In [149]:

```
from sympy import*
from IPython.display import Image, display, HTML
from scipy import optimize
import matplotlib.pyplot as plt
import numpy as np
```

Homework 2

1.1 Composition of Transformations

In [150]:

```
x, y, z, theta, phi, psi, x_p, y_p, z_p, nu, dx, dy, dz=symbols('x y z theta phi psi x^{\pr
```

In [151]:

```
HR_X_G_phi=Matrix([[1,0,0,0],
                   [0,cos(phi),-sin(phi),0],
                   [0,sin(phi),cos(phi),0],
                   [0,0,0,1]])
HT_Y_L_y=Matrix([[1,0,0,0],
                   [0,1,0,y],
                   [0,0,1,0],
                   [0,0,0,1]])
HR_Z_G_theta=Matrix([[cos(theta),-sin(theta),0,0],
                   [sin(theta),cos(theta),0,0],
                   [0,0,1,0],
                   [0,0,0,1]])
HR_X_G_psi=Matrix([[1,0,0,0],
                   [0,cos(psi),-sin(psi),0],
                   [0,sin(psi),cos(psi),0],
                   [0,0,0,1]])
```

The multiplication order is the following based on the frame that we are working with. In the first group are the rotation around the world frame in order STEP_3*STEP_1 according to the rules of extrinsic rotation in the second group are the rotations/translations around/along the current frame in the order of STEP_2*STEP_4 according to the rules of intrinsic transformations.

$$R_{\theta} \cdot R_{\phi} \cdot T_{y} \cdot R_{\psi}$$

In [152]:

```
simplify(HR_Z_G_theta*HR_X_G_phi*HT_Y_L_y*HR_X_G_psi) #.subs({y:5,phi:pi/2,theta:pi/2,psi:p
```

Out[152]:

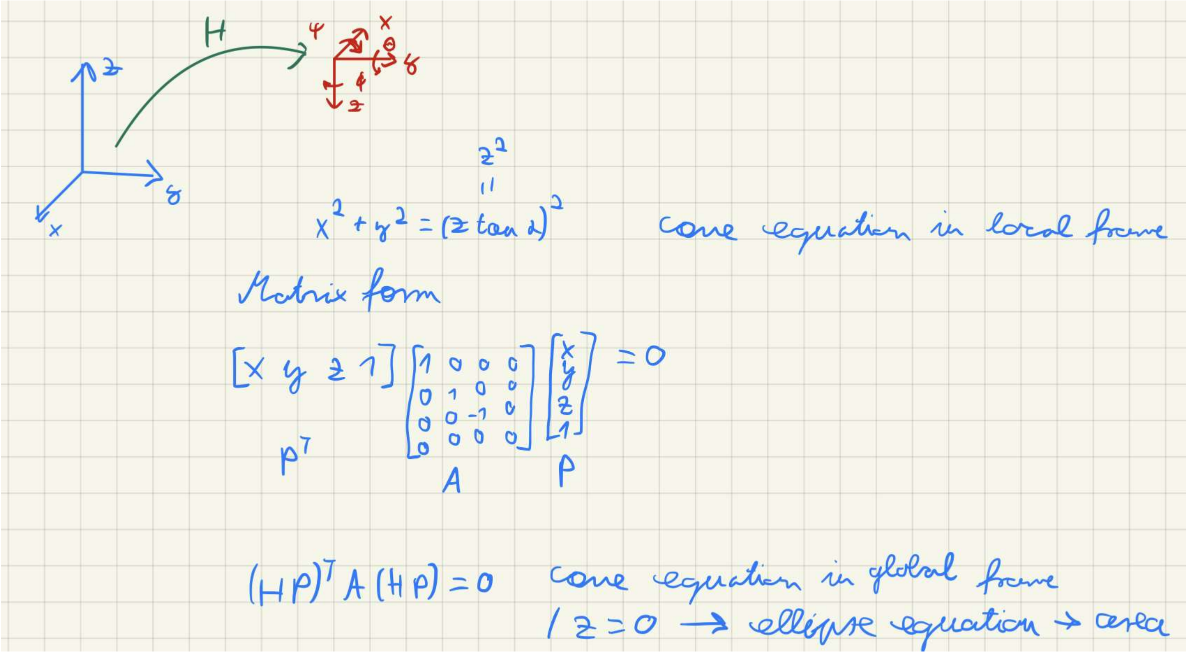
$$\begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\phi+\psi) & \sin(\theta)\sin(\phi+\psi) & -y\sin(\theta)\cos(\phi) \\ \sin(\theta) & \cos(\theta)\cos(\phi+\psi) & -\sin(\phi+\psi)\cos(\theta) & y\cos(\phi)\cos(\theta) \\ 0 & \sin(\phi+\psi) & \cos(\phi+\psi) & y\sin(\phi) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2 Modeling beyond rigid transformations

In [153]:

```
Image("question_1_2.png")
```

Out[153]:



In [154]:

```
x, y, z, dx, dy, dz=symbols('x y z dx dy dz')
```

In [155]:

```
#z and y pointing the same direction so x is -x, therefore -dx
#H_x ratation around X
```

First I calculated the homogeneous transformation matrix that transforming the coordinate frame of the drone's camera to the world frame. Since the camera pointing downwards, I implemented a rotation around X-Axis, therefore the X Axis of the two coordinate frame's are pointing towards each other while the Y and Z are coincident, as a result the translation along X carries a negative sign.

