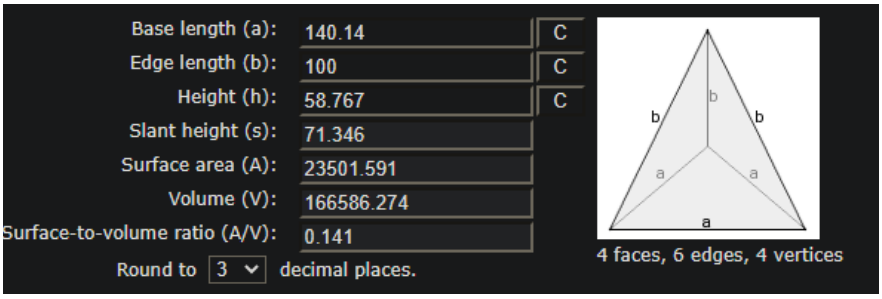
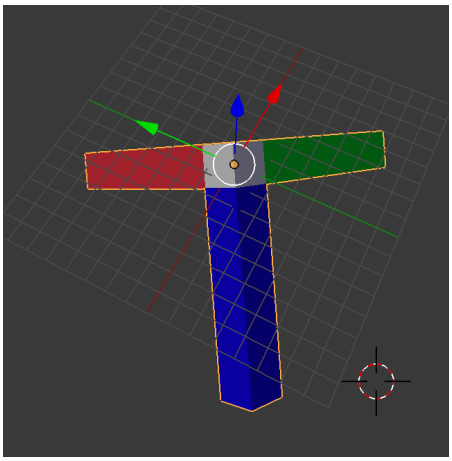
Yaneev Hacohen - ygh2 and Krashagi Gupta - kxg360
Robotics I

Problem 1

- Assuming frame2 forms a pyramid, its height can be approximated below, where $b = 100mm$ and $a = 2(100/\sqrt{2}) = 141mm$, since any two straws are perpendicular and form a 45-45-90 triangle



- Therefore the z-coordinate of frame2 wrt frame1 is appx 59mm based off the calculation above
- By inspection, $x = 60mm, y = 80mm$
- Finally, $o_{2/1} = [60; 80; 59]mm$
- Using a dummy set of axes in a 3D modelling program to check ourselves, we can say that frame2 is approximately rotated as below, with the thin axes shown representing frame1, and the thick axes representing frame2
- The rotation of the axes shown is approximately the same as the one in the problem set, without the origin offset
  - 225d about the X-axis (frame1 r)
  - 40d about the Y-axis (frame1 g)
  - 90d about the Z-axis (frame1 b)
  - In this specific order!



- Having matlab generate the final rotation matrix is done as follows

```
X = rotx(deg2rad(235));
Y = roty(deg2rad(40));
Z = rotz(deg2rad(90));
R=Z*Y*X
```

- With final variable $R_{2/1}$ as

```
R =

         0    0.5736   -0.8192
    0.7660   -0.5265   -0.3687
   -0.6428   -0.6275   -0.4394
```

- Checking that this is a valid rotation matrix can be done as follows, with dot product of each column = 0, magnitude of the cross product of each column = 0, and det(R) = 1

```
%check valid rotaiton matrix
dots = dot(R(:,1),R(:,2))+dot(R(:,2),R(:,3))+dot(R(:,3),R(:,1));
crosses = norm(cross(R(:,1),R(:,2)))+norm(cross(R(:,2),R(:,3)))+norm(cross(R(:,3),R(:,1)));
detR = det(R);
```

- As expected, this is a valid rotation matrix (summing each cross product is 1+1+1=3)

crosses	3
detR	1.0000
dots	-5.5511e-17

- To create $R_{1/2}$ and $o_{1/2}$, all that is necessary is the inverse of A, where A is calculated and equal to as follows

```
o = [60; 80; 59];
A = [R o];
A = [A; 0 0 0 1]
```

```
A =

         0    0.5736   -0.8192   60.0000
    0.7660   -0.5265   -0.3687   80.0000
   -0.6428   -0.6275   -0.4394   59.0000
         0         0         0    1.0000
```

- Taking the inverse yields

```
Ainv =

    0.0000    0.7660   -0.6428  -23.3591
    0.5736   -0.5265   -0.6275   44.7316
   -0.8192   -0.3687   -0.4394  104.5679
         0         0         0    1.0000
```

- With $R_{1/2}$ as:

```
 0.0000    0.7660   -0.6428
 0.5736   -0.5265   -0.6275
-0.8192   -0.3687   -0.4394
```

and $o_{1/2}$ as:

```
-23.3591
 44.7316
104.5679
```

Krashagi did this on paper and made measurements by hands, and found the origin and Rotation matrix, however it did not meet the expectations of each column being unity magnitude. Her effort is attached as a pdf at the end of the document.

