Midterm- Corrections

Renee Catanach G20220457

GitHub link: https://github.com/rcatanach17/Midterm

Robot Arm System Code Explained

```
# Renee Catanach
# G20220457

import numpy as np #import package - array management, math functions
import cv2 # we can display images to the screen
```

First the code begins with the implementation of the packages

Those packages include cv2, which is used for visual media programming problems, cv2.imshow() method is used to display an image to the window

Numpy is a good library that enables for mathematical problems to be solved easy and efficiently..

```
def makeTmat(a,b):
    T = np.eye(3,3)
    T[0,2] = a
    T[1,2] = b
    return T

def makeRmat(deg):
    rad = deg2rad(deg)
    c = np.cos(rad)
    s = np.sin(rad)
    R = np.eye(3,3)
    R[0,0] = c
    R[0,1] = -s
    R[1,0] = s
    R[1,1] = c
    return R
```

This is how we manipulate the coordinate systems. MakeRmat transforms the coordinates degrees to enable easy manipulation that will be used later in functions such as to rotate the polygons in which we are creating.

```
def getLine(x0,y0,x1,y1):
    points=[]
    if abs(x1-x0)>=abs(y1-y0):
        if x0>=x1:
            for x in range(x0,x1-1,-1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x,int(y)))
        else:
            for x in range(x0,x1+1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x,int(y)))
    else:
        if y0>=y1:
            for y in range(y0,y1-1,-1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x),y))
        else:
            for y in range(y0,y1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x),y))
    return points
```

This function is used to make a line. This operation is done by manipulating different points that is determined by their according coordinates. Then the points are stored in an array and then outputted together, forming the line.

```
def drawLine(canvas, x0, y0, x1, y1, color=(255, 255, 255)):

    xys = getLine(x0, y0, x1, y1)
    for xy in xys:
        x, y = xy
        canvas[y, x, :] = color

    return
```

draw line function is where we implement the get line function and this is what is outputted to the screen/canvas.

```
def deg2rad(deg):
    rad = deg * np.pi / 180.
    return rad
```

Function that converts degrees to radians.

```
def getRect(h,w):
    points = []
    points.append( (0, 0, 1))
    points.append( (0, w, 1))
    points.append( (h, w, 1))
    points.append( (h, 0, 1))
    points = np.array(points)
    return points
```

Get rectangle is what is used to create the robots arms. We manipulate the height and width, and the coordinates to output the rectangle.

```
def drawLinePQ(canvas, p, q, color):
    drawLine(canvas, p[0], p[1], q[0], q[1], color)
    return

def drawPolygon(canvas, pts, color, axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k,0], pts[k,1],
                         pts[k+1,0], pts[k+1,1], color)
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True: # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color=(255, 128, 128))
    #
    return
```

drawLinePQ is a function created to easily return the line printed to the canvas.

drawPolygon determined by number of vertices provided will output the polygon desired.

```
def erasePolygon(canvas, pts, color=(0,0,0), axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k,0], pts[k,1],
                         pts[k+1,0], pts[k+1,1], color)
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True: # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color)
    return
```

Function inspired by classmates, we will erase polygon created to get the desired shape. Color = (0,0,0) will give us black, which will then blend in with background, giving the effect that we “erased the polygon”

```
def rotatePoints(degree, points):
    R = makeRmat(30) # define how many degrees you would like to rotate
    qT = R @ points.T
    points = qT.T
    return points
```

Here is where we rotate the polygon determined by the number of degrees, within makeRmat. The polygon will then be manipulated to the output.