In [156]:

```
HR_X_G_psi=Matrix([[1,0,0,0],
                   [0,cos(psi),-sin(psi),0],
                   [0,sin(psi),cos(psi),0],
                  [0,0,0,1]])
HR_Y_G_theta=Matrix([[cos(theta),0,sin(theta),0],
                   [0,1,0,0],
                   [-sin(theta),0,cos(theta),0],
                    [0,0,0,1]])
HR_Z_G_phi=Matrix([[cos(phi),-sin(phi),0,0],
                   [sin(phi),cos(phi),0,0],
                   [0,0,1,0],
                  [0,0,0,1]])
T=Matrix([[1,0,0,-dx],
         [0,1,0,dy],
         [0,0,1,dz],
         [0,0,0,1]])
H_X=Matrix([[1,0,0,0],
            [0,-1,0,0],
            [0,0,-1,0],
            [0,0,0,1]])
H=simplify(T*H_X*HR_Z_G_phi*HR_Y_G_theta*HR_X_G_psi)
H
```

Out[156]:

$$\begin{bmatrix} \cos(\phi)\cos(\theta) & -\sin(\phi)\cos(\psi)+\sin(\psi)\sin(\theta)\cos(\phi) & \sin(\phi)\sin(\psi)+\sin(\theta)\,\mathrm{c} \\ -\sin(\phi)\cos(\theta) & -\sin(\phi)\sin(\psi)\sin(\theta)-\cos(\phi)\cos(\psi) & -\sin(\phi)\sin(\theta)\cos(\psi)+ \\ \sin(\theta) & -\sin(\psi)\cos(\theta) & -\cos(\psi)\cos \\ 0 & 0 & 0 \end{bmatrix}$$

The matrix representation of the cone equation in the local frame is created below.

In [157]:

```
#cone equation
P=Matrix([x,y,z,1])
A=Matrix([[1,0,0,0],
        [0,1,0,0],
        [0,0,-1,0],
        [0,0,0,0]])
P.T*A*P
```

Out[157]:

$$\left[x^2 + y^2 - z^2 \right]$$

The cone equation in the global frame is created below using the transformation matrix.

In [158]:

```
exp=(simplify(((H*P).T*A*(H*P)).subs({z:0})))[0]
p=Poly(exp.evalf(),x,y,x*y,x**2,y**2)
p
```

Out[158]:

$$\mathrm{Poly}\Big(\left(\sin^2(\phi)\cos^2(\theta) - \sin^2(\theta) + \cos^2(\phi)\cos^2(\theta)\right)x^2$$
$$+ \left(2\sin^2(\phi)\sin(\psi)\sin(\theta)\cos(\theta) + 2\sin(\psi)\sin(\theta)\cos^2(\phi)\cos(\theta) + 2\sin(\psi)\sin(\right.$$
$$+ \left(-2dx\cos(\phi)\cos(\theta) - 2dy\sin(\phi)\cos(\theta) - 2dz\sin(\theta)\right)x$$
$$+ \left(\sin^2(\phi)\sin^2(\psi)\sin^2(\theta) + \sin^2(\phi)\cos^2(\psi) + \sin^2(\psi)\sin^2(\theta)\cos^2(\phi) - \sin^2(\psi\right.$$
$$+ \left(2dx\sin(\phi)\cos(\psi) - 2dx\sin(\psi)\sin(\theta)\cos(\phi) - 2dy\sin(\phi)\sin(\psi)\sin(\theta) - 2d\right.$$
$$x, y, xy, x^2, y^2, domain = \mathrm{EX}\Big)$$

Using the polinomial form of the equation in a plane - meaning z has been set to 0- we can find the coefficients required for the calculation of the area.

