Assignment 1: 3D Visualization of Satellites in Python

Abstract

In this assignment the student will learn how to apply Python to visualize satellites orbiting the Earth together with different reference frames and rotations between them.

1 Objective

After completing this assignment the student will learn:

- How to use Python to simulate satellites
- How to create reference frames and how they are defined relative to each other
- How to perform rotations
- How to create a gif animation in Python

2 Introduction

Before describing the steps needed to complete todays assignment, some fundamentals are needed. Specifically, core Python syntax, the fundamental mathematics needed to simulate a satellite flying in a circular orbit, and how the Earth is rotating around its $\mathbf{z}^i$ axis.

2.1 Python

Python applies the following syntax for programming.

2.1.1 If-Sentence

An if-sentence can be handled in Python as shown below which gives an output value that is limted between an upper and lower bound.

```python
# Example if sentence
value = 10
min_value = -5
max_value = 5

if value >= max_value:
    output = max_value
elif value <= min_value:
    output = min_value
else:
    output = value
```

2.1.2 For-Loop

A for loop can be created for a range of values as

```python
# Using a for loop with range
for number in range(1,6):
    print("This is number ", number)
```

Note that the range(1,6) gives a range between 1 and 5, as the start is inclusive, but stop is exclusive. It is also possible to loop through strings, e.g.

```python
# Using a for loop with a string
fruits = ["Apple", "Pear", "Banana"]
for fruit in fruits:
    print("The fruit is ", fruit)
```

2.1.3 While Loop

Similarly as a for loop, a while loop can be applied as

```python
# Using a while loop
value = 10
lower_bound = 0.1

while value > lower_bound:
    print("The value is ", value)
    value = value - 0.05

print("The final value is ", value)
```

2.1.4 Functions

A Python function can be created as follows:

```python
# Creating a function
def my_python_function(argument1, argument2):
    output1 = argument1 + argument2
    output2 = argument1 - argument2

    return output1, output2
```

Then the function can be called as

```python
# Calling the function
arg1 = 10
arg2 = 2
out1, out2 = my_python_function(arg1, arg2)
```

allowing access to the variables out1 and out2. In terms of Python, the key item to keep control of is the tabulation as all lines within a function must be tabulated one step. This allows functions within functions as long as the tabulation is maintained.

2.1.5 Numpy

A matrix can be constructed using the Numpy library as

```python
# Creating a matrix
import numpy as np
matrix = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
```

and a vector can be created as

```python
# Creating the vector
vector = np.array([1,2,3])
```

and they can be multiplied together using either @ or *.dot*() as

```python
# Matrix-Vector product
new_vector = matrix @ vector
new_vector = matrix.dot(vector)
```

2.2 Reference Frames

In todays assignment there are three reference frames that will be used, namely

- Body frame, which is fixed to the satellite and moves with the satellite and is denoted by superscript b.
- Earth Centered Inertial (ECI) frame is fixed to the Earth where $\mathbf{x}^i$ points towards the Vernal Equinox, $\mathbf{z}^i$ goes through the North pole, while $\mathbf{y}^i$ completes the right handed system. The ECI frame is denoted by superscript i and is considered an inertial reference frame where the laws of Newton are valid.
- Earth Centered Earth Fixed (ECEF) frame is initially aligned with the ECI frame, and rotates with Earth's rotational speed. The ECEF frame is denoted by superscript e.

2.3 Orbital Mechanics of a Circular Orbit

The position to a point in a circular orbit that is defined in the $\mathbf{y}^i - \mathbf{z}^i$ plane can be described as

$$x = 0 \tag{1}$$

$$y = r\cos(\varphi) \tag{2}$$

$$z = r\sin(\varphi) \tag{3}$$

where r is the radius from the center of the Earth to the location of the satellite. In a circular trajectory both the radius and speed will be constant. The angle to the satellite is denoted φ and describes where in the orbit the satellite is (in the elliptic case this can be compared to the true anomaly). The first step to find the behaviour of the satellite is to differentiate the above equations such that

$$\dot{x} = 0 \tag{4}$$
$$\dot{y} = -r\dot{\varphi}\sin(\varphi) \tag{5}$$
$$\dot{z} = r\dot{\varphi}\cos(\varphi) \tag{6}$$