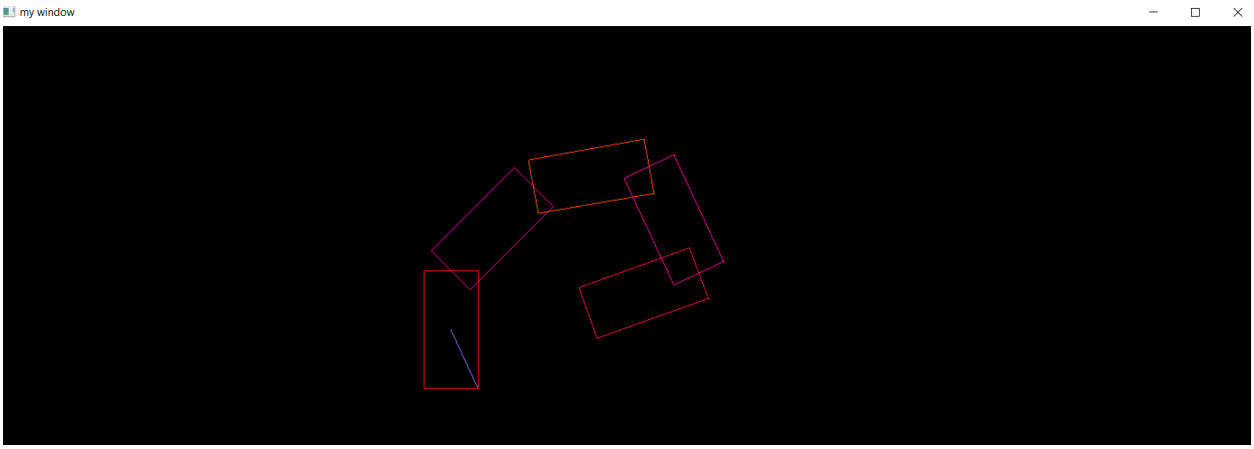
```
def main():
    width, height = 1400, 1000
    canvas = np.zeros( (height, width, 3), dtype='uint8')
    alpha = 5
    beta = 10
    teta = 5

    while True:
        # 2. regular n-gon
        color = np.random.randint(0, 256, size=3)
        ngon = 4 # np.random.randint(3, 13)
        h = 60
        w = 130
        rect1 = getRect(h, w) # vertices of the N-gon
        rect2 = rect1.copy()  # hard copy
        rect3 = rect1.copy()  # hard copy
```

```
        rect4 = rect1.copy()  # hard copy

        # Rect 1
        Tc = makeTmat(600,400)
        R1 = makeRmat(-180)
        T1 = makeTmat(w/2,0)
        H = Tc @ R1 @ T1
        rect1 = (H @ rect1.T).T
        rect1 = rect1.astype('int')
        drawPolygon(canvas, rect1, (0, 0, 255), axis=True)

        # Rect 2
        T2 = makeTmat(0,w)
        T3 = makeTmat(h/2,0)
        T4 = makeTmat(-h/2,0)
        R2 = makeRmat(45)
        H2 = H @ T2 @ T3 @ R2 @ T4
        rect2 = (H2 @ rect2.T).T
        rect2 = rect2.astype('int')
        drawPolygon(canvas, rect2, (160, 0, 255))

        # Rect 3
        R2 = makeRmat(35)
        H3 = H2 @ T2 @ T3 @ R2 @ T4
        rect3 = (H3 @ rect3.T).T
        rect3 = rect3.astype('int')
        drawPolygon(canvas, rect3, (10, 80, 255))

        # Rect 4
        R2 = makeRmat(75)
        H4 = H3 @ T2 @ T3 @ R2 @ T4
        rect4 = (H4 @ rect4.T).T
        rect4 = rect4.astype('int')
        drawPolygon(canvas, rect4, (160, 10, 255))

        # Rect 5
        R2 = makeRmat(95)
        H5 = H4 @ T2 @ T3 @ R2 @ T4
        rect5 = (H5 @ rect5.T).T
        rect5 = rect5.astype('int')
        drawPolygon(canvas, rect5, (80, 15, 255))

        cv2.imshow("my window", canvas)
        if cv2.waitKey(20) == 27: break

        erasePolygon(canvas, rect1, axis=True)

        alpha += 3
        beta += 3
        teta += -2
    #
#
```

```
if __name__ == "__main__": # __
    main()
```

Here is the main part of the code where the rectangles are being drawn. By making hard copies of one initial rectangle and then manipulating its coordinates by shifting and rotating we can get the arm effect. I drew four rectangles as stated in the assignment.

