CMPE 330 Assignment 1

Genevieve Hayes
October 14, 2020

The script **main.m** will call the testing functions for Assignment 1. All functions are testing functions are located in the same folder as main.m. To run each question's tests independently within main.m, use the "Run Section" option with the cursor next to the desired question.

Question 1: Intersect-Two-Lines

```
function [M,dM] = closestIntersectionOf2Vectors(P1,v1,P2,v2)
```

Method

- Check if lines are parallel by taking the cross product of their direction vectors.
  - If the lines are parallel ($v_2 \times v_1 = 0$)return [NaN, NaN, NaN] for intersection and error.
  - If the lines are not parallel, create v_3, V and P such that:

$$v_3 = v_2 \times v_1$$

$$V = \begin{bmatrix} -v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$P = \begin{bmatrix} P_{1x} - P_{2x} \\ P_{1y} - P_{2y} \\ P_{1z} - P_{2z} \end{bmatrix}$$

Using the formula of a line $L = P + tV$, determine t of closest intersection by

$$t_{int} = V^{-1} \times P$$

Then compute the location on the line of closest intersection, $L_1 = P_1 + t_{int}V_1$ and $L_2 = P_2 + t_{int}V_2$. The midpoint between these gives the closest intersection point:

$$M = \frac{(L_2 + L_1)}{2}$$

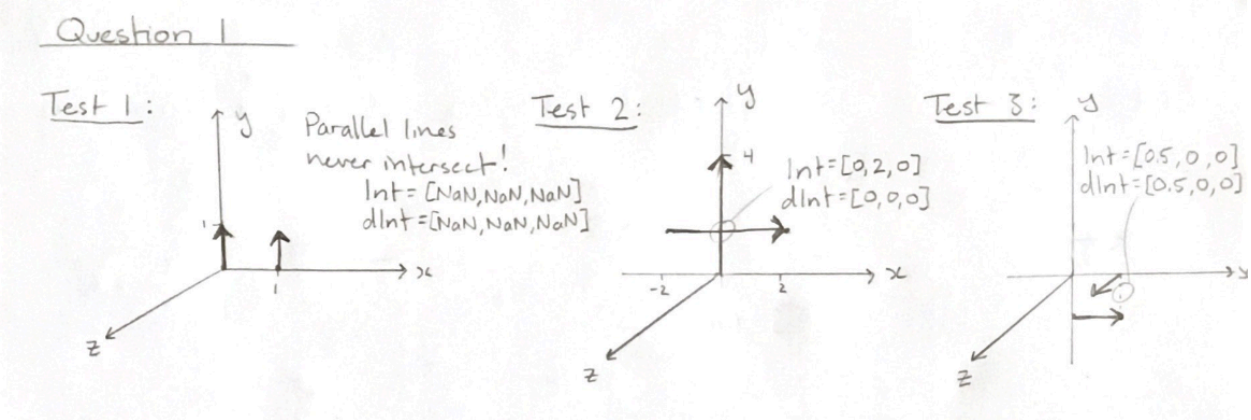
with error:

$$dM = abs\left(\frac{(L_2 - L_1)}{2}\right)$$

Testing

- Test 1: Case of 2 parallel unit vectors. The expected output is [NaN, NaN, NaN] for both M and dM.
- Test 2: Case of perpendicular, intersecting vectors. The expected output is an intersection at y = 2 (M = [0,2,0]) with dy = 0 (dM = [0,0,0]).

- Test 3: Case of vectors that never intersect. Their closest approach occurs when L1 = [0,0,0] and L2 = [1,0,0]. The expected approximate intersection is [0.5,0,0] with error [0.5,0,0].



Question 2: Intersect-Line-and-Plane

```
function Int = intersectionOfVectorWithPlane(A,n,P,v)
```

Method

- Check if line is parallel with plane by taking the cross product of the normal vector and the direction vector.
  - If they are parallel return [NaN, NaN, NaN] for intersection.
  - If they are not parallel, compute their time of intersection, t using:

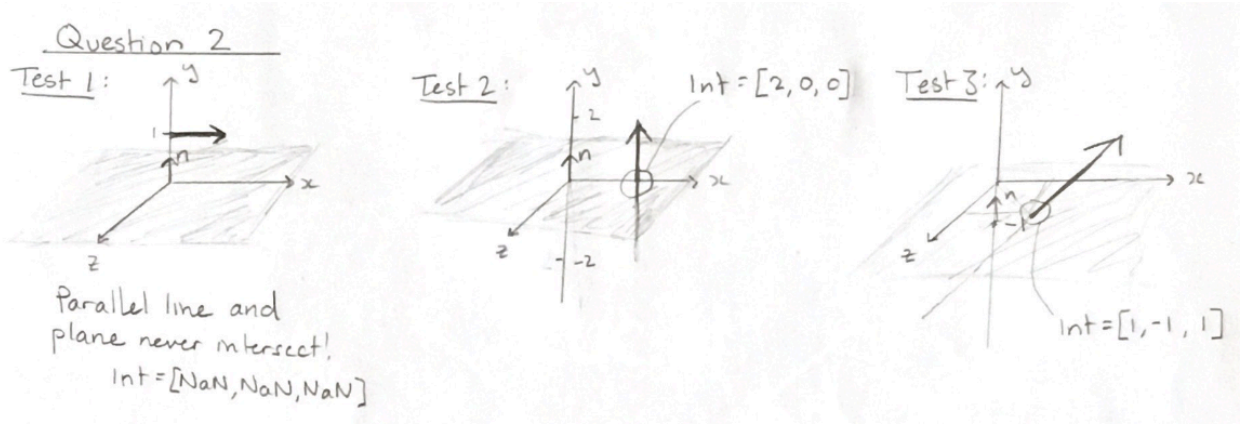
$$t_{int} = \frac{(A - P) \cdot n}{v \cdot n}$$