Problem 2

There are two methods of solving this problem, method1 is as follows

- First import the points

```
% import vectors
vec1 = csvread('p1vecs.csv');
vec2 = csvread('p2vecs.csv');
```

- Let A be the transformation matrix
- We know that A**vec1* = *vec2*
- Therefore *A* = *vec2*vec1*$^{-1}$
  - Where *vec1*$^{-1}$ is the pseudo inverse of *vec1*
- Having MATLAB solve this for us is easy

```
%% Method 1 - using pseudo - inverse
vec1_a = vec1;
vec2_a = vec2;
% change to 4x1 to work with translation matrix
vec1_a(4,:) = 1;
vec2_a(4,:) = 1;
% calculate pseudo-inverse
piVec1 = pinv(vec1_a);
%calculate transformation matrix
A_m1 = vec2_a*piVec1;
```

- Solving for *A_m1* yields

0.5162	-0.8039	0.2955	1.1000
0.8150	0.3549	-0.4580	2.2000
0.2633	0.4773	0.8384	3.3000
4.4409e-16	6.6613e-16	-2.2204e-16	1

- Giving *R* as:

0.5162	-0.8039	0.2955
0.8150	0.3549	-0.4580
0.2633	0.4773	0.8384

and *o* as:

1.1000
2.2000
3.3000

The second method of solving this problem utilizes singular value decomposition as follows

- Using the algorithm from the link sent out:

```
%% Method 2
%compute average vectors
vec1_av = mean(vec1,2);
vec2_av = mean(vec2,2);
%compute center vectors (aka distances from average)
vec1_diffs = vec1 - vec1_av;
vec2_diffs = vec2 - vec2_av;
%calculate covariance matrix
H = vec1_diffs*vec2_diffs';
%calculate SVD
[U,~,V] = svd(H);
R = V*U';   %final rotation matrix
%find origin pts.
t = -R*vec1_av + vec2_av;
%final transformation matrix
A_m2 = [R t; 0 0 0 1];
```

- Solving for A_m2 yields:

0.5162	-0.8039	0.2955	1.1000
0.8150	0.3549	-0.4580	2.2000
0.2633	0.4773	0.8384	3.3000
0	0	0	1

- Giving *R* as:

0.5162	-0.8039	0.2955
0.8150	0.3549	-0.4580
0.2633	0.4773	0.8384

and *o* as:

1.1000
2.2000
3.3000

Comparing A from method1 and method2

- A comparison can be done by applying the transformation matrix on vec1 and seeing the difference between that and vec2

```
%% Check work
m1_diffs = norm(vec2_a - A_m1*vec1_a);
m2_diffs = norm(vec2_a - A_m2*vec1_a);
```

- Producing the following error approximations, as we can see both methods are effective, with the second being *slightly* better

m1_diffs	1.3088e-14
m2_diffs	2.6041e-15

Challenge Question

- The challenge question can be computed using method2 above utilizing the SVD
- First, we import
- Assumption: from the PSET that the camera's "approximately pointing straight-down" means that it *is* pointing straight down at a height of *z=3*

```
%% Import options
%import
lidarPts = csvread('image_key_pts_metric.csv');
cameraPixels = csvread('image_key_pts_pixels.csv');
%transpose to get 20 column vectors
lidarPts = lidarPts';
cameraPixels = cameraPixels';
%calculate camera pts
cameraPts = nan(size(cameraPixels));
z=3;
cameraPts(1,:) = ((cameraPixels(1,:)-1850)/1638)*z;
cameraPts(2,:) = ((cameraPixels(2,:)-1000)/1638)*z;
cameraPts(3,:) = 3;
%set z-coordinate of both
lidarPts(3,:) = 0;  %from PSET
```

- Then we use use the method as mentioned above

```
%% Solve using SVD
%compute average vectors
camera_av = mean(cameraPts,2);
lidar_av = mean(lidarPts,2);
%compute center vectors (aka distances from average)
camera_diffs = cameraPts - camera_av;
lidar_diffs = lidarPts - lidar_av;
%calculate covariance matrix
H = camera_diffs*lidar_diffs';
%calculate SVD
[U,~,V] = svd(H);
V(:,3)=V(:,3).*-1;
R = V*U';   %final rotation matrix
%find origin pts.
t = -R*camera_av + lidar_av;
%final transformation matrix
A = [R t; 0 0 0 1];
```