which describes the velocity vector of the satellite. It is known that the speed is equal to the magnitude of the velocity vector, such that we have that

$$V^2 = \dot{x}^2 + \dot{y}^2 + \dot{z}^2 = (-r)^2\dot{\varphi}^2\sin^2(\varphi) + r^2\dot{\varphi}^2\cos^2(\varphi) = r^2\dot{\varphi}^2 \tag{7}$$

giving an equation for the change in angle as

$$\dot{\varphi} = \frac{V}{r}. \tag{8}$$

For circular orbits, the velocity vector will always be perpendicular to the radius vector, such that the speed can be found as (Sidi, 1997, p15)

$$V = \sqrt{\frac{\mu}{r}} \tag{9}$$

with $\mu = G \cdot M_{Earth}$, where $G = 6.669 \cdot 10^{-11}$ is the gravitational constant and $M_{Earth} = 5.9742 \cdot 10^{24}$ is the mass of the Earth. This gives a constant speed for circular orbits and a consequently constant angular rate which then allows the angle to the satellite to be updated to calculate the position at any time.

2.4 Simple Rotations

To facilitate visualization of change in orientations, it is common to use rotation matrices, where simple rotations are defined as

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{10}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{11}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

where ϕ, θ, and ψ are the roll, pitch and yaw angles respectively.

2.5 Simulating Earth's Rotation

Earth Centered Earth Fixed (ECEF) is denoted by superscript e and is initially aligned with the ECI frame and moves with the rotation of the Earth around the $\mathbf{z}^i$. The angular speed of the Earth is $\omega_{i,e}$, enabling the rotation matrix from ECEF to ECI to be found as

$$\mathbf{R}_e^i = \begin{bmatrix} \cos(\omega_{i,e}t) & -\sin(\omega_{i,e}t) & 0 \\ \sin(\omega_{i,e}t) & \cos(\omega_{i,e}t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where the angular speed can be found as $\omega_{i,e} = \frac{2\pi}{24\cdot 60\cdot 60} = 7.2722 \cdot 10^{-5}$ rad/s.

2.6 Recommended Notation

When dealing with rotation matrices in software, it is easy to lose track of what is rotating from and to. It is therefore recommended to apply the following notation for the following formula:

$$\mathbf{v}^{to} = \mathbf{R}_{from}^{to} \mathbf{v}^{from} \tag{13}$$

as

```python
# Example showing how to perform rotations in Python
import numpy as np
v_to = R_to_from @ v_from
```

such that a rotation from ECEF to ECI frame, $\mathbf{v}^i = \mathbf{R}_e^i \mathbf{v}^e$, results in

```python
# Example showing how to calculate a vector in ECI frame
import numpy as np
v_i = R_i_e @ v_e
```

This allows rotation matrices and (later) quaternions to apply the same notation which makes things much easier to keep track of. Note that here numpy is used to allow matrix/vector multiplications.

2.7 Forward Euler's Method

Forward Euler's method is defined as

$$x_k = x_{k-1} + hf(t, x_{k-1}) \tag{14}$$

where x_k is the current state, h is the step size, while $\dot{x} = f(t, x_{k-1})$. This means that if we know the change of a state, i.e. its velocity, we can apply integration using this technique and will be deployed in this assignment to create the animation.

3 Visualization

3.1 Setting up the Integrated Development Environment (IDE)

Setting up the IDE

1. Start by installing Anaconda on your computer `https://www.anaconda.com/download`
2. Run Anaconda Prompt as administrator and run the following commands
   - conda create -n pyvista_env python=3.11
   - conda activate pyvista_env
   - conda install -c conda-forge numpy matplotlib pyvista imageio pillow spyder scipy
3. Run Spyder from your environment by writing ”spyder”. Spyder will be used to develop the code. This approach removes many issues that arise if you work directly on the user environment.