Then compute the intersection point:

$$Int = P + tv$$

Testing

- Test 1: Case of vector parallel to plane. The expected output is [NaN, NaN, NaN] for the intersection.
- Test 2: Case of Y direction vector perpendicular to X plane, intersecting at x = 2 (Int = [2,0,0]).
- Test 3: Case of Y-Z direction vector passing through a plane below the X – axis. The expected intersection is at x = 1, y = -1, z = 1 (Int = [1,-1,1]).



Question 3: Intersect-Line-and-Ellipsoid

```
function [numInt,Int1,Int2] = intersectionOfVectorWithEllipsoid(P,v,a,b,c)
```

Method

- Construct the equation of the line:

$$L(t) = P + tv$$

- The equation for the ellipsoid is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

  To find the time that the intersection occurs between the line and the ellipsoid, find the solution for t that satisfy:

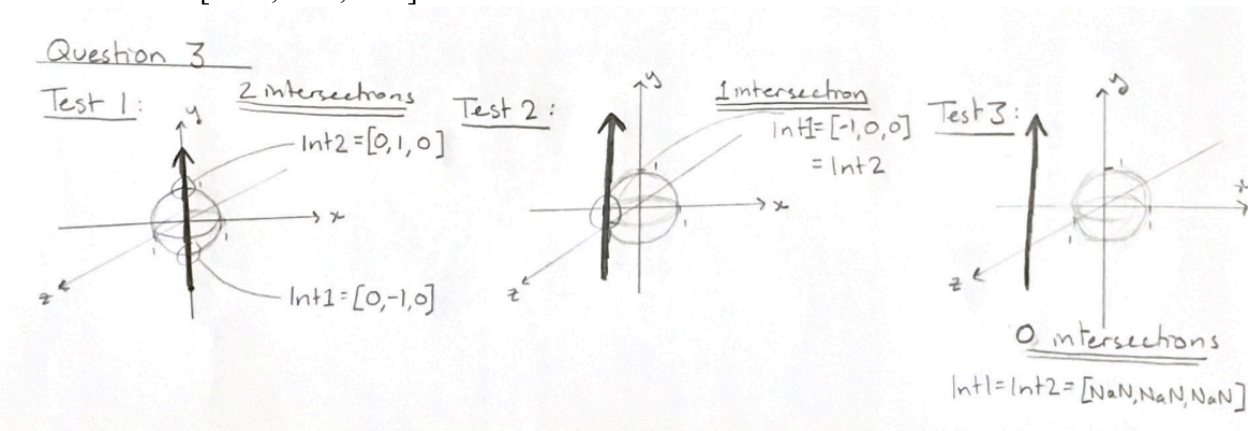
$$\frac{L(t)_x^2}{a^2} + \frac{L(t)_y^2}{b^2} + \frac{L(t)_z^2}{c^2} = 1$$

- Check if an intersection occurs between the line and the ellipsoid. If t_{int} has no real number solutions then no intersections occur and the function returns Int1 = Int2 = [NaN, NaN, NaN].
- If t_{int} has real solutions then substitute them into $L(t_{int})$ to get Int1 and Int2.
- If Int1 is equal to Int2, then only 1 intersection occurs. Otherwise there are 2 distinct intersection between the line and the ellipsoid.

Testing

- Test 1: Case of 2 intersections with the ellipsoid. The expected output is numInt = 2, Int1 = [0,-1,0] and Int2 =[0,1,0].
- Test 2: Case of 1 intersection with the ellipsoid. The expected output is numInt = 1, Int1 = Int2 =[-1,0,0].

- Test 3: Case of 0 intersections with the ellipsoid. The expected output is numInt = 0, Int1 = Int2 =[NaN,NaN,NaN].



Question 4: Number-of-Intersections-between-Sphere-and-Cylinder

```
function numInt = numIntersectionsOfSphereAndCylinder(C,R,r,P,v)
```

Method

- Create a unit vector perpendicular to v, the direction vector of the cylinder by solving $v_x \cdot 1 + v_y \cdot 1 + v_z \cdot v_{2z} = 0$. Then $v_2 = \frac{[1,1,v_{2z}]}{norm(v_2)}$
    - Create v_3, V and P such that:

$$v_3 = v_2 \times v_1$$

$$V = \begin{bmatrix} -v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$P = \begin{bmatrix} P_{1x} - P_{2x} \\ P_{1y} - P_{2y} \\ P_{1z} - P_{2z} \end{bmatrix}$$

