TTK4255 Course Work 3:
Geometric image formation

Andreas Haugland

NTNU | Department of Engineering Cybernetics

Part 1 - Choosing a sensor and lens

Task 1.1:

We have

$$
\begin{aligned}
Y'/f &= Y/Z \\
f &= (Y'Z)/Y \\
&= (1e-6*60)/1e-2 \\
&= 5mm
\end{aligned}
$$

Task 1.2:

Image size = 1e-2*1024 = 10.24m. The drone moves 10m per image and the overlap area becomes 10.24m - 10m = 24cm 23.4

Part 2 - Implementing the pinhole camera model

Task 2.1

$$
\begin{aligned}
\tilde{w} &= z, \\
\tilde{u} &= (s_x * f_x) * x + c_x * z \\
\Rightarrow \frac{\tilde{u}}{\tilde{w}} &= c_x + s_x * f_x * \frac{x}{z} \\
&= u
\end{aligned}
$$

The same can be shown for $\tilde{w}$.

Task 2.2

Figure displays the result from task 2.2

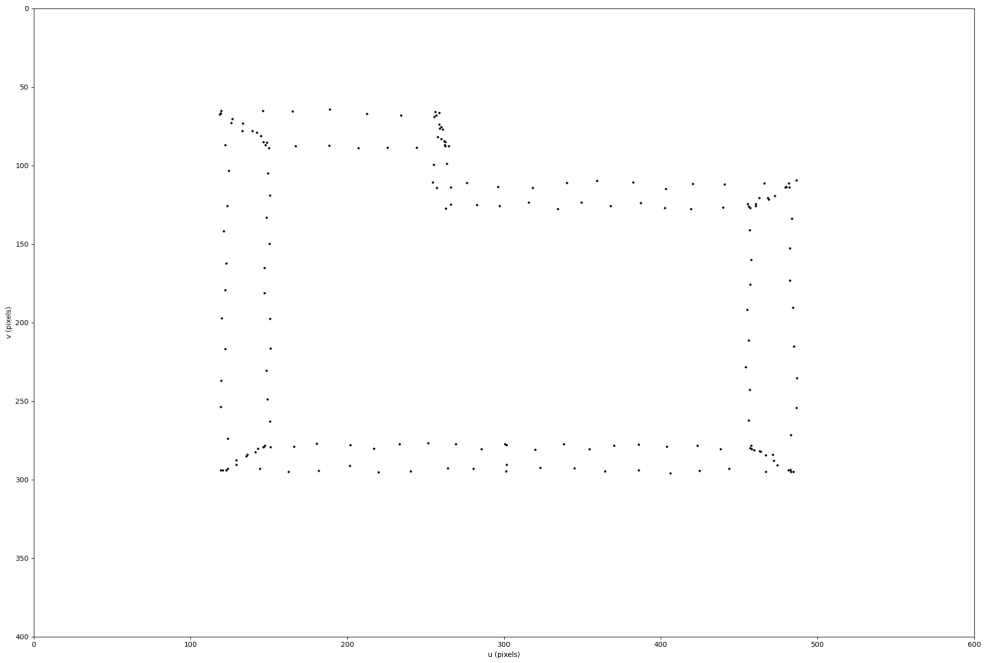

Figure 1: Pinhole model task 2

Part 3 - Homogeneous coordinates and transformations

Task 3.1:

Since u and v values are the same for X and $\frac{X}{k}$, it is not necessary to divide x_c by $\tilde{W}$, as seen in eq. 1

$$u = c_x + s_x * f_x * \frac{x}{z} = c_x + s_x * f_x * \frac{\frac{x}{k}}{\frac{z}{k}} \tag{1}$$

Task 3.2:

The rotations were implemented as seen in figure and the result can be seen in figure .

common.py temp.py task2.py task3.py task4.py

```python
import numpy as np
import matplotlib.pyplot as plt
from math import radians

#
# Tip: Define functions to create the basic 4x4 transform
#
def translate_x(x):
    return np.array([[1, 0, 0, x],
                     [0, 1, 0, 0],
                     [0, 0, 1, 0]
                     [0, 0, 0 ,1]])

def translate_y(y):
    return np.array([[1, 0, 0, 0],
                     [0, 1, 0, y],
                     [0, 0, 1, 0],
                     [0, 0, 0, 1]])

def translate_z(z):
    return np.array([[1, 0, 0, 0],
                     [0, 1, 0, 0],
                     [0, 0, 1, z],
                     [0, 0, 0, 1]])

def rotate_x(theta_deg):
    theta_rad = np.deg2rad(theta_deg)
    c, s = np.cos(theta_rad), np.sin(theta_rad)
    return np.array([[1, 0, 0,  0],
                     [0, c, -s, 0],
                     [0, s, c,  0],
                     [0, 0, 0,  1]])

def rotate_y(phi_deg):
    phi_rad = np.deg2rad(phi_deg)
    c, s = np.cos(phi_rad), np.sin(phi_rad)
    return np.array([[c, 0, s,  0],
                     [0, 1, 0,  0],
                     [-s, 0, c, 0],
                     [0, 0, 0,  1]])

def rotate_z(psi_deg):
    psi_rad = np.deg2rad(psi_deg)
    c, s = np.cos(psi_rad), np.sin(psi_rad)
    return np.array([[c, -s, 0, 0],
                     [s,  c, 0, 0],
                     [0,  0, 1, 0],
                     [0,  0, 0, 1]])
```

common.py temp.py task2.py task3.py task4.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 12 09:27:39 2021