3.2 Spyder

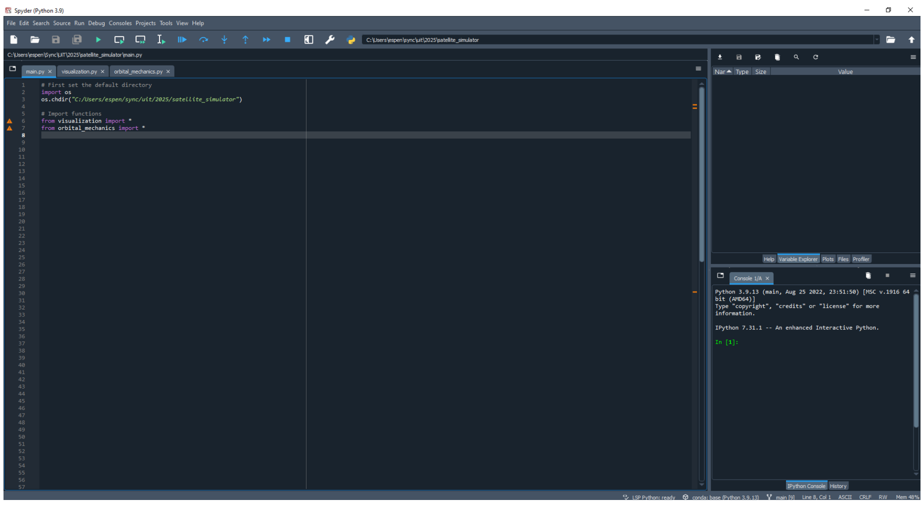
Figure 1 shows the main interface of the Spyder IDE. As can be seen, three main files have been created:

- `main.py` will be our main access point to the code
- `visualization.py` will contain our functions used to perform 2D and 3D visualization of satellites
- `orbital_mechanics.py` will contain our orbital mechanics functions.

After creating a Python script, the code can be executed by pressing Shift + Enter or pressing the Run button.

3.3 Pyvista

There are many libraries that allow 3D visualization, but one of the best is Pyvista built on top of the Visualization Toolkit (VTK) and its details can be found here `https://pyvista.org/`. It is worth mentioning that Matplotlib - which is an excellent 2D plotting framework, is not suited at all for 3D visualizations, as it is not true 3D, and you end up with distorted perspectives. To that end, this course applies Pyvista to do the 3D visualizations while Matplotlib is used for 2D plots. The API for Pyvista is detailed here: `https://docs.pyvista.org/`.


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\espen\sync\uit\2025\satellite_simulator
C:\Users\espen\Sync\UiT\2025\satellite_simulator\main.py
main.py
visualization.py
orbital_mechanics.py
Help
Variable Explorer
Plots
Files
Profiler
Console 1/A

```
# First set the default directory
import os
os.chdir("C:/Users/espen/sync/uit/2025/satellite_simulator")

# Import functions
from visualization import *
from orbital_mechanics import *
```

```
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64
bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 7.31.1 -- An enhanced Interactive Python.

In [1]:
```

Figure 1: Spyder IDE

Creating the First 3D Plots

1. Go to `https://docs.pyvista.org/examples/` and select "Geometric Objects".
2. Create a function in Python in `visualization.py` called "geometry_examples()" that executes the code.
3. Import the new function into your `main.py` and run it using geometry_examples() and show that you are able to obtain the same result as Figure 2. Note that you can play around with colors, and interact with the figure to change orientation of objects.

3.4 Creating Reference Frames

The geometry examples show how to create an arrow object. A reference frame is simply three different arrows with a given orientation. To enable Latex labeling, add the following to the beginning of your visualization script.[1]

```
# Enable LaTeX rendering and importing the xcolor package
plt.rcParams['text.usetex'] = True
plt.rcParams['text.latex.preamble'] = r'\usepackage{xcolor}'
```

[1]Latex requires Miktex `https://miktex.org/download`, and Ghostscript `https://www.ghostscript.com/` to create Latex symbols.

