VAN (086761) - Exercise 1

Kiril Reznik 347527541 | Maor Atar - 307834036

January 28, 2024

Basic Probability

Question 1

Consier a random vector x with a Gaussian distribution :

$$x \sim \mathcal{N}\left(\mu_x, \Sigma_x\right)$$

(a) An explicit expression for p(x)

$$p\left(x\right) = \det\left(2\pi\Sigma\right)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|x-\mu\|_{\Sigma}^{2}\right)}$$

where $\|\cdot\|_{\Sigma}$ is the Mahalanobis norm, defined as: $\|x\|_{\Sigma} = x^T\Sigma^{-1}x$

(b) Consider a linear transformation $y = Ax + b$, assuming A is invertible. Show that y is normally distributed and compute it's mean and covariance.

Gaussian invariant under Linear Transformations

Using that when A is invertible we have the relation:

$$f_Y\left(y\right) = \frac{f_x\left(A^{-1}\left(y-b\right)\right)}{\left|\det A\right|}$$

so,

$$\begin{aligned}
f_Y(y) =& \frac{f_x\left(A^{-1}(y-b)\right)}{|\det A|} \\
=& \frac{exp\left(-\frac{1}{2}\left(A^{-1}(y-b)-\mu_x\right)^T\Sigma_x^{-1}\left(A^{-1}(y-b)-\mu_x\right)\right)}{\sqrt{\det(2\pi\Sigma_x)}\,|\det A|} \\
=& \frac{exp\left(-\frac{1}{2}\left(y-(A\mu_x+b)\right)^T A^{-T}\Sigma_x^{-1}A^{-1}\left(y-(A\mu_x+b)\right)\right)}{\sqrt{\det(2\pi\Sigma_x)\det\left(A^TA\right)}} \\
\overset{1}{=}& \frac{exp\left(-\frac{1}{2}\left(y-(A\mu_x+b)\right)\left(A\Sigma_xA^T\right)^{-1}\left(y-(A\mu_x+b)\right)^T\right)}{\sqrt{\det\left(2\pi A\Sigma_xA^T\right)}}
\end{aligned}$$

1 - Because A is invertibale we have that $A^T\Sigma A$ is invertible too so we can close it with the same parentheses.

We got that Y's density function is of the with the form above (q1a) and therefore is a gaussian with mean $A\mu_x+b$ and covarivace $A^T\Sigma A$.
We can show that these are the right moments with another way:

Mean$T$

Using the linearity of the mean in 1D we get that: (a^i is A's row)

$$\mathbb{E}(y_i)\overset{1}{=}\mathbb{E}\left(\left\langle a^i,x\right\rangle+b^i\right)\overset{2}{=}\left\langle a^i,\mathbb{E}(x)\right\rangle+b^i$$

1 - Definition of y.

2 - Mean's lineartiy in every coordinate of x

Therefore, we get for y:

$$\mathbb{E}(y)=A\mathbb{E}(x)+b$$

Covariance

We'll show that:

$$Cov(y)=A^TCov(x)A$$

for every i,j we have:

$$\begin{aligned}
Cov(y) =& \mathbb{E}\left[(y-\mathbb{E}(y))(y-\mathbb{E}(y))^T\right]=\mathbb{E}\left[yy^T-\mathbb{E}(y)y^T+\mathbb{E}(y)\mathbb{E}(y)^T-y\mathbb{E}(y)^T\right] \\
=& \mathbb{E}\left(yy^T\right)-\mathbb{E}(y)\mathbb{E}(y)^T+\mathbb{E}(y)\mathbb{E}(y)^T-\mathbb{E}(y)\mathbb{E}(y)^T=\mathbb{E}\left(yy^T\right)-\mathbb{E}(y)\mathbb{E}(y)^T
\end{aligned}$$

The seconed equation is just opening the parentheses and then using that mean is linear in 1D and mean of a multivariate Random vector is just the mean on its coordinates and a mean on $\mathbb{E}(y)$ is simply $\mathbb{E}(y)$ therefore for example $\mathbb{E}\left(y\mathbb{E}(y)^T\right) = \mathbb{E}(y)\mathbb{E}\left(y^T\right)$.
Now, using mean's linearity as proved before we get:

$$\begin{aligned}
\mathbb{E}\left((Ax+b)(Ax+b)^T\right) - \mathbb{E}(Ax+b)\,\mathbb{E}(Ax+b)^T &= \\
\mathbb{E}\left(Axx^TA^T + Axb^T + bx^TA^T + bb^T\right) - (A\mathbb{E}(x)+b)(A\mathbb{E}(x)+b)^T &= \\
A\mathbb{E}\left(xx^T\right)A^T + A\mathbb{E}(x)b^T + b\mathbb{E}\left(x^T\right)A^T + bb^T + \\
-A\mathbb{E}(x)\mathbb{E}(x)^TA^T - A\mathbb{E}(x)b^T - b\mathbb{E}(x)^TA^T - bb^T &= \\
A\mathbb{E}\left(xx^T\right)A^T - A\mathbb{E}(x)\mathbb{E}(x)^TA^T &= \\
A\left(\mathbb{E}\left(xx^T\right) - \mathbb{E}(x)\mathbb{E}(x)^T\right)A^T &= \\
A\mathrm{Cov}(x)A^T
\end{aligned}$$

(c)

We don't have here a complete solution but rather some thought.

