part 1: Convert LiDAR points from LiDAR to body frame (for depths)
Note that the LiDAR data is in the format (x, y, z, r) where x, y, and z are
distances in metres and r is a reflectance value for the point which can be
ignored. x is forward, y is left, and z is up. Depth can be calculated using
```

```
# d^2 = x^2 + y^2 + z^2
'''
# 1) LiDAR Feature Coordinate Transformation

# Answer 1.1
C_BL = rotation_matrix_tait_bryan(-10, -23, -90)
T_BL = transformation_matrix(C_BL, 2.57, -0.52, 1.32)
print_func(C_BL, "C_BL")
print_func(T_BL, "T_BL")

# Answer 1.2
pose_lidar_test_1 = np.array([[3.64], [8.30], [2.45]])
pose_body_test_1 = transform_point(T_BL, pose_lidar_test_1)
print_func(pose_body_test_1, "pose_body_test_1")

# Answer 1.3
T_LB = np.around(np.linalg.inv(T_BL), 3)
print_func(T_LB, "T_LB")

# Convert all LiDAR points from LiDAR to body frame
pose_lidar = np.transpose(lidar_points)[0:3, :]
print_func(pose_lidar, "pose_lidar")
pose_body = transform_point(T_body_lidar, pose_lidar)
print_func(pose_body, "pose_body")

# Calculate depth of lidar points in body frame
depth_lidar = calculate_depth(pose_body)
```

```
print_func(depth_lidar, "depth_lidar")

'''
Part 2: Convert LiDAR points from body to camera 2 frame
'''

# Answer 2.1
C_BC = rotation_matrix_y(90)
print_func(C_BC, "C_BC")
T_BC = transformation_matrix(C_BC, 2.82, 0.11, 1.06)
print_func(T_BC, "T_BC")

# Answer 2.2
pose_body_test_2 = np.array([[4.47], [-0.206], [0.731]])
T_CB = np.linalg.inv(T_BC)
print_func(T_CB, "T_CB")
pose_camera_test = transform_point(T_CB, pose_body_test_2)
print_func(pose_camera_test, "pose_camera_test")

pose_normalized_test = normalized_image_plane_projection(pose_camera_test)
print_func(pose_normalized_test, "pose_normalized_test")

pose_distortion_test = lens_distortion(pose_normalized_test, -0.369, 0.197, 0.00135, 0.000568, -0.068)
print_func(pose_distortion_test, "pose_distortion_test")

pixel_test = pixel_coordinates(pose_distortion_test, 959.79, 956.93, 696.02, 224.18)
print_func(pixel_test, "pixel_test")
```

```
'''
Part 3: Project the points from the camera 2 frame to the image plane. You
may assume no lens distortion in the image. Remember to filter out points
where the projection does not lie within the image field, which is 1242x375.
'''

# Convert points from lidar frame to camera frame
pose_camera = transform_point(T_cam2_lidar, pose_lidar)
print_func(pose_camera, "pose_camera")

# Normalize points
pose_normalized = normalized_image_plane_projection(pose_camera)
print_func(pose_normalized, "pose_normalized")

# Factor Plum Distortion model
# pose_distortion = lens_distortion(pose_normalized, -0.369, 0.197, 0.00135, 0.000568, -0.068)
# print_func(pose_distortion, "pose_distortion")

# Convert points from camera frame to image frame
pixel = pixel_coordinates(pose_normalized, intrinsics[0,0], intrinsics[1,1], intrinsics[0,2],
intrinsics[1,2])
print_func(pixel, "pixel")

# Create matrix where a row containing respective depth measurements wrt the body frame is added
pixel_with_depth = np.ones((np.shape(pixel)[0]+1, np.shape(pixel)[1]))
pixel_with_depth[0:2, :] = pixel
```

```
pixel_with_depth[2, :] = depth_lidar
print_func(pixel_with_depth, "pixel_with_depth")

# Delete points that are outside the camera's field of view
pixel_with_depth_reduced = pixel_coordinates_based_on_resolution(pixel_with_depth, 1242, 375)
print_func(pixel_with_depth_reduced, "pixel_with_depth_reduced")

# Part 4: Overlay the points on the image with the appropriate depth values.
# Use a colormap to show the difference between points' depths and remember to
# include a colorbar.
plt.figure()
plt.imshow(image)
plt.scatter(pixel_with_depth_reduced[0,:], pixel_with_depth_reduced[1,:],
c=pixel_with_depth_reduced[2,:], cmap='viridis', s=1)
plt.colorbar()
plt.show()
```

```
'''
Creating functions in this script to use in starter_code.py
'''
import numpy as np
import math
import os

def rotation_matrix_x(theta):
C = np.zeros((3, 3))
theta_rad = theta * math.pi / 180
```

