Homework 1 - ECE 4560

Brett Erickson

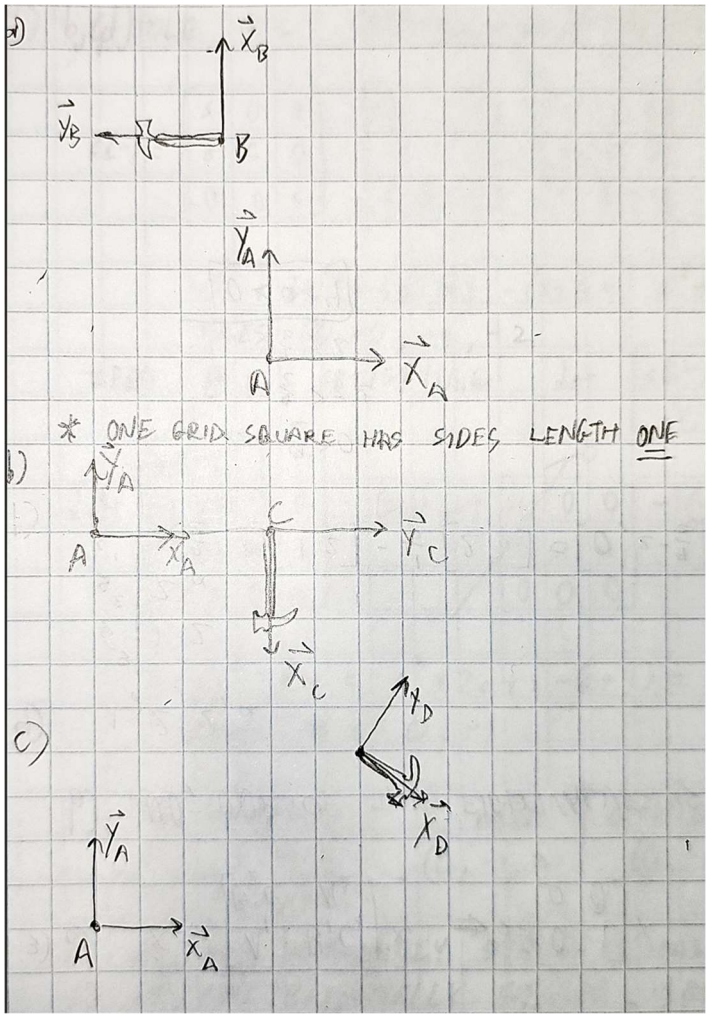
Source code can be found at

https://github.com/berickon3/ECE-4560-Robotics-Code/

```
% Import own library
import robotics.*
```

1) 3 hours of work, primarily because I wanted to make a package to handle transforms.

2)


a)
$\vec{X}_B$
$\vec{Y}_B$
B
$\vec{Y}_A$
A
$\vec{X}_A$
* ONE GRID SQUARE HAS SIDES LENGTH ONE
b)
$\vec{Y}_A$
A
$\vec{X}_A$
C
$\vec{Y}_C$
$\vec{X}_C$
c)
$\vec{Y}_D$
$\vec{X}_D$
$\vec{Y}_A$
A
$\vec{X}_A$

3)

The function planarTranslation2d I wrote takes arguments containing the point in xy, displacment in xy, and rotation as theta in radians. It is shown in **Appendix B**

The rotation matrix is:

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

```
point = [2;0];
theta = pi;
displacement = [-1;5];

planarTranslation2d(point,displacement,theta)
```

```
ans = 2×1
    -3
     5
```

```
theta = -pi/2;
displacement = [4;0];

planarTranslation2d(point,displacement,theta)
```

```
ans = 2×1
     4
    -2
```

```
theta = -pi/6;
displacement = [6;4];

planarTranslation2d(point,displacement,theta)
```

```
ans = 2×1
    7.7321
    3.0000
```

4)

Following problem 3, I realized it would be easier to treat each transform as an object and define operations that can be done with them as functions. The transform2d.m file defines the class and gives functions for instantiation, point transformation, and composition of multiple transforms. It is given in **Appendix A**

```
world_transform = transform2d([2;4], -pi/2);
B_transform = transform2d([-1;5],pi);
C_transform = transform2d([4;0],-pi/2);
D_transform = transform2d([6;4],-pi/6);

world_B_transform = world_transform.compose(B_transform);
world_C_transform = world_transform.compose(C_transform);
world_D_transform = world_transform.compose(D_transform);

world_B_transform.pointOperation(point)
```

```
ans = 2×1
     9
    11
```

```
world_C_transform.pointOperation(point)
```

```
ans = 2×1
    2.0000
    4.0000
```

```
world_D_transform.pointOperation(point)
```

```
ans = 2×1
    7.0000
    0.2679
```

5)

a) displacement = $\begin{bmatrix} -6 \\ 2 \end{bmatrix}$ theta = -$\frac{\pi}{2}$

b) displacement = $\begin{bmatrix} 2 \\ 6 \end{bmatrix}$ theta = $\frac{\pi}{2}$

c)

```
AB_transform = transform2d([-6; 2], -pi/2);
AB_transform.rotation
```

```
ans = 2×2
     0.0000    1.0000
    -1.0000    0.0000
```

$$g_{AB} = \left(\begin{bmatrix} -6 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \right)$$

6)

a) Visited and helped with partial assembly of SO-101 leader

b) I am most interested in the SO-101 platform. For one, it is open source and can be combined with the Hugging Face LeRobot machine learning library. I've got an Nvidia Jetson at home, so maybe I can leverage that. Also it is low cost, so I just bought the parts to work on it at home.

c) Completed

Appendix A

transform2d.m

```
classdef transform2d
    %TRANSFORM This class defines a transform in 2d

    properties
        displacement
        rotation
        points
        theta
    end

    methods
        function obj = transform2d(displacement,theta)
            obj.displacement = displacement;
            obj.theta = theta;
            obj = obj.updateRotMat();
            obj.points = NaN([2,1]);
        end

        function obj = compose(obj,transform2)
            obj.updateRotMat();
            transform2.updateRotMat();
            overallDisplacement = obj.displacement + obj.pointOperation(transform2.displacement);
            overallRotation = obj.rotation * transform2.rotation;
            overallTheta = atan2(overallRotation(2,1),overallRotation(1,1));
            obj = robotics.transform2d(overallDisplacement,overallTheta);
        end

        function points = pointOperation(obj,points)
            obj.updateRotMat();
            points = obj.displacement + obj.rotation*points;
        end

        function obj = updateRotMat(obj)
            obj.rotation = [cos(obj.theta), -sin(obj.theta); sin(obj.theta), cos(obj.theta)];
        end
    end
end
```

Appendix B

planarTranslation2d.m

```
function point = planarTranslation2d(point,displacement,theta)
    point = displacement + [cos(theta), -sin(theta); sin(theta), cos(theta)]*point;
end
```