Option 1 - make H "invertible"

Every matrix has a finite number of eigen values so we take λ to be one that **isn't** $H's$ eigen value. thus we have that $H-\lambda I$ is invertible. Since, if it wasn't invertible, it means there's a vector $x \neq 0$ so that $(H-\lambda I)x = 0 \to Hx = \lambda x$ means that x is an eigen vector and λ is an eigen value, contradiction.
so we have that:

$$y = Hx + \lambda Ix - \lambda Ix + b \to y = (H-\lambda I)x + \lambda Ix + b$$

but from the previous question we know that $(H-\lambda I)x \sim \mathcal{N}\left((H-\lambda I)\mu_x, (H-\lambda I)\Sigma_x(H-\lambda I)^T\right)$ and $\lambda Ix + b \sim \mathcal{N}\left(\lambda\mu_x + b, \lambda^2\Sigma_x\right)$.
So, we just need to show that the sum of two normal distributed random variable is normal distributed with sum over the mean and the covariance: means :

$$\begin{aligned}
\mu_y &= (H-\lambda I)\mu_x + \lambda\mu_x = H\mu_x \\
\Sigma_y &= (H-\lambda I)\Sigma_x(H-\lambda I)^T + \lambda^2\Sigma_x \\
&= H\Sigma_xH^T - \lambda H\Sigma_x - \lambda(H\Sigma_x)^T + \lambda^2\Sigma_x
\end{aligned}$$

Option 2 - Marginalization

$$p(y) = \int_x p(y \mid x)\cdot p(x)$$

And we have from the previous question:

Question 2 |

Let $\mathbf{p}(x) \sim \mathcal{N}(x_0, \Sigma_{x_0})$ be a prior distribution over $x \in \mathbb{R}^n$ with known mean x_0 and covariance Σ_{x_0}. Consider a measurement $z = Hx + v$ where H is known the measurement model and v is a gaussian noise with zero mean and known covriance R.

(a) A posteriori probability function

Option 1:
Using Bayes we have:

$$p(x \mid z) = p(z \mid x) \cdot p(x) \frac{1}{p(z)}$$

and by the total probability theorem we have

$$p(z) = \int_x p(z \mid x) p(x)$$

therefore:

$$(\star) : p(x \mid z) = \frac{p(z \mid x) \cdot p(x)}{\int_{x'} p(z \mid x') p(x')}$$

But we have $p(x)$ and $p(z \mid x)$:

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|x-\mu\|_\Sigma^2\right)}$$

and $z \mid x \sim \mathcal{N}(Hx, R)$:

$$p(z \mid x) = \det(2\pi R)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|z-Hx\|_R^2\right)}$$

So we just need to plug it in equation $(\star)$.

(b) An expresion for the MAP and $\mathbf{p}(x \mid z)$ mean and covariance

The MAP is :

$$\text{argmax}_x p(x \mid z)$$

We notice that

$$\begin{aligned} p(x \mid z) =& p(z \mid x) \cdot p(x) \frac{1}{p(z)} = \eta \cdot p(z \mid x) \cdot p(x) \\ =& \eta \cdot \det(2\pi\Sigma_z)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|z-\mu_z\|_{\Sigma_z}^2\right)} \cdot \det(2\pi\Sigma_x)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|x-\mu_x\|_{\Sigma_x}^2\right)} \\ =& \eta \cdot \det\left(4\pi^2\Sigma_z\Sigma_x\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\left(\|z-\mu_z\|_{\Sigma_z}^2 + \|x-\mu_x\|_{\Sigma_x}^2\right)} \end{aligned}$$

It means the $p(x \mid z)$ is of the form $\eta e^{-\frac{1}{2}f}$ where f is a quadratic equation so it's a Normal distributation. Now we'll find

its Mean and Covariance.
Before we do that we mention that the maximum value of a normal distribution is obtained in the mean. so the **MAP is actually the mean.**
In addition for a normal distribution

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} e^{\left(-\frac{1}{2}\|x-\mu\|_{\Sigma}^{2}\right)}$$