"""

import numpy as np
import matplotlib.pyplot as plt
from common import *
from scipy.spatial.transform import Rotation
from math import radians

# Tip: Use np.loadtxt to load data into an array
K = np.loadtxt('../data/task2K.txt')
X = np.loadtxt('../data/task3points.txt')

T = translate_z(6) @ rotate_x(15) @ rotate_y(45)


# Task 2.2: Implement the project function
X = T @ X

u,v = project(K, X)

# You would change these to be the resolution of your image. Here we have
# no image, so we arbitrarily choose a resolution.
width,height = 600,400

#
# Figure for Task 2.2: Show pinhole projection of 3D points
#
plt.figure(figsize=(4,3))
draw_frame(K, T)
plt.scatter(u, v, c='black', marker='.', s=20)

# The following commands are useful when the figure is meant to simulate
# a camera image. Note: these must be called after all draw commands!!!!

plt.axis('image')     # This option ensures that pixels are square in the figure (preserves aspec
                      # This must be called BEFORE setting xlim and ylim!
plt.xlabel("u (pixels)")
plt.ylabel("v (pixels)")
plt.xlim([0, width])
plt.ylim([height, 0]) # The reversed order flips the figure such that the y-axis points down
plt.show()
```

Figure 2: Code for task 3

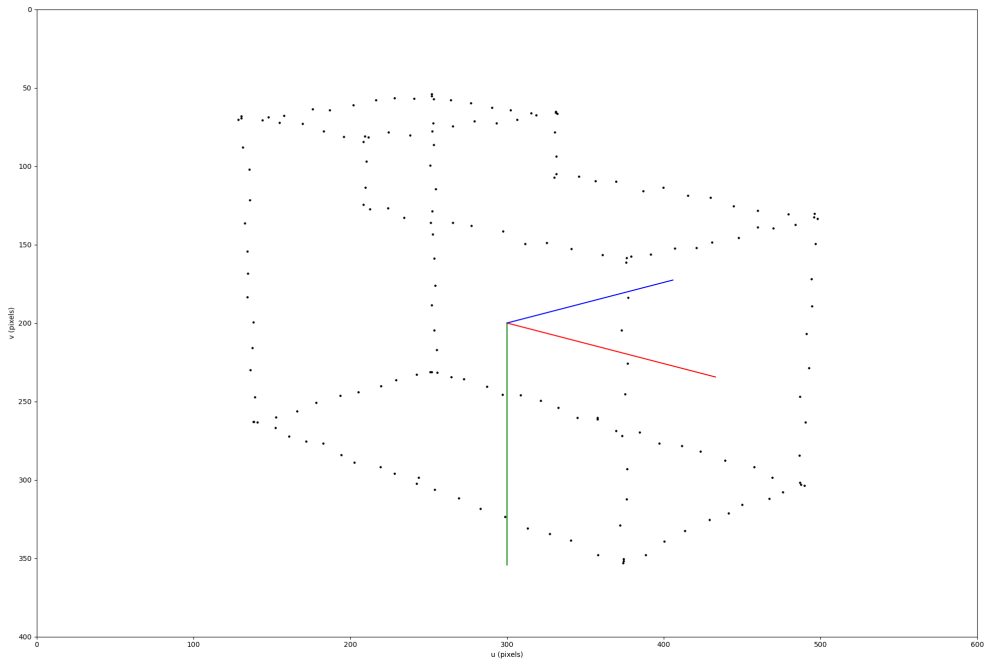

Figure 3: Pinhole model task 2

Part 4 - Image formation model for the Quanser helicopter