In [159]:

```
Param=Matrix([p.coeffs()[0],p.coeffs()[1],p.coeffs()[3],p.coeffs()[2],p.coeffs()[4],p.coeff
Param
```

Out[159]:

$$\left[\begin{matrix}\sin^2(\phi)\cos^2(\theta) - \sin^2(\theta) + \cos^2(\phi)\cos^2(\theta)\\ 2\sin^2(\phi)\sin(\psi)\sin(\theta)\cos(\theta) + 2\sin(\psi)\sin(\theta)\cos^2(\phi)\cos(\theta) + 2\sin(\\ \sin^2(\phi)\sin^2(\psi)\sin^2(\theta) + \sin^2(\phi)\cos^2(\psi) + \sin^2(\psi)\sin^2(\theta)\cos^2(\phi) - \sin^2(\psi)c\\ -2dx\cos(\phi)\cos(\theta) - 2dy\sin(\phi)\cos(\theta) - 2dz\sin(\theta)\\ 2dx\sin(\phi)\cos(\psi) - 2dx\sin(\psi)\sin(\theta)\cos(\phi) - 2dy\sin(\phi)\sin(\psi)\sin(\theta) - 2dy\cos(\\ dx^2 + dy^2 - dz^2\end{matrix}\right.$$

The covered area can be ploted below.

In [160]:

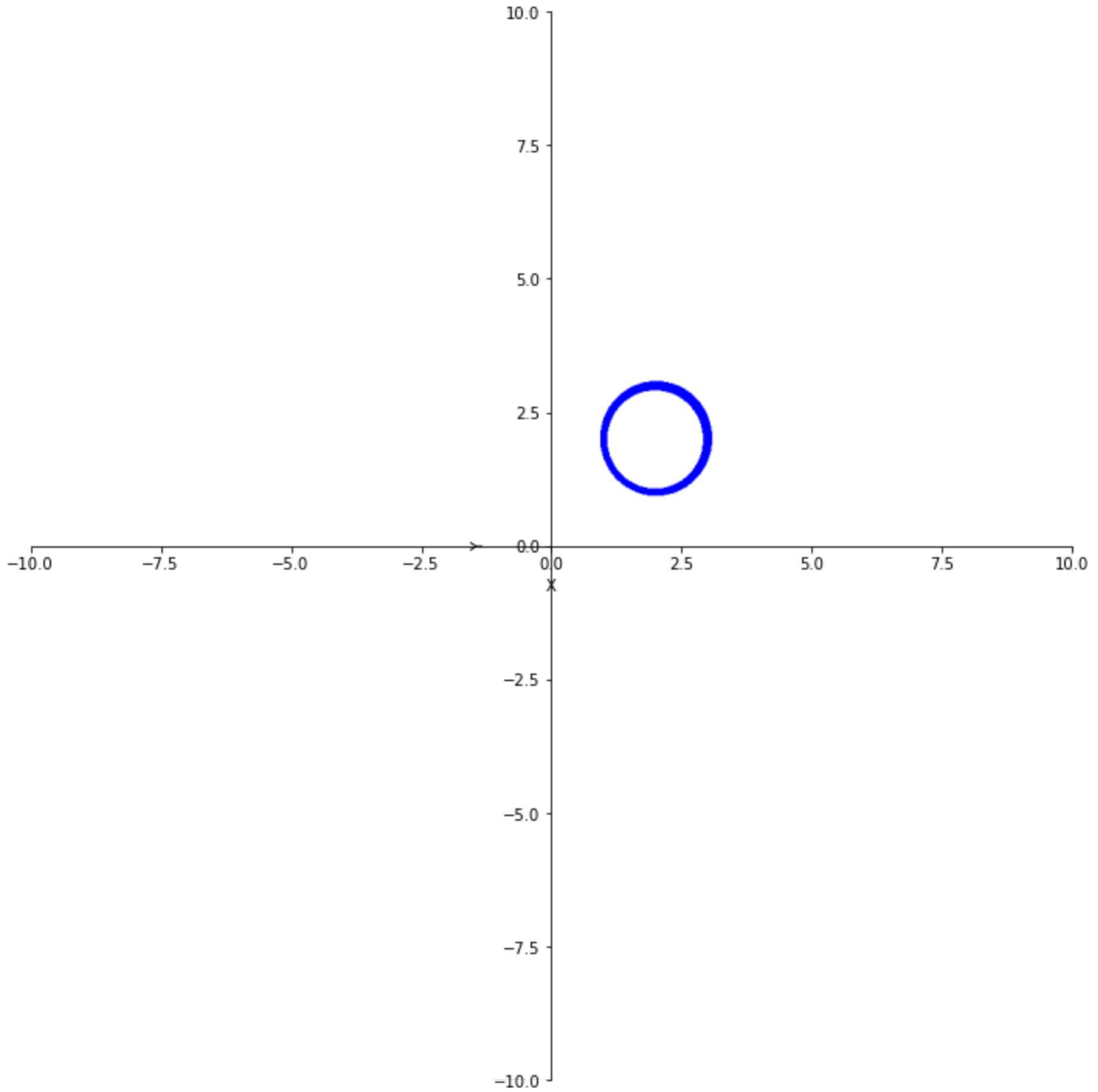
```
plot_p=(Param[0]*(x**2)+Param[1]*(x*y)+Param[2]*(y**2)+Param[3]*x+Param[4]*y+Param[5]).subs
plot_p
```