With denoting $f = -\frac{1}{2}\|x-\mu\|_{\Sigma}^{2}$ we have that:

$$f'(\mu) = 0$$

$$f'' = \Sigma^{-1}$$

So, we back to our density function $p(x \mid z)$ we havethe deriving the power $\|z-\mu_z\|_{\Sigma_z}^{2} + \|x-\mu_x\|_{\Sigma_x}^{2}$ once and setting it to zero will give us the mean and twice will gives us the Σ^{-1}.
First we write:

$$\begin{aligned}\|z-\mu_z\|_{\Sigma_z}^{2} + \|x-\mu_x\|_{\Sigma_x}^{2} =& \|z-Hx\|_{R}^{2} + \|x-\mu_x\|_{\Sigma_x}^{2} \\ =& \|R^{-1/2}(Hx-z)\|_{2}^{2} + \|\Sigma_x^{-1/2}(x-\mu_x)\|_{2}^{2}\end{aligned}$$

So, we drive and set to zero:

$$\|R^{-1/2}Hx - R^{-1/2}z\|_{2}^{2} + \|\Sigma_x^{-1/2}x - \Sigma_x^{-1/2}\mu_x\|_{2}^{2} = 0$$

$$\begin{aligned}2\left(R^{-1/2}H\right)^{T}\left(R^{-1/2}Hx - R^{-1/2}z\right) + 2\Sigma_x^{-T/2}\left(\Sigma_x^{-1/2}x - \Sigma_x^{-1/2}\mu_x\right) =& 0 \\ H^{T}R^{-1}(Hx-z) + \Sigma_x^{-1}(x-\mu_x) =& 0 \\ \Sigma_x^{-1}x + H^{T}R^{-1}Hx - H^{T}R^{-1}z - \Sigma_x^{-1}\mu_x =& 0 \\ \Sigma_x^{-1}x + H^{T}R^{-1}Hx =& H^{T}R^{-1}z + \Sigma_x^{-1}\mu_x \\ x =& \left(H^{T}R^{-1}H + \Sigma_x^{-1}\right)^{-1}\left(H^{T}R^{-1}z + \Sigma_x^{-1}\mu_x\right)\end{aligned}$$

so

$$x = \left(H^{T}R^{-1}H + \Sigma_x^{-1}\right)^{-1}\left(H^{T}R^{-1}z + \Sigma_x^{-1}\mu_x\right)$$

And deriving it again we get:

$$\left(\Sigma_x^{-1}x + H^{T}R^{-1}Hx - H^{T}R^{-1}z - \Sigma_x^{-1}\mu_x\right)' = \Sigma_x^{-1} + H^{T}R^{-1}H$$

So the covariance is:

$$\Sigma_x^{-1} + H^{T}R^{-1}H$$

Finally we have:

$$\mathcal{N}\left(\mu=\left(H^TR^{-1}H+\Sigma_x^{-1}\right)^{-1}\left(H^TR^{-1}z+\Sigma_x^{-1}\mu_x\right),\Sigma=\Sigma_x^{-1}+H^TR^{-1}H\right)$$

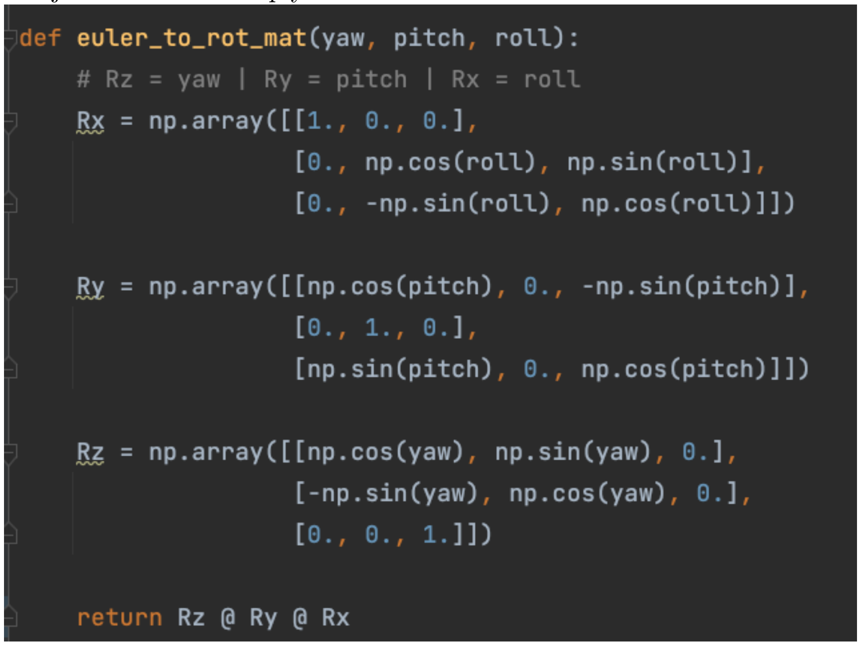
Hands-on Excercises

All code can be found in the APPENDIX part at the end of the HW

Question 1

(a) Rotations

We just need to multiply the 3 matrices:



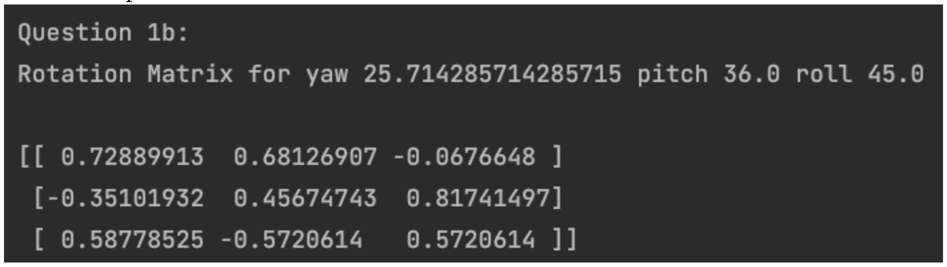
```python
def euler_to_rot_mat(yaw, pitch, roll):
    # Rz = yaw | Ry = pitch | Rx = roll
    Rx = np.array([[1., 0., 0.],
                   [0., np.cos(roll), np.sin(roll)],
                   [0., -np.sin(roll), np.cos(roll)]])

    Ry = np.array([[np.cos(pitch), 0., -np.sin(pitch)],
                   [0., 1., 0.],
                   [np.sin(pitch), 0., np.cos(pitch)]])

    Rz = np.array([[np.cos(yaw), np.sin(yaw), 0.],
                   [-np.sin(yaw), np.cos(yaw), 0.],
                   [0., 0., 1.]])

    return Rz @ Ry @ Rx
```

(b) Rotation matrix

The rotation matrix from Body to global for $\psi = \pi/7, \theta = \pi/5, \phi = \pi/4$ is:

$$R = \begin{bmatrix} 0.7288913 & 0.6812907 & -0.0676648 \\ -0.35101932 & 0.45676743 & 0.81741497 \\ 0.58778525 & -0.5720614 & 0.5720614 \end{bmatrix}$$

Our code print:



```
Question 1b:
Rotation Matrix for yaw 25.714285714285715 pitch 36.0 roll 45.0

[[ 0.72889913  0.68126907 -0.0676648 ]
 [-0.35101932  0.45674743  0.81741497]
 [ 0.58778525 -0.5720614   0.5720614 ]]
```

(c) Rotation Matrix to Euler angles

Code can be seen in function

Theory

Multiplying all the 3 matrices we get: (Note that these matrix performing a rotation clock-wise)

$$R_z(\psi)\,R_y(\theta)\,R_x(\phi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta)\sin(\phi) & -\sin(\theta)\cos(\phi) \\ 0 & \cos(\phi) & \sin(\phi) \\ \sin(\theta) & -\sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\sin(\phi) + \sin(\psi)\cos(\phi) & -\cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ -\sin(\psi)\cos(\theta) & -\sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) + \cos(\psi)\sin(\phi) \\ \sin(\theta) & -\sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

And that equals to the given $R_{3\times 3}$matrix.
From this we can see that:

$$\theta = \sin^{-1}\left(R_1^3\right)$$

So we have 2 options:
We have 2 options. $\theta_1 = \sin^{-1}\left(R_1^3\right)$ and $\theta_2 = -\pi - \theta_1$. We use $-\pi$ because it's clock-wise.
And now we'll seperate into 2 cases,

cos $(\theta) \neq 0$**:**

we can compute the rest with, let's compute ϕ:

$$\begin{matrix} R_2^3 = -\sin(\phi)\cos(\theta) \\ R_3^3 = \cos(\phi)\cos(\theta) \end{matrix} \rightarrow \tan(\phi) = -\frac{R_2^3}{R_3^3} \rightarrow \phi = \arctan\left(\frac{-R_2^3}{R_3^3}\right)$$

But arctan has 2 options. We Notice that because the rotation matrix values are fixed and we have two options for θ we get that:

1. if $-\frac{R_2^3}{\cos\theta} < 0$ that we must have that $\phi \in [-\pi, 0]$
2. else: $\phi \in [0, \pi]$

To avoid devisions we can write the condition $R_2^3 \cdot \cos\theta > 0$
and we notice that $\cos(\theta) = -\cos(-\pi - \theta)$ so when for one option for θ we now what are the angles for θ_2.
so :