Task 4.1

The screw position were defined as

$$\begin{bmatrix} 0 & 0 \\ d & 0 \\ d & d \\ 0 & d \end{bmatrix}^T \tag{2}$$

where d = 11.45

Task 4.2

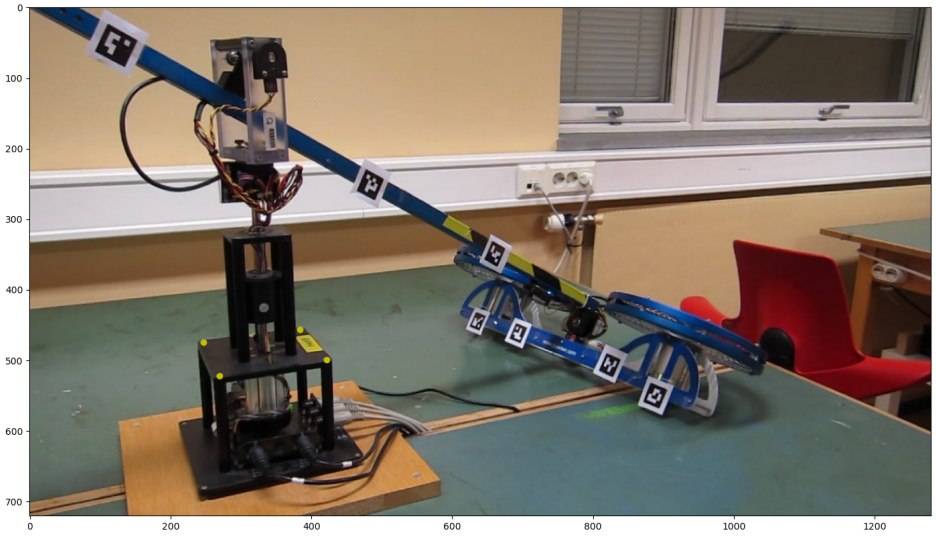

Figure 4: Screw position for task 4.2

Task 4.3

The translation from base frame to platform frame is given as

$$\begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 & 0.05725 \\ -\sin(\psi) & -\cos(\psi) & 0 & 0.05725 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

0.1 Task 4.4

The translation from hiunge frame to base frame is given as

$$\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Task 4.5

The translation from arm frame to hinge frame is given as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Task 4.6

The translation from rotor frame to arm frame is given as

$$\begin{bmatrix} 1 & 0 & 0 & 0.65 \\ 0 & \cos(\phi) & \sin(\phi) & 0 \\ 0 & -\sin(\phi) & cos(\phi) & -0.03 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Task 4.7

The final drawn image can be seen in figure 0.1, allbeit with small dots

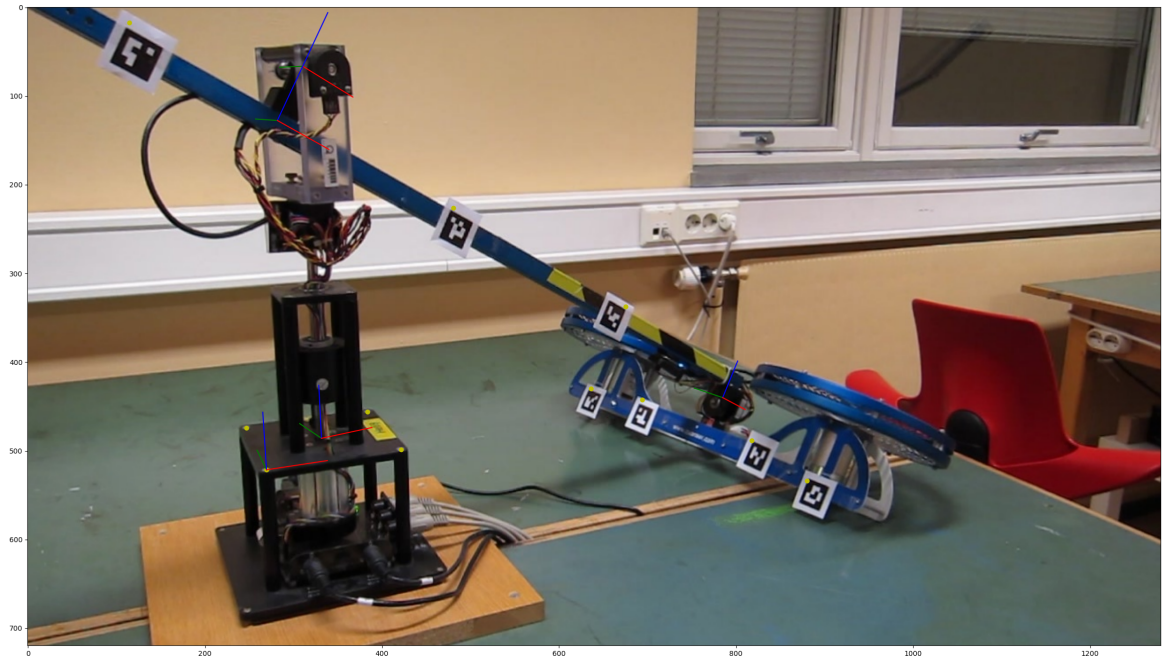

Figure 5: Image model for Quanser Helicopter

References

[1] Richard Szeliski: *Draft (2020) Computer Vision: Algorithms and applications 2nd edition*

[2] Source code for the report can be seen in my private git repo: *https://github.com/aHaugl/TTK4255-Robotsyn*