Out[160]:

$$x^2 - 4x + y^2 - 4y + 7$$

In [161]:

```
plot_implicit(plot_p, x_var=(x,-10,10),y_var=(y,-10,10), xlabel="X", ylabel="Y")
```



Out[161]:

```
<sympy.plotting.plot.Plot at 0x18d6a650a00>
```

In [162]:

```
a=Param[0]
b=Param[1]/2
c=Param[2]
d=Param[3]/2
e=Param[4]/2
f=Param[5]
K=Matrix([[a,b,d],[b,c,e],[d,e,f]])
```

The area can be calculated using the given equation

In [163]:

```
A=(-pi)/(sqrt(((a*c)-b**2)**3))*det(K)
simplify(A.subs({psi:0,theta:0,phi:0,dx:2,dy:2,dz:1}))
```

Out[163]:

π

The result is accurate since the radius of the circle is equal to 1.

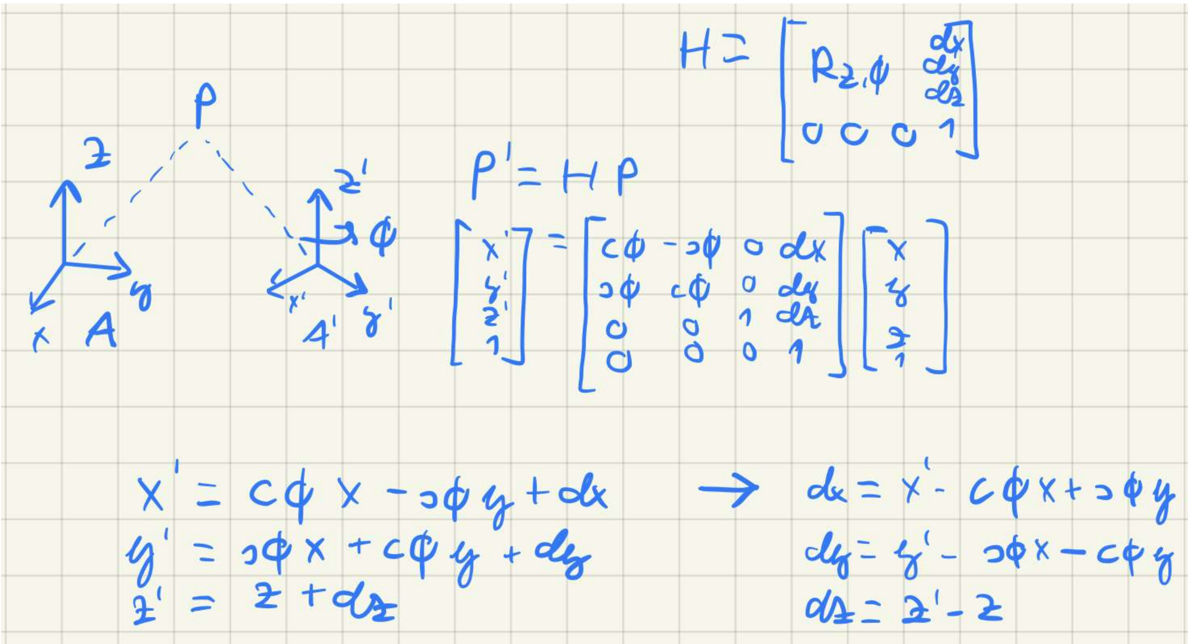
1.3 Transform Estimation

The position of P' is written up as the transformation of P using H.

In [164]:

```
Image("question_1_3.png")
```

Out[164]:



Rotation around Z-axis with ϕ and translation along X, Y, Z with dx, dy and dz respectively

In [165]:

```
HR_Z_phi=Matrix([[cos(phi),-sin(phi),0,dx],
                   [sin(phi),cos(phi),0,dy],
                   [0,0,1,dz],
                   [0,0,0,1]])
HR_Z_phi
```

Out[165]:

$$\begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & dx \\ \sin(\phi) & \cos(\phi) & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where dx, dy and dz are equal to the followings:

In [166]:

```
dx=x_p-x*cos(phi)+y*sin(phi)
dx
```

Out[166]:

$$-x\cos(\phi) + x' + y\sin(\phi)$$

In [167]:

```
dy=y_p-x*sin(phi)-y*cos(phi)
dy
```

Out[167]:

$$-x\sin(\phi) - y\cos(\phi) + y'$$

In [168]:

```
dz=z_p-z
dz
```

Out[168]:

$$-z + z'$$

2.1 Trajectory Optimisation

First using the rotations given around X, Y, Z, based on that the symbolic rotation matrix can be calculated.