1. if $-\frac{R_2^3}{\cos\theta_1} < 0$ that we must have that $\phi_1 \in [-\pi, 0]$ and $\phi_2 \in [0, \pi]$

2. else: $\phi_1 \in [0, \pi]$ and $\phi_2 \in [-\pi, 0]$

We can do the same for ψ:

$$\begin{matrix} R_1^1 = \cos(\psi)\cos(\theta) \\ R_1^2 = -\sin(\psi)\cos(\theta) \end{matrix} \to \tan(\psi) = \frac{-R_1^2}{R_1^1} \to \phi = \arctan\left(\frac{-R_1^2}{R_1^1}\right)$$

1. if $-\frac{R_1^2}{\cos\theta_1} < 0$ that we must have that $\psi_1 \in [-\pi, 0]$ and $\psi_2 \in [0, \pi]$
2. else: $\psi_1 \in [0, \pi]$ and $\psi_2 \in [-\pi, 0]$

and

cos $(\theta) = 0$**:**

It means that $\theta = \frac{\pi}{2}$ OR $\frac{3\pi}{2}$.
If $\theta = \frac{\pi}{2}$:

$$= \begin{bmatrix} 0 & \cos(\psi)\sin(\phi) + \sin(\psi)\cos(\phi) & -\cos(\psi)\cos(\phi) + \sin(\psi)\sin(\phi) \\ 0 & -\sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi) & \sin(\psi)\cos(\phi) + \cos(\psi)\sin(\phi) \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \sin(\psi+\phi) & -\cos(\psi+\phi) \\ 0 & \cos(\psi+\phi) & \sin(\psi+\phi) \\ 1 & 0 & 0 \end{bmatrix}$$

So we have,

$$\psi + \phi = \tan^{-1}\left(\frac{R_2^1}{R_2^2}\right)$$

$$\psi + \phi = \tan^{-1}\left(-\frac{R_3^2}{R_3^1}\right)$$

So:
This linear system has an infinite solutions. So we can choose ψ to be whatever we want (including 0) and we get that $\phi_1 = \tan^{-1}\left(\frac{R_2^1}{R_2^2}\right)$ and $\phi_2 = \pi - \tan^{-1}\left(\frac{R_2^1}{R_2^2}\right)$.
and else:

$$= \begin{bmatrix} 0 & -\cos(\psi)\sin(\phi) + \sin(\psi)\cos(\phi) & \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\phi) \\ 0 & \sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi) & -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\phi) \\ -1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \sin(\psi-\phi) & \cos(\psi-\phi) \\ 0 & \cos(\psi-\phi) & -\sin(\psi-\phi) \\ -1 & 0 & 0 \end{bmatrix}$$

Same as above we get:
$\phi_1 = -\tan^{-1}\left(\frac{R_2^1}{R_2^2}\right)$ and $\phi_2 = \pi + \tan^{-1}\left(\frac{R_2^1}{R_2^2}\right)$

Implementation

Can be found at the implementation python file in the **rot_mat_to_euler_angles(rot_mat)** function.

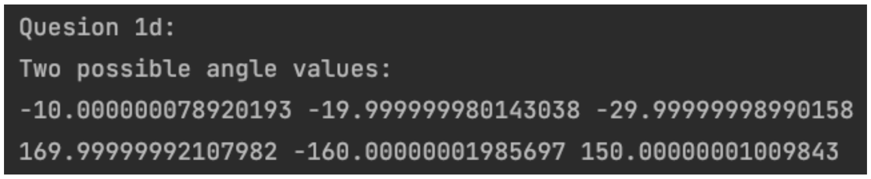
(d) Rotation matrix to euler angles

For the matrix:

$$R = \begin{bmatrix} 0.813797681 & 0.440969611 & 0.378522306 \\ 0.46984631 & 0.882564119 & 0.0180283112 \\ -0.342020143 & 0.163175911 & 0.925416578 \end{bmatrix}$$

Is :

$$\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} -10 \\ -20 \\ -30 \end{pmatrix} or \begin{pmatrix} 170 \\ -160 \\ 150 \end{pmatrix}$$



```
Quesion 1d:
Two possible angle values:
-10.000000078920193 -19.999999980143038 -29.99999998990158
169.9999999210798 2 -160.00000001985697 150.00000001009843
```

Question 2

We want to calculate the transformation from Global to Camera, Means we want: $R_G^C, t_{C\to G}^C$. We are given R_G^C and $t_{C\to G}^G$ and we have that:

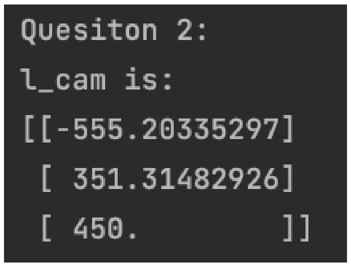
$$t_{C\to G}^c = R_G^C t_{C\to G}^G$$

So, our desired matrix is:

$$\begin{bmatrix} \ell^C \\ 1 \end{bmatrix} = \begin{pmatrix} R_G^C & R_G^C t_{C\to G}^G \\ 0 & 1 \end{pmatrix} \begin{bmatrix} \ell^G \\ 1 \end{bmatrix}$$

Result:

$$\ell^C = \begin{pmatrix} -555.2033 \\ 351.31482 \\ 450 \end{pmatrix}$$



```
Quesiton 2:
l_cam is:
[[-555.20335297]
 [ 351.31482926]
 [ 450.        ]]
```

Question 3

An autonomous ground vehicle (robot) is commanded to move forward by 1 meter each time step. Due to imperfect control system, the robot instead moves forward by 1.01 meter and also rotates by 1 degree.

(a)

In a 2D scenario, the pose of a robot is represented by a 3-element vector: (x, y, ϑ)T , where (x, y) represents the position coordinates, and ϑ represents the orientation angle.
Translation is relative to robot's frame, therefore $t = (0, 1)^T$ - We asume that "forward" means in the Y'th Global CS direction.
Commanded:

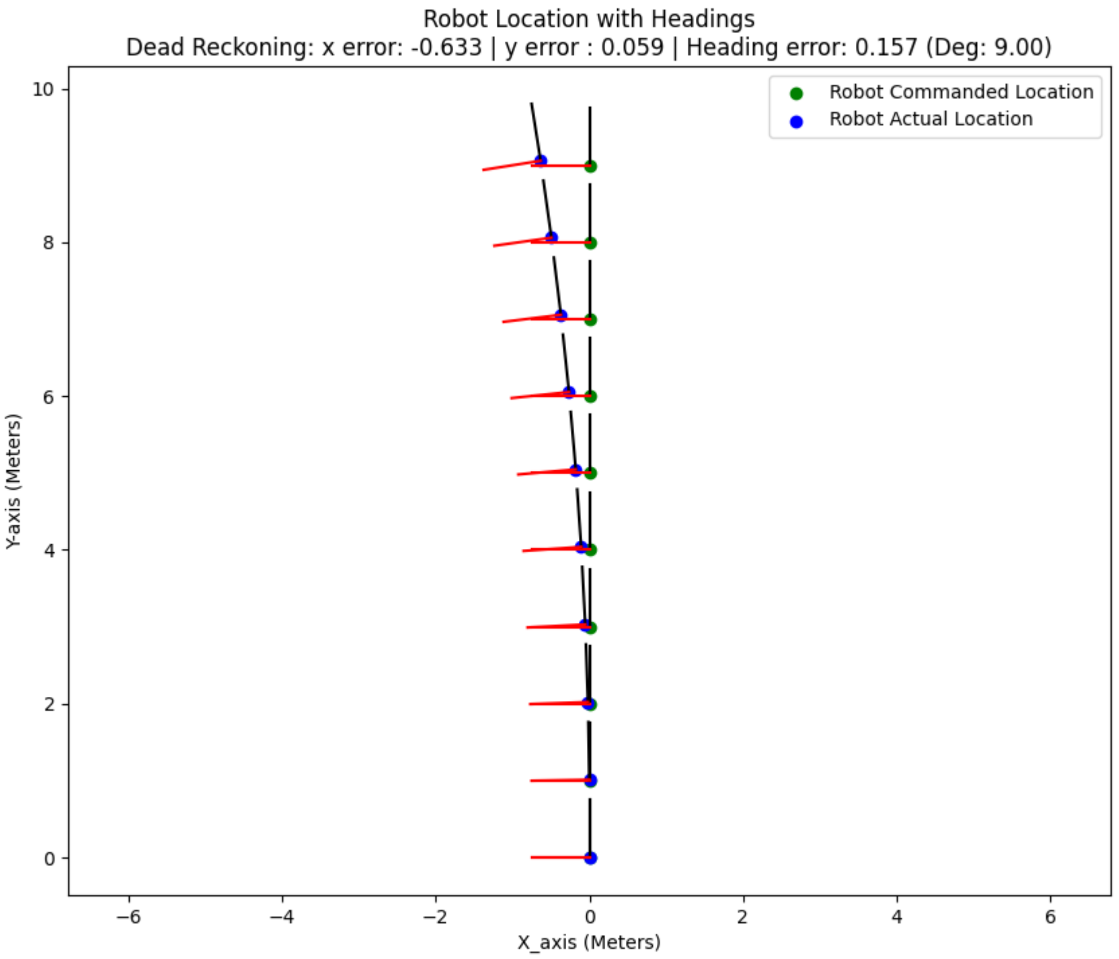
$$T_k^{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Actual (assuming angle's error is counter clock-wise):

$$T_k^{k+1} = \begin{bmatrix} \cos(1) & -\sin(1) & 0 \\ \sin(1) & \cos(1) & 1.01 \\ 0 & 0 & 1 \end{bmatrix}$$

(b)

Evolution of robot pose for 10 steps using pose composition. In terms of x-y position and orientation angle for consecutive times k, k + 1:
Notice that the heading is actually the image of e_2 under the transformation so it'll be R_1(means the first column).
We thought that "Forward" means in the Y direction.