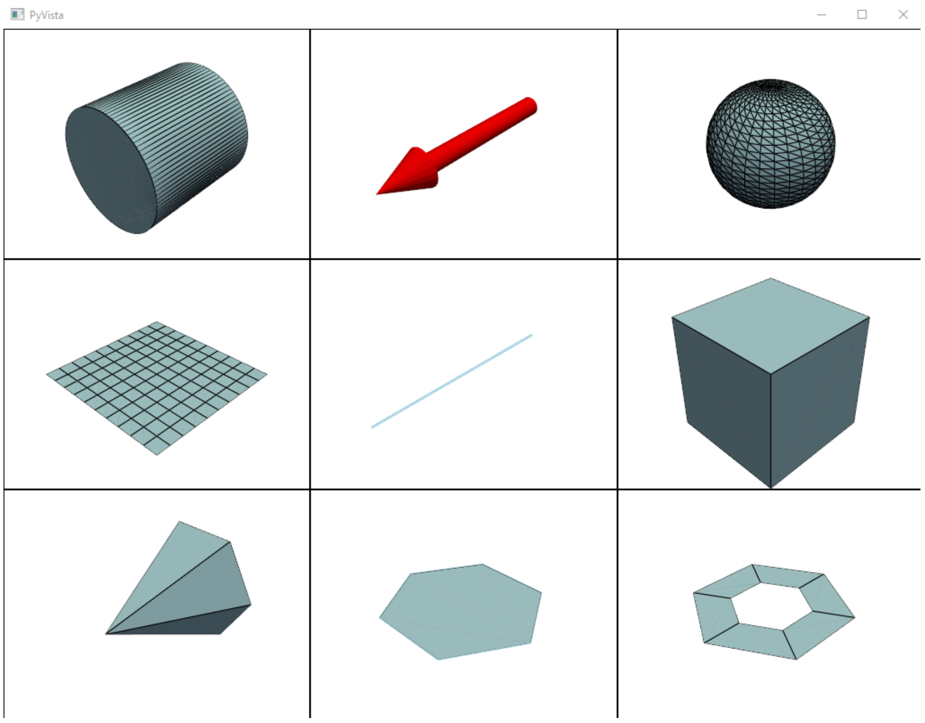


Figure 2: Geometry examples of primitives such as cylinders, arrows, spheres, cubes, surfaces and so on.

Creating a Reference Frame

1. Create a function called create_reference_frame(plotter, labels, scale = 1) which returns a reference_frame_mesh object.

2. Specify one arrow for each axis using the unit direction vector (1,0,0), (0,1,0) and (0,0,1). E.g. the x-axis arrow looks like

```
    x_arrow = pv.Arrow(start=(0,0,0),
                       direction=(1,0,0),
                       tip_length=0.25,
                       tip_radius=0.1,
                       tip_resolution=20,
                       shaft_radius=0.05,
                       shaft_resolution=20,
                       scale=scale,
                       )
```

3. The reference frame mesh can then be created as

```
# Adding the reference frame components
reference_frame_mesh = {"scale" : scale,
    "x": plotter.add_mesh(x_arrow, color='red',
    show_edges = False),
    "y": plotter.add_mesh(y_arrow, color='blue',
    show_edges = False),
    "z": plotter.add_mesh(z_arrow, color='green',
    show_edges = False),
    "x_label": pv.Label(text=labels[0], position=np.
    array([1,0,0])*scale, size=20),
    "y_label": pv.Label(text=labels[1], position=np.
    array([0,1,0])*scale, size=20),
    "z_label": pv.Label(text=labels[2], position=np.
    array([0,0,1])*scale, size=20)}
```

where we are using labels as an input to put labels that are positioned along the different axes to show the difference between the different reference frames. Scale can also be used to enable easy visualization of differences. Note that the mesh, labels and the scale are properties with the reference frame mesh, as they all can be different for different reference frames, and when performing live rotations, they require updates.

The function you created should now be callable using the commands

```
plotter = pv.Plotter(off_screen=False)
eci_frame = create_reference_frame(plotter, labels=np.array(["$
    \mathbf{x}^i$", "$\mathbf{y}^i$", "$\mathbf{z}^i$"]), scale
    =2)
plotter.add_actor(eci_frame["x_label"])
plotter.add_actor(eci_frame["y_label"])
plotter.add_actor(eci_frame["z_label"])
plotter.show()
```

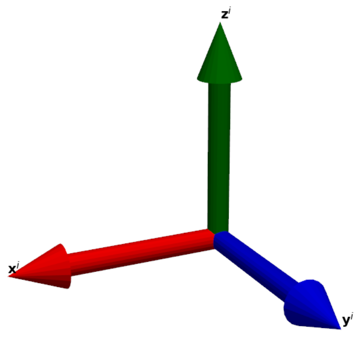


Figure 3: ECI reference frame with labels

with results as shown in Figure 3. The add_actor() for the different labels are required to ensure that the positioning of the labes are updated during animations.