In [169]:

```
HR_X_G_psi=Matrix([[1,0,0],
                   [0,cos(psi),-sin(psi)],
                   [0,sin(psi),cos(psi)]])
HR_Y_G_theta=Matrix([[cos(theta),0,sin(theta)],
                   [0,1,0],
                   [-sin(theta),0,cos(theta)]])
HR_Z_G_phi=Matrix([[cos(phi),-sin(phi),0],
                   [sin(phi),cos(phi),0],
                   [0,0,1]])
simplify(HR_Z_G_phi*HR_Y_G_theta*HR_X_G_psi).evalf()
```

Out[169]:

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

We can plug in the given values.

In [170]:

```
R=(((simplify(HR_Z_G_phi*HR_Y_G_theta*HR_X_G_psi))).subs({psi:(35/180*pi),theta:(15/180*pi)
R
```

Out[170]:

$$\begin{bmatrix} 0.907673371190369 & -0.140666775591018 & 0.395400947743853 \\ 0.330366089549352 & 0.820524878545215 & -0.466473118801319 \\ -0.258819045102521 & 0.554032293222323 & 0.791240115236224 \end{bmatrix}$$

We can check that our calculation theta is accurate.

In [171]:

```
(asin(-R[2,0])/pi*180).evalf()
```

Out[171]:

$$15.0$$

To achieve the smoothest rotation the drone rotates around one specific axis with a specific angle, that are results of its rotation around the local axises.From the transformation matrix R the Axis-Angle representation (using Rodriguez form) can be calculated. The required angle is ν:

In [172]:

```
nu=simplify(acos((trace(R)-1)/2)) #angle
(nu/pi*180).evalf() #deg
```

Out[172]:

$$40.5605521115655$$

Using R and ν the cross product tensor can be calculated.

In [173]:

```
nx=simplify((R-R.T)/(2*sin(nu)))
nx
```

Out[173]:

$$\begin{bmatrix} 0 & -0.362192939275823 & 0.503051654519739 \\ 0.362192939275823 & 0 & -0.784700775852614 \\ -0.503051654519739 & 0.784700775852614 & 0 \end{bmatrix}$$

From the cross product tensor we can define the Axis n that we are rotation around. The X value is the [2,1] element, the Y value is the [0,2] element, while the Z value is the [1,0] element

In [174]:

```
n = (Matrix([nx[2,1],nx[0,2],nx[1,0]]))
n #axis
```

Out[174]:

$$\begin{bmatrix} 0.784700775852614 \\ 0.503051654519739 \\ 0.362192939275823 \end{bmatrix}$$

This axis needs to be an eigenvector, that is true, see below

In [175]:

```
n.norm()
```

Out[175]:

$$1.0$$

Since we are rotation around a random axis the rotation around X, Y, Z should end at the same time, therefore the largest distance to travel requires the highest angular velocity, so $\omega_x = 1 deg/s$, I am working with constant velocities.

In [176]:

```
w_x=1/180*pi #rad/s
w_x.evalf()
```

Out[176]:

$$0.0174532925199433$$

The angular rotations in the global frame can be calculated from ν using the ratio defined by the axis n. Using the largest angular rotation and the coressponding maximum angular velocity we can calculate the time. Using t and the calculated angular rotations ω_y, ω_z are the following.

In [198]:

```
psi_r=(n[0]*nu).evalf() #angular around local X
```

In [178]:

```
t=psi_r/w_x.evalf()
t #s
```

Out[178]:

31.8278967109558

In [179]:

```
theta_r=(n[1]*nu).evalf() #angular around local Y
```

In [180]:

```
w_y=theta_r/t
w_y #rad/sec
```

Out[180]:

0.0111888607086372

In [181]:

```
(w_y/pi*180).evalf() #deg/sec
```

Out[181]:

0.641074496164668

In [182]:

```
phi_r=(n[2]*nu).evalf() #angular around local Z
```

In [183]:

```
w_z=phi_r/t
w_z #rad/sec
```

Out[183]:

0.00805588513783542

In [184]:

```
(w_z/pi*180).evalf() #deg/sec
```

Out[184]:

0.461568218640135

To check the calculations we can calculate ω_k the angular velocity around n in two different ways - as a vectorial sum of $\omega_x, \omega_y, \omega_z$ or dividing nu with t and the results should be the same, which is true.

In [185]:

```
sqrt(w_x**2+w_y**2+w_z**2).evalf() #rad/s
```

Out[185]:

$$0.022241971789794$$

In [186]:

```
w_k=nu/t
w_k #rad/s
```

Out[186]:

$$0.022241971789794$$

In [187]:

```
nu/w_k
```

Out[187]:

$$31.8278967109558$$

In [188]:

```
(w_k/pi*180).evalf() #deg/sec
```

Out[188]:

$$1.27437111160423$$

The general form of the rotation matrix can be written up using Axis-Angle representation

In [189]:

```
gamma, u_x, u_y, u_z=symbols('gamma u_x u_y u_z')
u=Matrix([u_x,u_y,u_z])
konst=(1-cos(gamma))
```