Appendix

Code (Python)

question 1

```
def euler_to_rot_mat(yaw, pitch, roll):
    # Rz = yaw | Ry = pitch | Rx = roll
    Rx = np.array([[1., 0., 0.],
                    [0., np.cos(roll), np.sin(roll)],
                    [0., -np.sin(roll), np.cos(roll)]])

    Ry = np.array([[np.cos(pitch), 0., -np.sin(pitch)],
                    [0., 1., 0.],
                    [np.sin(pitch), 0., np.cos(pitch)]])

    Rz = np.array([[np.cos(yaw), np.sin(yaw), 0.],
                    [-np.sin(yaw), np.cos(yaw), 0.],
                    [0., 0., 1.]])

    return Rz @ Ry @ Rx

def rot_mat_to_euler_angles(rot_mat):
        # psi = yaw | theta = pitch | phi = roll
        yaw1, yaw2, pitch1, pitch2, roll1, roll2 = 0, 0, 0, 0, 0, 0
    if rot_mat[2, 0] != 1 or rot_mat[2, 0] != -1:
        pitch1 = np.arcsin(rot_mat[2, 0])
        pitch2 = -np.pi - pitch1
        atan_roll = np.arctan(-rot_mat[2, 1] / rot_mat[2, 2])
        atan_yaw = np.arctan(-rot_mat[1, 0] / rot_mat[0, 0])
        if rot_mat[2, 1] * np.cos(pitch1) > 0:
                        roll1 = atan_roll if -np.pi < atan_roll < 0 else atan_roll - np.pi
                        roll2 = np.pi + roll1
                else:
                        roll1 = atan_roll if 0 < atan_roll < np.pi else atan_roll - np.pi
                        roll2 = np.pi + roll1
        if rot_mat[1, 0] * np.cos(pitch1) > 0:
                        yaw1 = atan_yaw if -np.pi < atan_yaw < 0 else atan_yaw - np.pi
                        yaw2 = np.pi + yaw1
                else:
                        yaw1 = atan_yaw if 0 < atan_yaw < np.pi else atan_yaw - np.pi
                        yaw2 = np.pi + yaw1
         elif rot_mat[2, 0] == 1:
                        pitch1 = np.pi / 2
                        pitch2 = np.pi / 2
         roll1 = np.arctan(rot_mat[0, 1] / rot_mat[1, 1])
                        roll2 = roll1 - np.pi
         yaw1 = 0         yaw2 = 0
         elif rot_mat[2, 0] == -1:
                        pitch1 = 3 * np.pi / 2
                        pitch2 = 3 * np.pi / 2
         roll1 = -np.arctan(rot_mat[0, 1] / rot_mat[1, 1])
                    roll2 = roll1 - np.pi
```

```
            yaw1 = 0          yaw2 = 0
    return convert_to_range(yaw1), convert_to_range(yaw2), \
                                convert_to_range(pitch1), convert_to_range(pitch2), \
                                convert_to_range(roll1), convert_to_range(roll2)

def convert_to_range(angle):
        if angle < - np.pi:
                return 2 * np.pi + angle
        elif angle > np.pi:
                return angle - 2 * np.pi
        else:
                return angle
```

question 2

```
def q2():
    R_global2cam = np.array([[0.5363, -0.8440, 0.],
                             [0.8440, 0.5363, 0],
                             [0, 0, 1]])
    t_cam2global_globalCS = np.array([-451.2459, 257.0322, 400]).reshape(3, 1)
    t_cam2global_camCS = R_global2cam @ t_cam2global_globalCS

    l_global = np.array([450, 400, 50]).reshape(3, 1)

    l_cam = R_global2cam @ l_global + t_cam2global_camCS
    # l_cam = R_global2cam @ l_global - R_global2cam @ t_cam2global
    # l_cam = R_global2cam @ (l_global - t_cam2global)

    print("Quesiton 2:")
    print("l_cam is: ")
    print(l_cam)

    print()
    print("Check with inverse on l_cam:")
    print("Test inv: ", np.allclose(np.linalg.inv(R_global2cam)
@ l_cam + - t_cam2global_globalCS, l_global, atol=TOLERANCE))

    return
```