3.5 Creating the Satellite

To visualize the satellite, it is possible to use the primitives as shown above. Namely using cubes and cones that can be visualized using the add_mesh() function of the plotter object.

Creating the Satellite

1. Create a function called create_satellite(plotter, size=0.5) where plotter is the pyvista plotter object, while size is an argument that allows for scaling of the satellite.
2. Create the main satellite body using

   ```
   #Creating the satellite body
   body_b = pv.Box(bounds=(-size,size,-size,size,-size,
       size))
   ```

   and it should look like a cube.
3. Now create the solar panels Define three variables, panel_width, panel_length, panel_thickness to represent a single solar panel and let them be a function of size such that they scale with the satellite. You can now create the solar panel using the box function as

   ```
   # Creating the solar panel
   solar_panel_b = pv.Box(bounds=(-panel_thickness/2,
       panel_thickness/2, -panel_width/2,panel_width/2,
       -panel_length/2,panel_length/2))
   ```

4. Create a scientific instrument (e.g. a camera) mounted along the $\mathbf{x}^b$ axis.

   ```
   # Creating a scientific instrument to be along the
       x_b axis
   scientific_instrument_b = pv.Cone(center=(size
       -0.01,0,0), direction=(-1,0,0), height=0.5*size,
       radius=0.5*size, resolution=50)
   ```

5. You can now visualize the satellite using plotter.add_mesh(body_b) and similarly for the other two objects

3.5.1 Applying Textures to the Satellite

A texture is a 2D picture that needs to be mapped onto a 3D surface. To address this, a u-v mapping is needed telling the software what elements to put on which surface. To that end, create the u-v mapping for the satellite as follows.

Applying Textures to the Satellite

1. Download the satellite textures:

   - satellite_body_texture.png
   - solar_panel_texture.png
   - camera_texture.png

   and place them in the same folder as the code.

2. Create the texture objects using pv.read_texture(”texture.png”) function. Giving

   ```
   # Creating a scientific instrument to be along the
       x_b axis
   satellite_texture = pv.read_texture("
       satellite_texture.png")
   ```

   and similarly for the other two textures with fitting names.

3.
   ```
   # Ensure that the body has texture coordinates
   u = np.array([0, 1, 1, 0] * 6)
   v = np.array([0, 0, 1, 1] * 6)
   texture_coordinates = np.c_[u, v]
   body_b.texture_map_to_plane(inplace=True)
   ```

4. For the solar panels the texture map can be created as

   ```
   # Ensure that the panels have texture coordinates
   solar_panel_b.texture_map_to_plane(origin=(-size -
       panel_thickness/2, 0, 0), point_u=(-size -
       panel_thickness/2, panel_width/2, 0), point_v=(-
       size - panel_thickness/2, 0, panel_length/2),
       inplace=True)
   ```

5. And for the scientific instrument

   ```
   scientific_instrument_b.texture_map_to_sphere(inplace
       =True)
   ```

6. The complete satellite can now be created by assemblying the different components

   ```
   satellite_mesh = {
       "Body": plotter.add_mesh(body_b, texture=
       satellite_texture, show_edges=True),
       "Solar Panels": plotter.add_mesh(solar_panel_b,
       texture=solar_panel_texture, show_edges=True),
       "Scientific Instrument": plotter.add_mesh(
       scientific_instrument_b, texture=camera_texture,
       show_edges=False)}
   ```

The satellite function should now be possible to use as

```
# Calling the create_satellite function
satellite_mesh = create_satellite(plotter, 2)
```

where the size of the satellite can be varied.

3.6 Creating the Earth

To create the Earth, we take basis in one of the examples from Pyvista at `https://docs.pyvista.org/examples/99-advanced/planets.html`

```
# Creating the Earth mesh function
from pyvista import examples

def create_earth(plotter, radius):

    earth = examples.planets.load_earth(radius=radius)
    earth_texture = examples.load_globe_texture()
    earth_mesh = plotter.add_mesh(earth, texture=earth_texture,
     smooth_shading=True)

    return earth_mesh
```

3.7 Pyvista Rotations

The satellite body will change its attitude (or orientation) over time. The same is true for the Earth that is rotating around its $\mathbf{z}^i$ axis. One specific feature of the Pyvista framework, is that the rotation order of rotation matrices is in the order (y-x-z), and not z-y-x as is perhaps more common. To that end, the Pyvista rotation matrix can be constructed as

```
def pyvista_rotation_matrix_from_euler_angles(orientation_euler
    ):
    # Pyvista rotates in the order y-x-z
    phi = orientation_euler[0]*np.pi/180
    theta = orientation_euler[1]*np.pi/180
    psi = orientation_euler[2]*np.pi/180

    R = rotation_z(psi).dot(rotation_x(phi)).dot(rotation_y(
    theta))

    return R
```

Remember that rotation order starts from the right, i.e. rotation around y-axis is performed first, then x, and last the z-axis.