In [190]:

```
R_Axis_Angle=Matrix([[cos(gamma)+(u[0]**2)*konst, u[0]*u[1]*konst-u[2]*sin(gamma), u[0]*u[2
                    [u[1]*u[0]*konst+u[2]*sin(gamma), cos(gamma)+(u[1]**2)*konst, u[1]*u[2]
                    [u[2]*u[0]*konst-u[1]*sin(gamma), u[2]*u[1]*konst+u[0]*sin(gamma), cos(
R_Axis_Angle
```

Out[190]:

$$\left[\begin{matrix} u_x^2 \cdot (1-\cos(\gamma)) + \cos(\gamma) & u_x u_y (1-\cos(\gamma)) - u_z \sin(\gamma) & u_x u_z (1-\cos(\gamma)) + u_y \\ u_x u_y (1-\cos(\gamma)) + u_z \sin(\gamma) & u_y^2 \cdot (1-\cos(\gamma)) + \cos(\gamma) & -u_x \sin(\gamma) + u_y u_z (1 - \\ u_x u_z (1-\cos(\gamma)) - u_y \sin(\gamma) & u_x \sin(\gamma) + u_y u_z (1-\cos(\gamma)) & u_z^2 \cdot (1-\cos(\gamma)) + c \end{matrix}\right.$$

In [191]:

```
B=R_Axis_Angle.subs({gamma:w_k*t,u_x:n[0],u_y:n[1],u_z:n[2]})
```

We can check the angular rotation values using R and R_Axis_Angle.

We can check the angular rotation values using R and R_Axis_Angle.

In [192]:

```
simplify(HR_Z_G_phi*HR_Y_G_theta*HR_X_G_psi).evalf() #R
```

Out[192]:

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

In [193]:

```
theta_2=asin(-B[2,0])
(theta_2/pi*180).evalf()
```

Out[193]:

15.0

In [194]:

```
phi_2=acos(B[0,0]/cos(theta_2))/pi*180
phi_2.evalf()
```

Out[194]:

20.0

In [195]:

```
psi_2=acos(B[2,2]/cos(theta_2))/pi*180
psi_2.evalf()
```

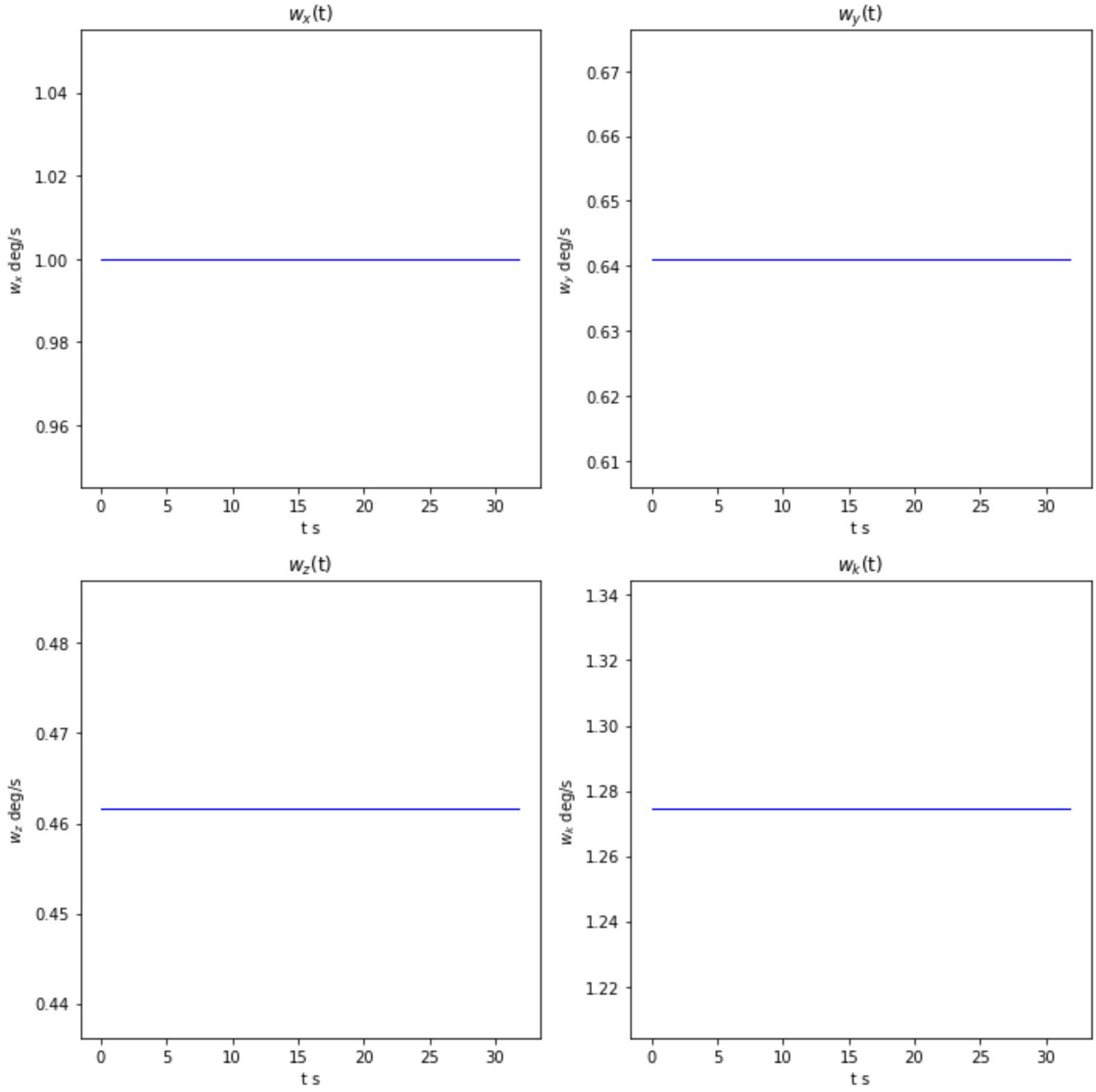
Out[195]:

35.0

```
The angular velocities around the local axises and the angular rotations in the global
frame can be plotted over time.
```

In [196]:

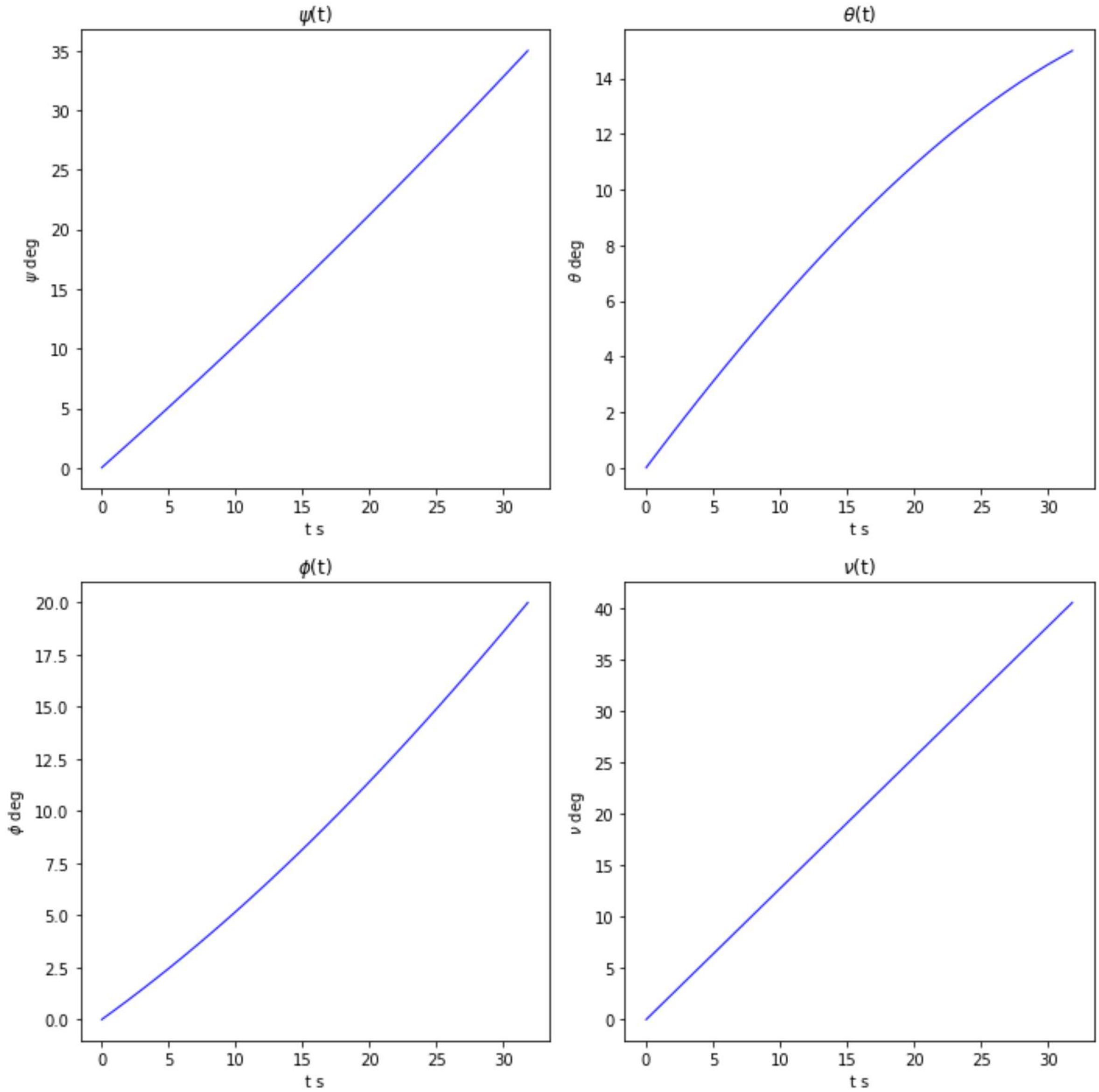
```
plt.rcParams['figure.figsize'] = [10, 10]
sampling=[]
a_velocity_x=[]
a_velocity_y=[]
a_velocity_z=[]
a_velocity_k=[]
for i in range(0,int(t)):
    sampling.append(i)
    a_velocity_x.append(((w_x/pi*180).evalf()))
    a_velocity_y.append(((w_y/pi*180).evalf()))
    a_velocity_z.append(((w_z/pi*180).evalf()))
    a_velocity_k.append(((w_k/pi*180).evalf()))

sampling.append(t)
a_velocity_x.append(((w_x/pi*180).evalf()))
a_velocity_y.append(((w_y/pi*180).evalf()))
a_velocity_z.append(((w_z/pi*180).evalf()))
a_velocity_k.append(((w_k/pi*180).evalf()))
plt.subplot(2, 2, 1)
plt.plot(sampling, a_velocity_x, color='blue', linestyle='solid', linewidth = 1)
plt.title("$w_x$(t)")
plt.xlabel("t s")
plt.ylabel("$w_x$ deg/s")

plt.subplot(2, 2, 2)
plt.plot(sampling, a_velocity_y, color='blue', linestyle='solid', linewidth = 1)
plt.title("$w_y$(t)")
plt.xlabel("t s")
plt.ylabel("$w_y$ deg/s")

plt.subplot(2, 2, 3)
plt.plot(sampling, a_velocity_z, color='blue', linestyle='solid', linewidth = 1)
plt.title("$w_z$(t)")
plt.xlabel("t s")
plt.ylabel("$w_z$ deg/s")

plt.subplot(2, 2, 4)
plt.plot(sampling, a_velocity_k, color='blue', linestyle='solid', linewidth = 1)
plt.title("$w_k$(t)")
plt.xlabel("t s")
plt.ylabel("$w_k$ deg/s")
plt.tight_layout()
```