Using the formula of a line $L = P + tV$, determine t of closest intersection by

$$t_{int} = V^{-1} \times P$$

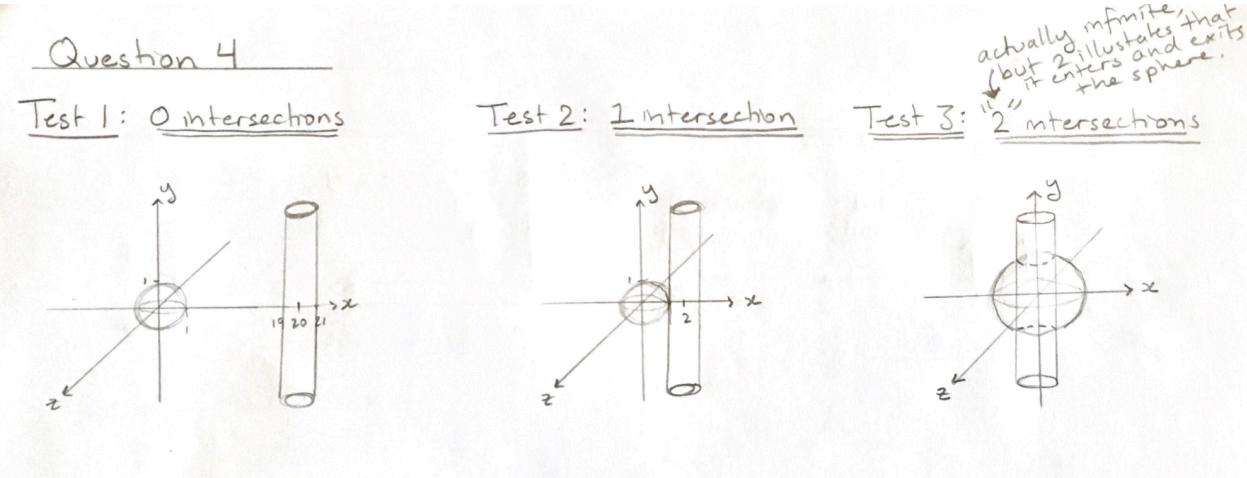
Then compute the location on the line of closest intersection of the line with the sphere center, $L_1 = P_1 + t_{int} V_1$.

Compute the distance $d = abs(L_1 - C)$.

Compute the value $val = \|d\| - R - r$. If val < 0 then there are "2" intersections between the cylinder and the sphere, i.e. the cylinder enters and exits the sphere at different locations. If val = 0 then there is exactly 1 intersection between the cylinder and the sphere. If val > 0, then there are no intersections between the cylinder and the sphere.

Testing

- Test 1: Case of zero intersections between sphere and cylinder. The function return numInt = 0.
- Test 2: Case of single point intersection between sphere and cylinder. Sphere is the unit vector centered at the origin, the cylinder has radius 1 centered at [2,0,0] and extends in the y-direction. The function return numInt = 1.
- Test 3: Case where cylinder enters and exits the sphere at different locations. In this case the function return numInt = 2.



Question 5: Reconstruct-Sphere-From-Points

```
function [C,R] = reconstructSphereFromPoints(pointsMatrix)
```

Method

- This function uses a simplified form of least-squares fitting to reconstruct a sphere with defined center and radius best fit to the points. It is simplified because it does not minimize the residuals. It requires at least 4 or more 3-dimensional points as input.
- The sphere equation can be written as $(x-a)^2+(y-b)^2+(z-c)^2=r^2$ where a, b, c and r are the unknowns and x, y and z are the data points.
- For each data point, the above equation can be rewritten as a linear equation containing with a, b, c and r as variable:

$$2ax - 2by - 2cz + (a^2 + b^2 + c^2 - r^2) = -(x^2 + y^2 + z^2)$$

Let $D = (a^2 + b^2 + c^2 - r^2)$, then the equation become:

$$2ax - 2by - 2cz + D = -(x^2 + y^2 + z^2)$$

This can be simplified further using matrices:

$$A * Y = B$$

$$Y = B * A^{-1}$$

Where the first 3 values of the column vector Y are the center of the sphere, C. The last value of Y is equal to D, therefore the radius, R can be found using:

$$R = \sqrt{C_x^2 + C_y^2 + C_z^2 - Y_4}$$

The output center values and radius were rounded to 5 decimal places.

Testing

The MATLAB function [X, Y, Z] = sphere was used to generate 20x20 patch test spheres.

- Test 1: Case of unit sphere (R=1) centered at the origin (C = [0,0,0]).
- Test 2: Case of unit sphere (R=1) centered at C = [1,1,0].
- Test 3: Case of sphere with radius R=5 centered at C = [3,3,3].