Constructing the Rotation Matrices

1. Create the functions with basis in Equations (10)-(12):
   - rotation_x(phi)
   - rotation_y(theta)
   - rotation_z(psi)
2. Implement the pyvista_rotation_matrix_from_euler_angles(orientation_euler) function and show that you are able to rotate vectors.

3.8 Updating the Satellite Pose

Pose is defined as the combination of position and attitude. Let the Euler angles be defined as $\mathbf{\Theta}_{i,b} = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^\top$ representing the attitude of the body frame relative to the inertial frame, while $\mathbf{r}^i$ is the radius vector of the satellite body relative to the center of the Earth. Then the satellite pose can be updated using the following function:

```python
def update_satellite_pose(satellite_mesh, r_i, Theta_ib):
    satellite_mesh["Body"].SetPosition(r_i)
    satellite_mesh["Solar Panels"].SetPosition(r_i)
    satellite_mesh["Scientific Instrument"].SetPosition(r_i)

    satellite_mesh["Body"].SetOrientation(Theta_ib)
    satellite_mesh["Solar Panels"].SetOrientation(Theta_ib)
    satellite_mesh["Scientific Instrument"].SetOrientation(
   Theta_ib)
```

Updating Satellite Pose

1. Implement the satellite pose function which accepts Euler angles in degrees as input.

3.9 Rotating the Earth

The Earth has an angular speed around the $\mathbf{z}^i$ as detailed in Section 2.5. One thing to note is that Pyvista uses degrees, while the equation for angular speed was done in rad/s.

Updating Earth Orientation

1. Create a function update_earth_orientation(earth_mesh, t) that calculates the angular speed in deg/s and updates the orientation of the object earth_mesh as function of time.

3.10 Updating Reference Frames

Only the ECI reference frame is considered to be stationary or inertial in this assignment. This means that both the body frame and the ECEF frame must be updated through simulations.

Updating reference frames

1. Create the function update_body_frame_pose(body_frame, r_i, Theta_ib) with body_frame mesh, radius vector and Euler angles in degrees as input.
   - The orientation can be set using SetOrientation function
   - The position can be set using SetPosition function
   - The labels require special care and their position can be found using

     ```
     # Update label positions
     body_frame["x_label"].position = r_i + R_i_b.dot(
         np.array([1,0,0]))*body_frame["scale"]
     ```

     and similarly for the $\mathbf{y}^b$ and $\mathbf{z}^b$ axes (with corresponding unit vectors). Here $\mathbf{R}_b^i$ is the rotation matrix from body to inertial using Pyvista rotation order.
2. Create the function update_ecef_frame_orientation(ecef_frame, t) with ecef_frame mesh and time as input.
   - The orientation can be set using SetOrientation function where the angle can be found as $\omega_{i,e}t$ measured in degrees and applied to the $\mathbf{z}^e$ axis.

     ```
     # Rotate reference frames
     ecef_frame["x"].SetOrientation([0,0,w_ie_i*t])
     ```

   - The labels require special care and their position can be found using

     ```
     # Update label positions
     ecef_frame["x_label"].position = rotation_z(
         w_ie_i*np.pi/180*t).dot(np.array([1,0,0]))*
         ecef_frame["scale"]
     ```

     and similarly for the $\mathbf{y}^e$ and $\mathbf{z}^e$ axes (with corresponding unit vectors).

3.11 Complete Scene

To allow visualization of the satellite relative to the Earth, its size can be made much larger than what it is in real life. Also, the orbit is commonly very low, e.g. 500km altitude, which then makes it very difficult to discern the orbit. To that end, let the orbit, satellite, and reference frames be functions of Earth's radius to allow them to scale relative to Earth.