```
C[0,0] = 1
C[1,1] = math.cos(theta_rad)
C[1,2] = -math.sin(theta_rad)
C[2,1] = math.sin(theta_rad)
C[2,2] = math.cos(theta_rad)
return C

def rotation_matrix_y(theta):
C = np.zeros((3, 3))
theta_rad = theta * math.pi / 180
C[0,0] = math.cos(theta_rad)
C[0,2] = math.sin(theta_rad)
C[1,1] = 1
C[2,0] = -math.sin(theta_rad)
C[2,2] = math.cos(theta_rad)
return C

def rotation_matrix_z(theta):
C = np.zeros((3, 3))
theta_rad = theta * math.pi / 180
C[0,0] = math.cos(theta_rad)
C[0,1] = -math.sin(theta_rad)
C[1,0] = math.sin(theta_rad)
C[1,1] = math.cos(theta_rad)
C[2,2] = 1
return C
```

```
def rotation_matrix_tait_bryan(theta_x, theta_y, theta_z):
C_x = rotation_matrix_x(theta_x)
print_func(C_x, "C_x")
C_y = rotation_matrix_y(theta_y)
print_func(C_y, "C_y")
C_z = rotation_matrix_z(theta_z)
print_func(C_z, "C_z")

C_iv_init = np.matmul(C_z, C_y)
C_iv = np.matmul(C_iv_init, C_x)
return np.around(C_iv, 3)

def transformation_matrix(C, x_vi, y_vi, z_vi):
T_iv = np.zeros((4,4))
T_iv[0,0] = C[0,0]
T_iv[0,1] = C[0,1]
T_iv[0,2] = C[0,2]
T_iv[1,0] = C[1,0]
T_iv[1,1] = C[1,1]
T_iv[1,2] = C[1,2]
T_iv[2,0] = C[2,0]
T_iv[2,1] = C[2,1]
T_iv[2,2] = C[2,2]
T_iv[0,3] = x_vi
T_iv[1,3] = y_vi
T_iv[2,3] = z_vi
T_iv[3,3] = 1
```

```
return np.around(T_iv, 3)

def print_func(x, text_x):
print(text_x)
print(x)

def transform_point(T, p):
p_hom = np.ones((4, np.shape(p)[1]))
p_hom[0:np.shape(p)[0], :] = p[0:np.shape(p)[0], :]
# print("p_hom")
# print(p_hom)
p_transformed_hom = np.matmul(T, p_hom)
p_transformed = p_transformed_hom[0:3, :]
return p_transformed

def normalized_image_plane_projection(pose):
pose_norm = np.ones((3, np.shape(pose)[1]))
for i in range(np.shape(pose)[1]):
pose_norm[0, i] = pose[0, i]/pose[2, i]
pose_norm[1, i] = pose[1, i]/pose[2, i]
return pose_norm

def lens_distortion(pose_norm, k_1, k_2, k_3, T_1, T_2):
pose_distort = np.ones((3, np.shape(pose_norm)[1]))
for i in range(np.shape(pose_norm)[1]):
x_n = pose_norm[0, i]
y_n = pose_norm[1, i]
```

```
r = math.sqrt(pow(x_n, 2) + pow(y_n, 2))

pose_distort[0, i] = (1 + k_1*pow(r, 2) + k_2*pow(r, 4) + k_3*pow(r, 6)) * x_n + 2*T_1*x_n*y_n + T_2*(pow(r, 2) + 2*pow(x_n, 2))

pose_distort[1, i] = (1 + k_1*pow(r, 2) + k_2*pow(r, 4) + k_3*pow(r, 6)) * y_n + 2*T_2*x_n*y_n + T_1*(pow(r, 2) + 2*pow(y_n, 2))

return pose_distort


def pixel_coordinates(pose_distort, f_x, f_y, c_x, c_y):

pixel_matrix = np.array([[f_x, 0, c_x], [0, f_y, c_y], [0, 0, 1]])

pixel_coord_hom = np.matmul(pixel_matrix, pose_distort)

pixel_coord = np.delete(pixel_coord_hom, 2, 0)

return np.around(pixel_coord, 0)


def calculate_depth(pose):

depth_matrix = np.zeros((1, np.shape(pose)[1]))

for i in range(np.shape(pose)[1]):

depth_matrix[0, i] = math.sqrt(pow(pose[0, i], 2) + pow(pose[1, i], 2) + pow(pose[2, i], 2))

return depth_matrix


def pixel_coordinates_based_on_resolution(pixel_coord, x_c_lim, y_c_lim):

index_to_del = []

for i in range(np.shape(pixel_coord)[1]):

if (pixel_coord[0, i] > x_c_lim) or (pixel_coord[0, i] < 0) or (pixel_coord[1, i] > y_c_lim) or (pixel_coord[1, i] < 0):

index_to_del.append(i)

pixel_coord_updated = np.delete(pixel_coord, index_to_del, 1)

return pixel_coord_updated
```