Output of results



Stars Twinkling in Night Sky Code Explained

```
import numpy as np
import cv2
import time

def makeT(a, b):
    T = np.eye(3, 3)
    T[0, 2] = a
    T[1, 2] = b
    return T


def makeR(deg):
    rad = deg2rad(deg)
    c = np.cos(rad)
    s = np.sin(rad)
    R = np.eye(3, 3)
    R[0, 0] = c
```

```
    R[0, 1] = -s
    R[1, 0] = s
    R[1, 1] = c
    return R


def getLine(x0, y0, x1, y1):
    points = []
    if abs(x1-x0) >= abs(y1-y0):
        if x0 >= x1:
            for x in range(x0, x1-1, -1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x, int(y)))
        else:
            for x in range(x0, x1+1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x, int(y)))
    else:
        if y0 >= y1:
            for y in range(y0, y1-1, -1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x), y))
        else:
            for y in range(y0, y1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x), y))
    return points


def drawLine(canvas, x0, y0, x1, y1, color=(255, 255, 255)):
    if True:
        xys = getLine(x0, y0, x1, y1)
        for xy in xys:
            x, y = xy
            canvas[y, x, :] = color


#


def deg2rad(deg):
    rad = deg * np.pi / 180.
    return rad


def getRegularNGon(ngon):
    delta = 360. / ngon
    points = []
    for i in range(ngon):
        degree = i * delta
        radian = deg2rad(degree)
        x = np.cos(radian)
        y = np.sin(radian)
        points.append((x, y, 1))
```

```
    #
    points = np.array(points)
    return points


def drawLinePQ(canvas, p, q, color):
    drawLine(canvas, p[0], p[1], q[0], q[1], color)
    return


def drawPolygon(canvas, pts, color, axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k, 0], pts[k, 1],
                 pts[k+1, 0], pts[k+1, 1], color)
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True:  # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color=(255, 128, 128))
    #
    return
```

This code begins similar to that of the robot arm code. I have initialized a number of functions that will be utilized. One connects coordinate points and forms a solid line by utilizing manipulation.

Then the MakeRmat transforms the coordinates degrees to enable easy manipulation that will be used later in functions such as to rotate the polygons in which we are creating.

```
def drawStar(canvas, pts, color, axis=False):
    drawLine(canvas, pts[0, 0], pts[0, 1], pts[2, 0], pts[2, 1], color)
    drawLine(canvas, pts[0, 0], pts[0, 1], pts[3, 0], pts[3, 1], color)
    drawLine(canvas, pts[1, 0], pts[1, 1], pts[3, 0], pts[3, 1], color)
    drawLine(canvas, pts[1, 0], pts[1, 1], pts[4, 0], pts[4, 1], color)
    drawLine(canvas, pts[2, 0], pts[2, 1], pts[4, 0], pts[4, 1], color)
    return


def erasePolygon(canvas, pts, color=(0, 0, 0), axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k, 0], pts[k, 1],
                 pts[k+1, 0], pts[k+1, 1], color)
```

```
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True:  # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color)
    return


def eraseStar(canvas, pts, color=(0, 0, 0), axis=False):
    drawLine(canvas, pts[0, 0], pts[0, 1], pts[2, 0], pts[2, 1], color)
    drawLine(canvas, pts[0, 0], pts[0, 1], pts[3, 0], pts[3, 1], color)
    drawLine(canvas, pts[1, 0], pts[1, 1], pts[3, 0], pts[3, 1], color)
    drawLine(canvas, pts[1, 0], pts[1, 1], pts[4, 0], pts[4, 1], color)
    drawLine(canvas, pts[2, 0], pts[2, 1], pts[4, 0], pts[4, 1], color)
    return


def rotatePoints(degree, points):
    R = makeR(30)
    qT = R @ points.T
    points = qT.T
    return points
```