Complete Scene

1. With the functionality that has been created up until now, create a function called visualize_scene() where you implement the following
   - Create Earth Mesh in the center with radius equal to Earth's radius
   - Create a satellite at a distance 3x Earth's radius along the $\mathbf{y}^i$ axis with an orientation of $\mathbf{\Theta}_{i,b} = \begin{bmatrix} 90 & 45 & 0 \end{bmatrix}^\top$. Let the size of the satellite be 0.1x Earth's radius to allow it to be easy to see.
   - Create the ECI reference frame with a size of 2x Earth's radius
   - Create the ECEF reference frame with a size of 1.5x Earth's radius
   - Create the body reference frame with a size of 0.5x Earth's radius
   - Add labels to each reference frame using add_actors() function
   - Update the satellite pose
   - Update the body orientation
2. Show that you are able to recreate the scene showed in Figure 4

3.12 Implement Circular Orbital Mechanics

It is now time to move over to orbital mechanics for circular orbits which are relatively easy to work with. Vectors are to be created using numpy arrays.

Complete Scene

1. Create the `orbital_mechanics.py` file
2. Create the function calculate_circular_angular_speed(r) with radius as an input using Equation (8).
3. Create calculate_satellite_position_in_circular_orbit(varphi, r) with angle to satellite and radius as input, returning the radius vector in inertial frame with basis in Equations (1)-(3).
4. Create the function calculate_circular_orbit_velocity(varphi, r) with angle to satellite and radius as input returning the velocity vector in inertial frame with basis in Equations (4)-(6).

3.13 Create a Gif Animation

In the following code, Forward Euler's method is used on φ simply as $\varphi_k = \varphi_{k-1} + \Delta t \dot{\varphi}$ to change the position in the orbit.

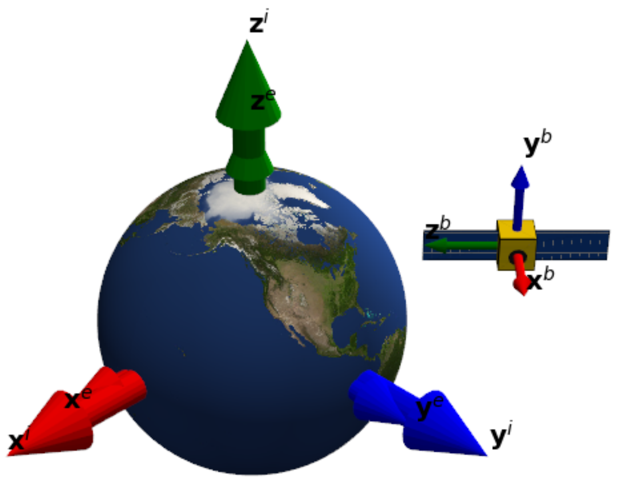


Figure 4: Complete scene with the different components

To create a gif animation, we can use the two plotter functions: open_gif() and write_frame(). Then, in between, we can do all the calculations needed to update the satellite pose, the orientation of the ECEF frame and the rotation of the Earth. A pseudo-code that facilitates animation to be stored in a gif is given below and can be populated based on the functionality developed in this assignement.

```
# Initialize the scene

# Initialize the orbit angle and time step

# Initialize the gif
plotter.open_gif("satellite_animation.gif")

n_frames = 100
simulation_time = 24*60*60
frame_interval = simulation_time // n_frames #integer division

for t in np.arange(0, simulation_time, time_step):
    # Extract radius vector
    # Calculate orbit angle
    # Apply Euler Forward to orbit angle

    # Only update on each frame
    if t % frame_interval == 0:
        # apply changes to the satellite
        # apply changes to the body frame
        # apply changes to the ECEF frame
        # apply changes to the Earth

        # Store the current scene
        plotter.write_frame()

# Closing and finalizing the gif
plotter.close()
```

Animation and Reporting

1. Take basis in the visualize_scene() and create a new function animate_circular_orbit() and create code needed to create a gif animation. Show that you are able to create a gif animation of the satellite orbiting the Earth where Earth is rotating around its own axis.

2. Write a report based on the work explaining what you have done, and show the results of the animation, e.g. screenshots at different timesteps to show that you are able to get it to move in a circular trajectory around the Earth.