Question 6: Generate-Random-Unit-Vector

```
function [vector] = generateRandomUnitVector(dim)
```

Method

- All random number generation for this function was done using the MATLAB function rand().
- First check if the dimension specified is 2 or 3, otherwise return an error.
    - If the dimension is 2, generate a random number θ between 0 and 2π. A random 2D unit vector centered at the origin is then given by

$$vector_{2D} = [\cos(\theta), \sin(\theta)]$$
$$vector_{2Dnorm} = \frac{\text{vector}_{2D}}{\|\boldsymbol{v}ector_{2D}\|}$$

    - If the dimension is 3, generate a random number θ between 0 and 2π and a random number z between 1 and -1. A random 3D unit vector centered at the origin is then given by

$$vector_{3D} = [\sqrt{1-z^2}\cos(\theta), \sqrt{1-z^2}\sin(\theta), z]$$
$$vector_{3Dnorm} = \frac{\text{vector}_{3D}}{\|\boldsymbol{v}ector_{3D}\|}$$

Testing

- Generate 200 2-dimensional vectors using the generateRandomUnitVector(2) function then plot them from the origin to demonstrate a graph of the 2D unit circle.
- Generate 200 3-dimensional vectors using the generateRandomUnitVector(3) function then plot them from the origin to demonstrate a graph of the 3D unit sphere.

Question 7: Generate-Orthonormal-Frame

```
function [Oe,e1,e2,e3] = generateOrthonormalFrame(A,B,C)
```

Method

- Use cross products to create 3 pairwise orthogonal vectors:

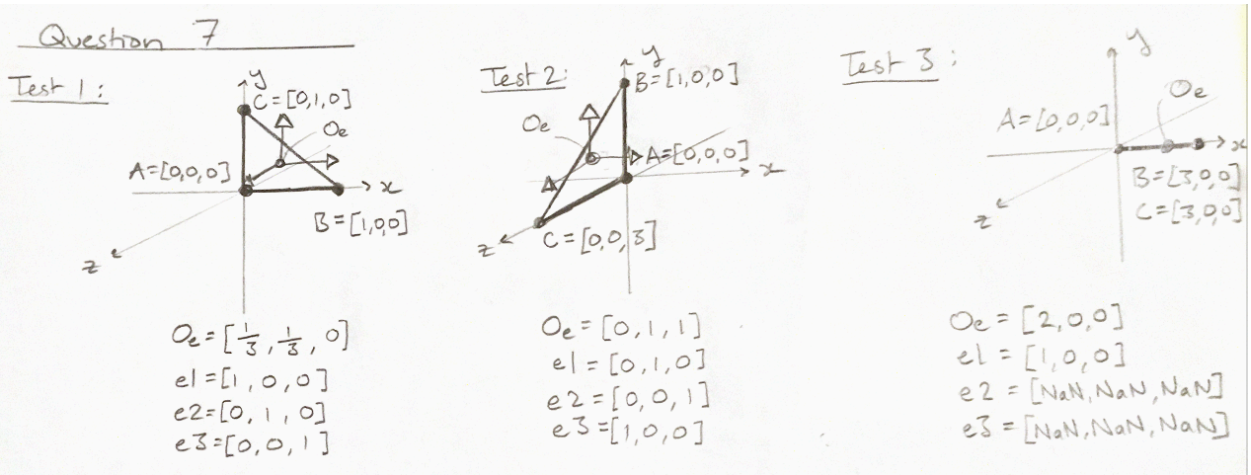
$$i = B - A$$
$$k = i \times (C - A)$$
$$j = k \times i$$

- Normalize each orthogonal vector to create an orthonormal frame (i, k, j).
- Take the Center of Gravity to be the center of the base frame:

$$O_e = \frac{A + B + C}{3}$$

Testing

- Test 1: Case of the origin and x, y unit vectors creating the triangle with corners at A = [0,0,0], B = [1,0,0], C = [0,1,0].
- Test 2: Case of an equilateral triangle centered about the point [0,1,1] with corners at A = [0,0,0], B = [0,3,0], C = [0,0,3].
- Test 3: Case of colinear input points A = [0,0,0], B = [3,0,0], C = [3,0,0]. This is a degenerate case that returns [NaN,NaN,NaN] for at 2 of the base vectors.



Question 8: Rotation-About-Frame-Axis

```
function [R3by3,R4by4] = rotationMatrixAboutFrameAxis(axis,angle)
```

Method

- Given θ, the 3x3 rotation matrices are found using:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

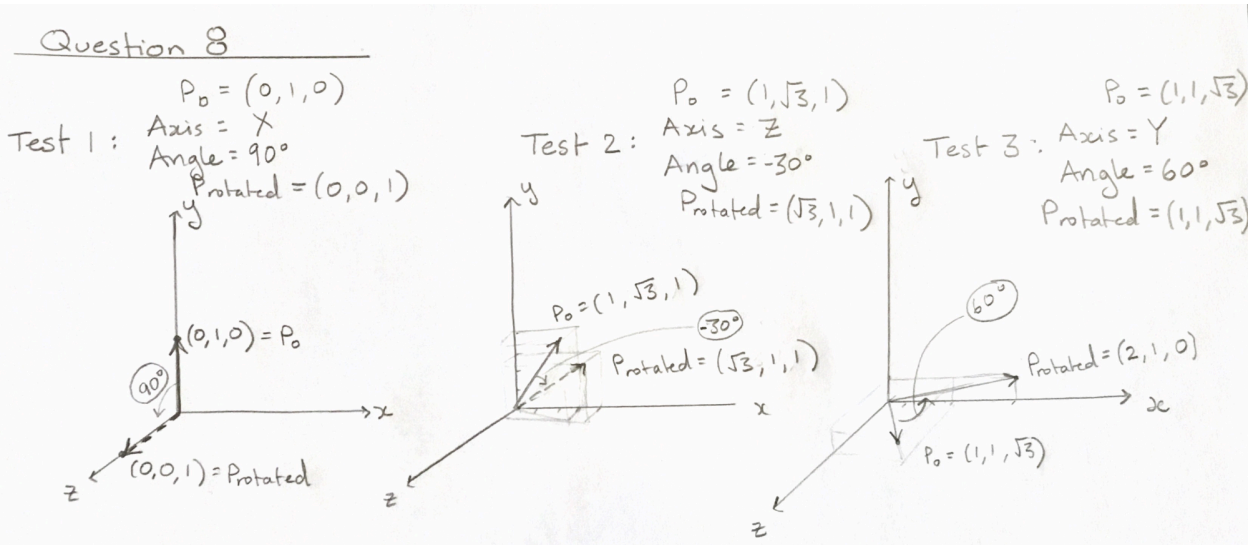
$$R_y = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- These can be made into the 4x4 homogeneous rotation matrices by padding the 4th column with [0,0,0,1] and the 4th row will be [0,0,0,1] as well.