- We can now inspect *R* and *t* and plug them into *challenge_Q_metric.m* to get an RMS of appx 28 which is better than the 100 we were given as a starting point

0.0616	0.9981	0
0.9981	-0.0616	0
0	0	1

1.6590
0.1350
3

```
rms_err =

   28.2641
```

- Due to the assumption mentioned above (and that the camera frame is not exactly where the camera), this prevents *rms_err* from truly being minimized, nonetheless, I believe this is a thoughtful attempt

Name : kx

Id :

mcdm

Q1. $100\,mm = 100 \times \frac{1}{1000}\,m = 0.1\,m$

find :- Origin of frame 2 w.r.t frame 1 $\quad \vec{o}_{2/1} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{3\times1}$

orientation of frame 2 w.r.t frame 1

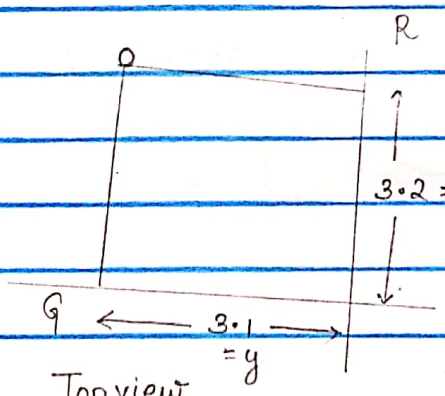
$$R_{2/1} = \begin{bmatrix} \vec{r}_{21} & \vec{q}_{21} & \vec{b}_{21} \end{bmatrix}_{3\times3}$$

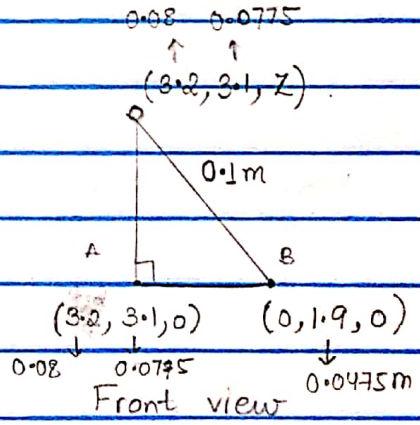
Make sure, $|r_{21}| = |q_{21}| = |b_{21}| \approx 1$

$r_{21} \times q_{21} = b_{21}$

Also find :- $R_{1/2}$ & $o_{1/2}$

Sol.





Stem length = 4cm ⇒ 0.1m

1cm ⇒ $\frac{0.1}{4}$ m

$x = 3.2 \times 0.1/4 = 0.08\,m$

$y = 3.1 \times 0.1/4 = 0.0775\,m$

1.9 ⇒ $1.9 \times 0.1/4 = 0.0475\,m = 0.0375$

using $OA^2 + AB^2 = OB^2$

$OA^2 = z^2$

$AB^2 = (0.08-0)^2 + (0.0775 - 0.0975)^2 + (0-0)^2$ = ~~7.3m~~ 8m

$OB^2 = 0.1^2$

$\Rightarrow z^2 + 7.3m = 0.1^2$

$z = 0.052m$

0.044 meter

$$O_{2/1} = \begin{bmatrix} 0.08 \\ 0.0775 \\ 0.044 \end{bmatrix}$$

$0.1/4$ My Projections could be wrong

$$\vec{r}_{21} = \begin{bmatrix} -0.7 \\ +2.0 \\ -0.044 \end{bmatrix} \text{ Scaled } \Rightarrow \begin{bmatrix} -0.0175 \\ 0.05 \\ -0.044 \end{bmatrix}$$

check $\vec{r} \times \vec{g} = \vec{b}$

$$\vec{g}_{21} = \begin{bmatrix} 1.3 \\ -1.9 \\ -0.044 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.0325 \\ -0.0475 \\ -0.044 \end{bmatrix}$$

$$\vec{b}_{21} = \begin{bmatrix} -3.2 \\ -1.9 \\ -0.044 \end{bmatrix} \Rightarrow \begin{bmatrix} -0.08 \\ -0.0475 \\ -0.044 \end{bmatrix}$$