The angular rotation values seem reasonable since ν is linear and the others are pretty close to linear.

In [197]:

```
B=R_Axis_Angle.subs({u_x:n[0],u_y:n[1],u_z:n[2]})
plt.rcParams['figure.figsize'] = [10, 10]
sampling=[]
psi_in_t=[]
theta_in_t=[]
phi_in_t=[]
nu_in_t=[]
for i in range(0,int(t)):
    B=R_Axis_Angle.subs({u_x:n[0],u_y:n[1],u_z:n[2]})
    sampling.append(i)
    nu=i*w_k
    B=B.subs({gamma:nu})
    theta_2=(asin(-B[2,0])).evalf()

    phi_in_t.append((acos(B[0,0]/cos(theta_2))/pi*180).evalf())
    psi_in_t.append((acos(B[2,2]/cos(theta_2))/pi*180).evalf())
    theta_in_t.append((theta_2/pi*180).evalf())
    nu_in_t.append(nu/pi*180)
B=R_Axis_Angle.subs({u_x:n[0],u_y:n[1],u_z:n[2]})
sampling.append(t)
nu=t*w_k
B=B.subs({gamma:nu})
theta_2=(asin(-B[2,0])).evalf()

phi_in_t.append((acos(B[0,0]/cos(theta_2))/pi*180).evalf())
psi_in_t.append((acos(B[2,2]/cos(theta_2))/pi*180).evalf())
theta_in_t.append((theta_2/pi*180).evalf())
nu_in_t.append(nu/pi*180)

plt.subplot(2, 2, 1)
plt.plot(sampling, psi_in_t, color='blue', linestyle='solid', linewidth = 1)
plt.title("$\psi$(t)")
plt.xlabel("t s")
plt.ylabel("$\psi$ deg")

plt.subplot(2, 2, 2)
plt.plot(sampling, theta_in_t, color='blue', linestyle='solid', linewidth = 1)
plt.title("$\\theta$(t)")
plt.xlabel("t s")
plt.ylabel("$\\theta$ deg")

plt.subplot(2, 2, 3)
plt.plot(sampling, phi_in_t, color='blue', linestyle='solid', linewidth = 1)
plt.title("$\phi$(t)")
plt.xlabel("t s")
plt.ylabel("$\phi$ deg")



plt.subplot(2, 2, 4)
plt.plot(sampling, nu_in_t, color='blue', linestyle='solid', linewidth = 1)
plt.title("$\\nu$(t)")
plt.xlabel("t s")
plt.ylabel("$\\nu$ deg")
plt.tight_layout()
```


ψ(t)
ψ deg
t s
θ(t)
θ deg
t s
ϕ(t)
ϕ deg
t s
ν(t)
ν deg
t s