Testing

- Test 1: 90-degree rotation about X-axis of a Y-direction unit vector.
- Test 2: 30-degree rotation about the Z-axis of vector [1,√3,1] (this is a 30-60-90-degree triangle). The expected output is [√3,1,1].
- Test 3: 60-degree rotation about the Y-axis of vector [1,1,√3] (this is a 30-60-90-degree triangle). The expected output is [2,1,0].



Question 9: Frame-Transformation-to-Home

```
function [T_e2h] = generateFrameTransformationToHome(Oe,e1,e2,e3)
```

Method

- The translation matrix is given by:

$$Trans_{e\ to\ h} = \begin{pmatrix} 1 & 0 & 0 & O_{ex} \\ 0 & 1 & 0 & O_{ey} \\ 0 & 0 & 1 & O_{ez} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- The rotation matrix is given by:

$$Rot_{e\ to\ h} = \begin{pmatrix} e1_x & e2_x & e3_x & 0 \\ e1_y & e2_y & e3_y & 0 \\ e1_z & e2_z & e3_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- These transformations can be used to transform a point in the orthonormal e-basis to the orthonormal h-basis (home frame):

$$P_h = Rot_{e\ to\ h} * Trans_{e\ to\ h} * P_e$$

- This can be written in the form of a single transformation matrix operation:

$$P_h = T_{e\ to\ h} * P_e$$

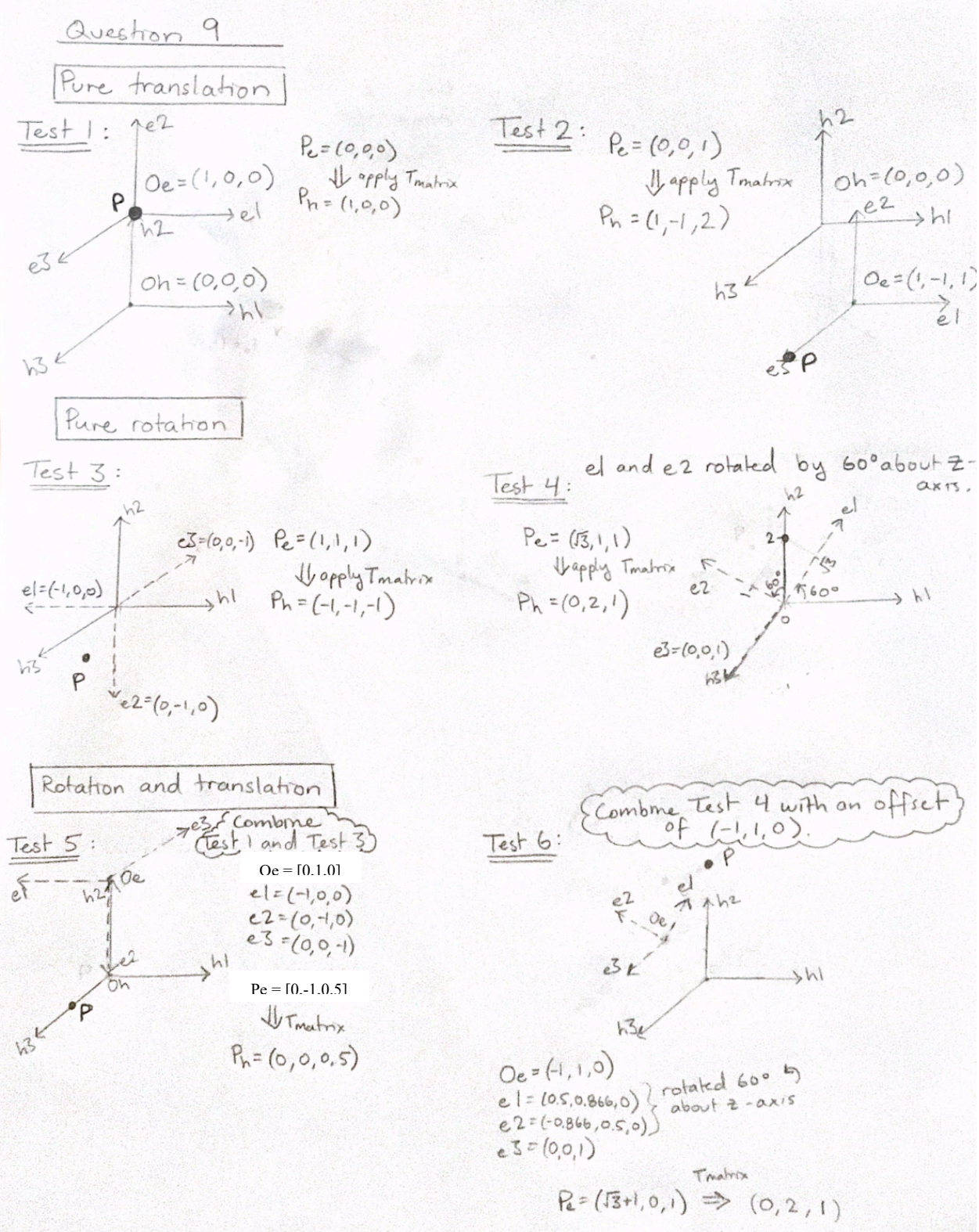
Where

$$T_{e\ to\ h} = Rot_{e\ to\ h} * Trans_{e\ to\ h}$$

Testing

In all test cases, Oh = [0,0,0], h1 = [1,0,0], h2 = [0,1,0] and h3 = [0,0,1].

- Test 1 (Pure Translation): Offset the origin of the e basis in the x-axis such that Oe = [1,0,0]. e1=h1, e2=h2 and e3=h3. For point Pe = [0,0,0] the expected output is Ph = [1,0,0].
- Test 2 (Pure Translation): Offset the origin of the e basis in x, y and z such that Oe = [1, -1,1]. e1=h1, e2=h2 and e3=h3. For point Pe = [0,0,1] the expected output is Ph = [1, -1,2].
- Test 3 (Pure Rotation): Make the basis vectors negative such that e1 = [-1,0,0], e2 = [0,-1,0] and e3 = [0,0,-1] and let Oe = Oh. For point Pe = [1,1,1] the expected output is Ph = [-1,-1,-1].
- Test 4 (Pure Rotation): Rotate e1 and e2 by 60 degrees counterclockwise (when looking in the negative z direction) about the z-axis. In this case e1 = [0.5,0.8660,0], e2 = [-0.866,0.5,0] and e3 = [0,0,1]. For point Pe = [$\sqrt{3}$,1,1] the expected output is Ph = [0,2,1].
- Test 5 (Translation and Rotation): This test case is a combination of Test 1 and Test 3. The origin of the e basis is offset in the x-axis such that Oe = [0,1,0]. The basis vectors are then made negative such that e1 = [-1,0,0], e2 = [0,-1,0] and e3 = [0,0,-1]. For point Pe = [0,-1,-0.5] the expected output is Ph = [0,0,0.5].
- Test 6 (Translation and Rotation): This test case is the same as Test 4 but with an additional offset in the x and y planes such that Oe = [-1,1,0]. e1 and e2 are again rotated by 60 degrees counterclockwise (when looking in the negative z direction) about the e3-axis. For the point Pe = [1+$\sqrt{3}$,0,1] the expected output is Ph = [0,2,1] again since Pe "undoes" the basis translation before the rotation-to-home is applied.