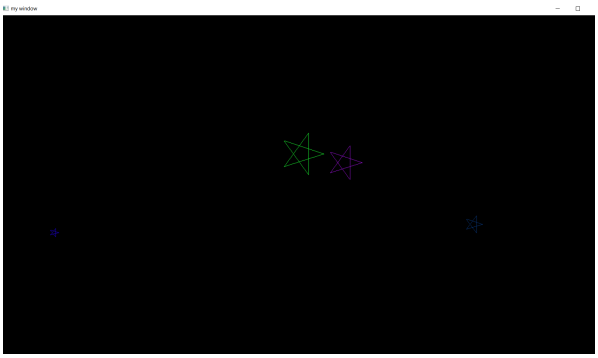
The make star function forms line within the polygon, then by calling the eraseStar function certain lines will be erased to create the star shape. Then the polygon will be erased as well, to ensure the outline is gone.

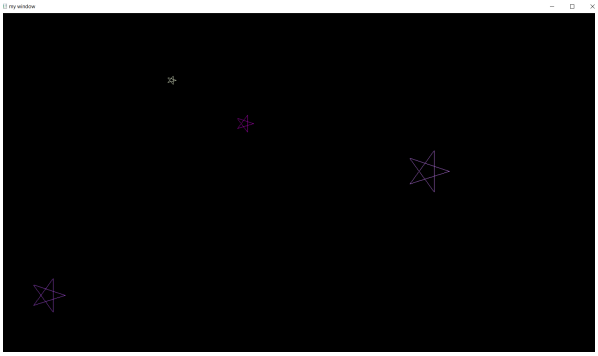
```


def main():
    width, height = 1400, 1000
    canvas = np.zeros((height, width, 3), dtype='uint8')

    while True:
        # 2. regular n-gon
        star1Pts = getRegularNGon(5)  # vertices of the N-gon
        star2Pts = getRegularNGon(5)  # vertices of the N-gon
        star3Pts = getRegularNGon(5)  # vertices of the N-gon
        star4Pts = getRegularNGon(5)  # vertices of the N-gon

        # Poly
        star1Pts *= 40
        star1Pts[:, 2] /= 40

        star1x = np.random.randint(100, 1200)
        star1y = np.random.randint(100, 700)
```