question 3b

```
def create_composed_T_arr(T, arr_len=10):
    arr = [None] * arr_len
    arr[0] = np.eye(3)
    for i in range(1, arr_len):
        arr[i] = T @ arr[i - 1]

    return arr



def get_locations_from_T(Ts_lst):
    return np.array([T[0:2, 2] for T in Ts_lst])



def get_headings(Ts_lst):
    return np.array([T[0:2, 1] for T in Ts_lst])



def draw_locs_and_headings(robot_locations, robot_headings,
xlim_bot, xlim_top, ylim_bot, ylim_top, plt_heading=True,
                           plt_loc=True, num=None, ):
    fig, ax = plt.subplots(figsize=(10, 8))
    plt.xlim(xlim_bot, xlim_top)
    plt.ylim(ylim_bot, ylim_top)
    num_robots = num if num is not None else len(robot_locations)
    robot_locations = robot_locations[:num_robots, :]
    robot_headings = robot_headings[:num_robots, :]
    # Plot robot locations
    if plt_loc:
        ax.scatter(robot_locations[:, 0], robot_locations[:, 1],
                label='Robot Location', color='blue')

    arrow_len = 0.5
    # Plot heading vectors
    if plt_heading:
        for i, (x, y) in enumerate(robot_headings):
            dx, dy = robot_headings[i] * arrow_len
            plt.plot(x, y, x + dx, y + dy, marker='o', c='red')
            plt.plot(x, y, x - dy, y - dx)



    # Set labels and legend
    ax.set_xlabel('X-axis')
    ax.set_ylabel('Y-axis')
    ax.set_title('Robot Location with Headings')
    ax.legend()

    # Show the plot
    fig.savefig("./robots_loc")


def draw_robot_pose(ax, robot_locations, robot_headings, type, clr):
    ax.scatter(robot_locations[:, 0], robot_locations[:, 1],
        label=f'Robot {type} Location', color=clr)

    arrow_len = 0.75
    # Plot heading vectors
    for i, (x, y) in enumerate(robot_locations):
        dx, dy = robot_headings[i] * arrow_len
        print(dx, dy)
        ax.plot([x, x + dx], [y, y + dy], color='black')

        # Calculate the perpendicular vector (-dy, dx)
        perpendicular_dx = -dy
        perpendicular_dy = dx

        # Plot the perpendicular line
        ax.plot([x, x + perpendicular_dx], [y, y + perpendicular_dy], color='red')

        # Set equal scaling for better visualization
        ax.set_aspect('equal', adjustable='datalim')


def draw_robot_pose_actual_and_commanded(commanded_loc, commanded_headings,
                actual_loc, actual_headings, title):
    fig, ax = plt.subplots(figsize=(10, 8))
    ax.set_xlim(-4, 4)

    draw_robot_pose(ax, commanded_loc, commanded_headings, "Commanded", 'green')
    draw_robot_pose(ax, actual_loc, actual_headings, "Actual", 'blue')
    # Set labels and legend
    ax.set_xlabel('X_axis (Meters)')
    ax.set_ylabel('Y-axis (Meters)')
    ax.set_title(f'Robot Location with Headings\n{title}')

    ax.legend()

    # Show the plot
    fig.savefig("./robots_loc")


def q3b():
    commanded_T = np.array([[1, 0, 0.],
                            [0, 1, 1],
                            [0, 0, 1]])
    actual_angle = 1 * DEG_TO_RAD
    actual_T = np.array([[np.cos(actual_angle), -np.sin(actual_angle), 0.],
                         [np.sin(actual_angle), np.cos(actual_angle), 1.0],
                         [0, 0, 1]])

    commanded_T_lst = create_composed_T_arr(commanded_T)
    actual_T_lst = create_composed_T_arr(actual_T)

    commanded_loc = get_locations_from_T(commanded_T_lst)
    actual_loc = get_locations_from_T(actual_T_lst)

    commanded_headings = get_headings(commanded_T_lst)
    actual_headings = get_headings(actual_T_lst)

    loc_diff = actual_loc[-1] - commanded_loc[-1]
    heading_err = np.arccos(np.dot(actual_headings[-1, :], commanded_headings[-1, :]))
    x_err = loc_diff[0]
    y_err = loc_diff[1]

    err_title = "Dead Reckoning: x error: {:.3f} | y error : {:.3f}
                         | Heading error: {:.3f} (Deg: {:.2f})".format(x_err,



    draw_robot_pose_actual_and_commanded(commanded_loc,
commanded_headings, actual_loc, actual_headings, err_title)
```