Question 10: Target-Tracking-Error-Simulation

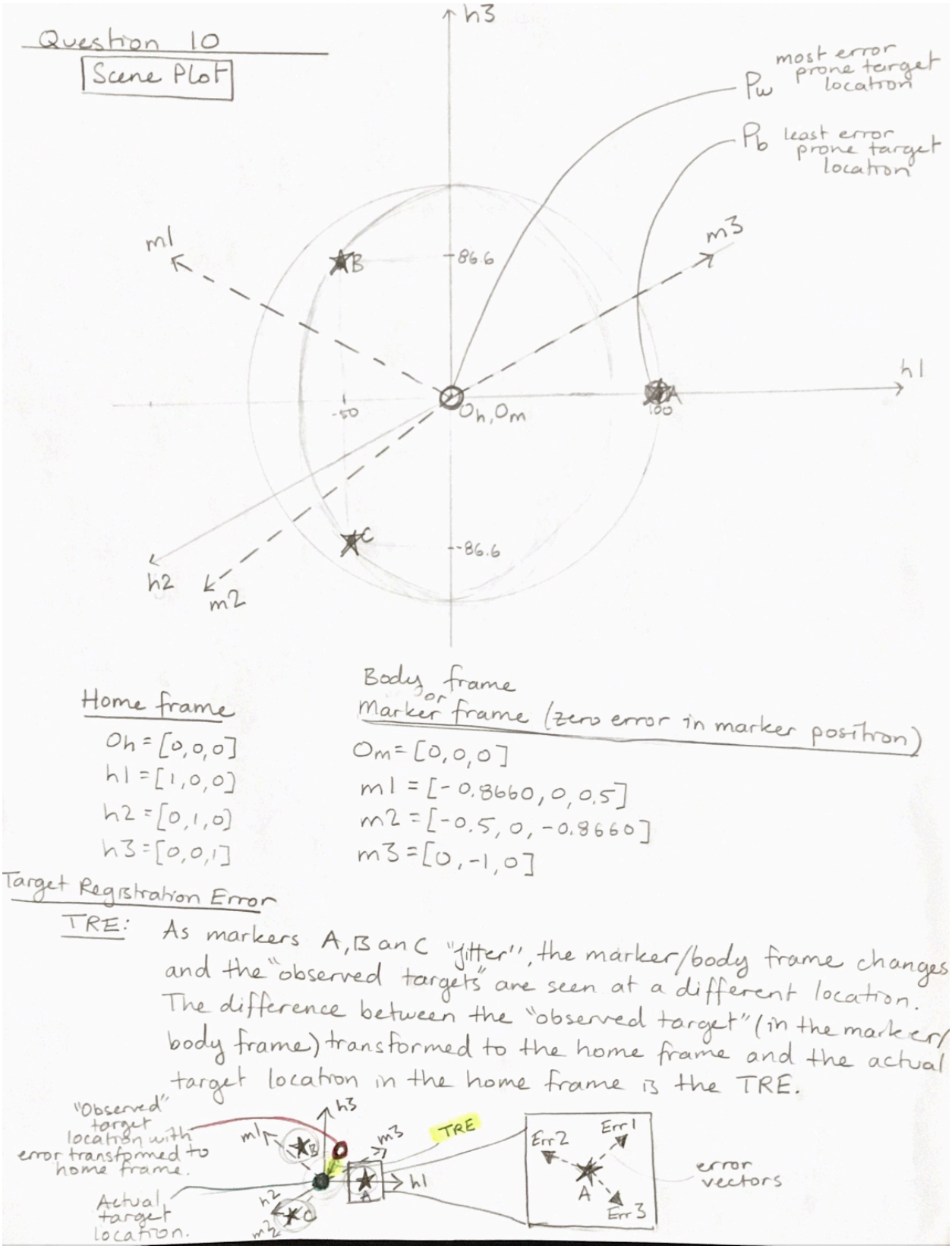
```
target_tracking_error_incrmenting % Runs the simulations that increments the
magnitude of the error vectors and exits of the maximum error is reached -
this is done for 2 target locations

target_tracking_error_analysis % Runs the simulation that increments the
magnitude of the error vectors and generates failure rate analysis for 2
target locations
```

Target Selection

The position of the markers and body relative to the target matters a lot. Based on tests on target location in the marker frame, **the worst location for the target is at the center of all 3 markers (at the origin)** because, on average, this results in the largest target registration error (TRE) and failure rate when there is error in the location of the markers. **The best location for the target is at the same location as one of the markers (A, B or C)** because this results in the smallest average TRE and failure rate when there is error in the location of the markers.

Scene Plot


Question 10
Scene Plot
h3
most error prone target location
Pw
Pb
least error prone target location
m1
m3
B
-86.6
h1
-50
Oh, Om
100
C
--86.6
h2
m2
Home frame
Oh = [0,0,0]
h1 = [1,0,0]
h2 = [0,1,0]
h3 = [0,0,1]
Body frame
or
Marker frame (zero error in marker position)
Om = [0,0,0]
m1 = [-0.8660, 0, 0.5]
m2 = [-0.5, 0, -0.8660]
m3 = [0, -1, 0]
Target Registration Error
TRE: As markers A, B and C "jitter", the marker/body frame changes and the "observed targets" are seen at a different location. The difference between the "observed target" (in the marker/body frame) transformed to the home frame and the actual target location in the home frame is the TRE.
"Observed" target location with error transformed to home frame.
Actual target location.
TRE
Err2
Err1
Err3
A
error vectors

Simulation Program

```
target_tracking_error_incrmenting % Runs the simulations that increments the magnitude of the error vectors and exits of the maximum error is reached - this is done for 2 target locations
```

The target tracking accuracy was considered a failure if the TRE reached 4.00mm as the displacement between the ground truth and the transformed target positions. A simulation was created that incremented through the error magnitude from 0 to 10mm in steps of 0.05mm. If the TRE reaches or excedes 4.00mm during the incrementation, the TRE and error magnitude are reported and the simulation stops. This simulation is conducted with the true target position at [0,0,0] and [100,0,0] respectively.

Analysis

```
target_tracking_error_analysis % Runs the simulation that increments the magnitude of the error vectors and generates failure rate analysis for 2 target locations
```

The failure rate as a function of the error magnitude was generated while incrementing the error magnitude from 0 to 10mm in steps of 0.05mm. Each error magnitude value was tested 5 times. This was conducted with the true target position at [0,0,0] and [100,0,0] respectively. The results of displayed in Figure 1 and Figure 2 below.

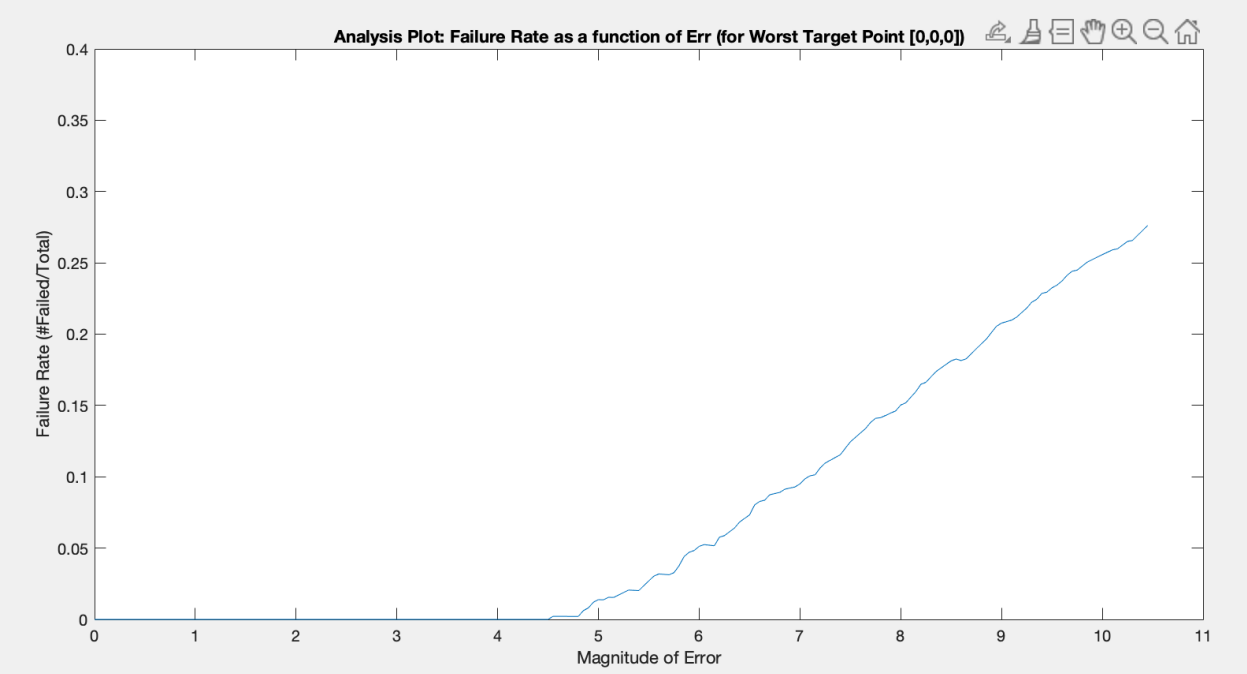


Figure 1 - Failure Rate as a function of the marker error magnitude for target location [0,0,0].

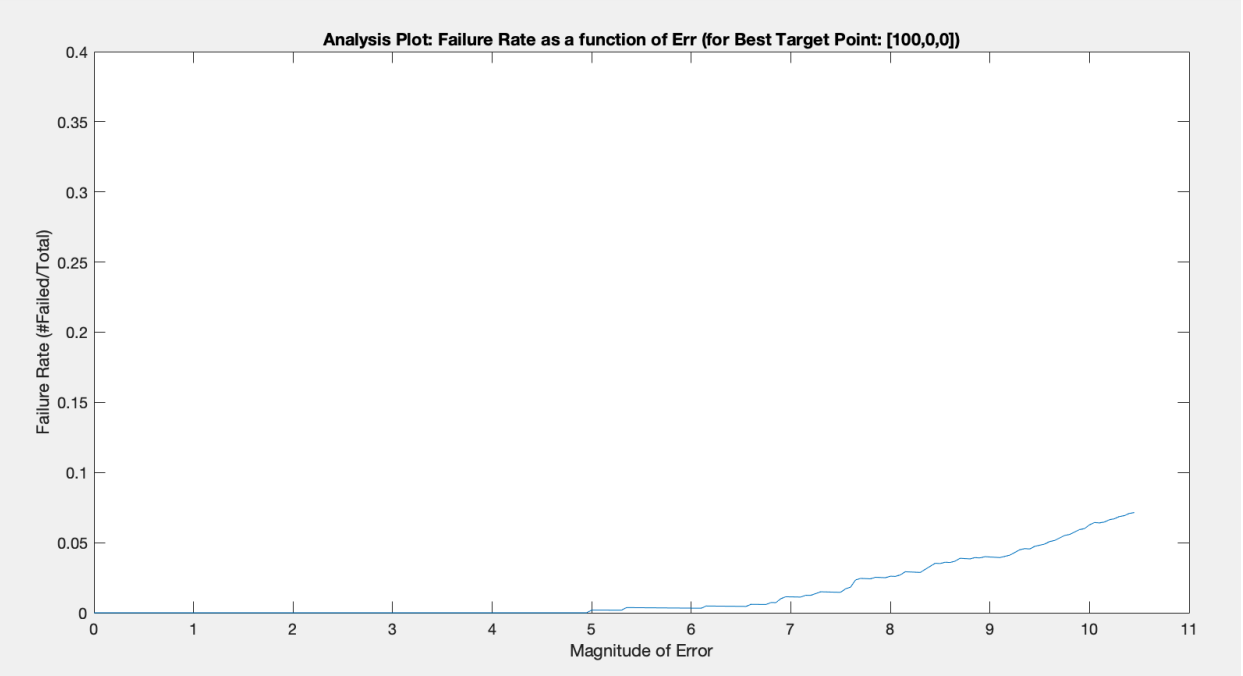


Figure 2 - Failure Rate as a function of the marker error magnitude for target location [100,0,0].