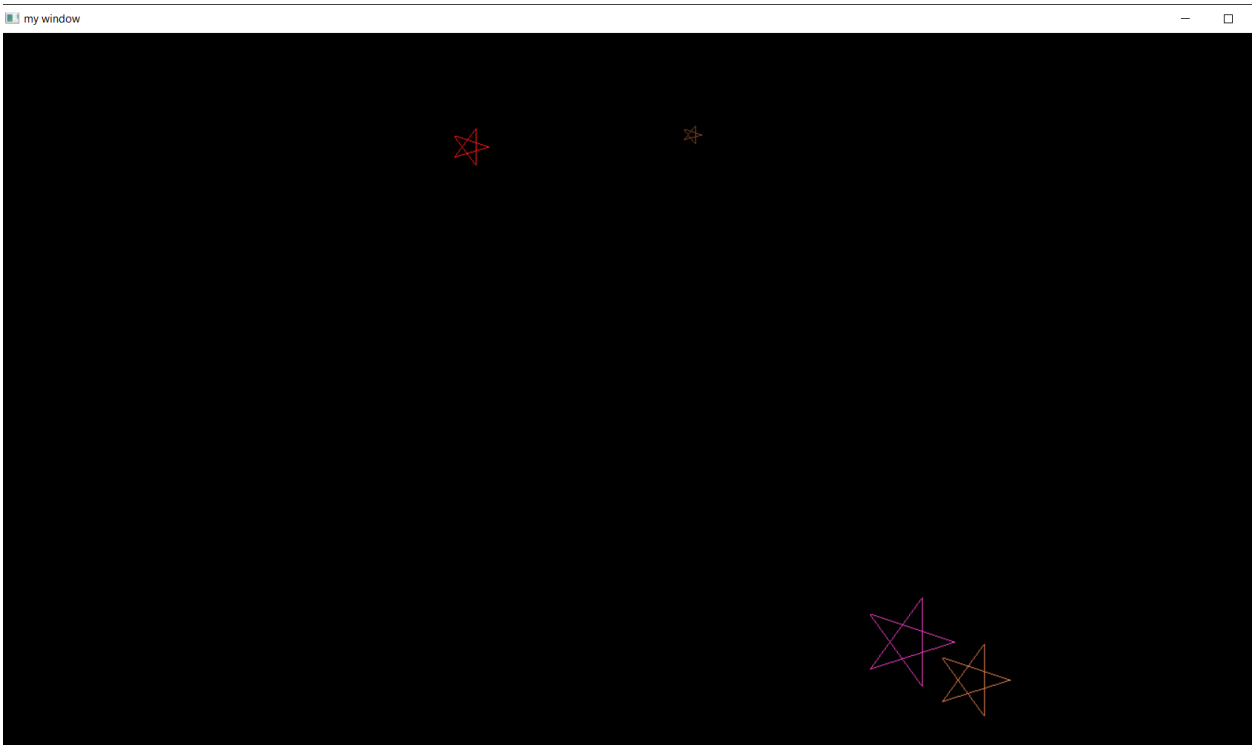
```
        Tc = makeT(star1x, star1y)
        star1Pts = (Tc @ star1Pts.T).T
        star1Pts = star1Pts.astype('int')
        drawStar(canvas, star1Pts, np.random.randint(0, 256, size=3), axis=False)

        star2Pts *= 20
        star2Pts[:, 2] /= 20

        star2x = np.random.randint(100, 1200)
        star2y = np.random.randint(100, 700)
        Tc = makeT(star2x, star2y)
        star2Pts = (Tc @ star2Pts.T).T
        star2Pts = star2Pts.astype('int')
        drawStar(canvas, star2Pts, np.random.randint(0, 256, size=3), axis=False)

        star3Pts *= 10
        star3Pts[:, 2] /= 10

        star3x = np.random.randint(100, 1200)
        star3y = np.random.randint(100, 700)
        Tc = makeT(star3x, star3y)
        star3Pts = (Tc @ star3Pts.T).T
        star3Pts = star3Pts.astype('int')
        drawStar(canvas, star3Pts, np.random.randint(0, 256, size=3), axis=False)

        star4Pts *= 50
        star4Pts[:, 2] /= 50

        star4x = np.random.randint(100, 1200)
        star4y = np.random.randint(100, 700)
        Tc = makeT(star4x, star4y)
        star4Pts = (Tc @ star4Pts.T).T
        star4Pts = star4Pts.astype('int')
        drawStar(canvas, star4Pts, np.random.randint(0, 256, size=3), axis=False)

        cv2.imshow("my window", canvas)
        if cv2.waitKey(20) == 27:
            break

        drawStar(canvas, star1Pts, (0, 0, 0), axis=False)
        drawStar(canvas, star2Pts, (0, 0, 0), axis=False)
        drawStar(canvas, star3Pts, (0, 0, 0), axis=False)
        drawStar(canvas, star4Pts, (0, 0, 0), axis=False)

        time.sleep(0.3)
    #
#


if __name__ == "__main__":  # __
    main()
```

This is the main part of the code, in which stars are called onto the screen. The stars are all positioned at different locations, and using the time function, they appear and disappear to give off the “blinking in the night” effect.

Output of results







Solar System Code Explained

```
import numpy as np
import cv2

def makeT(a,b):
    T = np.eye(3,3)
    T[0,2] = a
    T[1,2] = b
    return T

def makeR(deg):
    rad = deg2rad(deg)
    c = np.cos(rad)
    s = np.sin(rad)
    R = np.eye(3,3)
    R[0,0] = c
    R[0,1] = -s
    R[1,0] = s
    R[1,1] = c
    return R

def getLine(x0,y0,x1,y1):
    points=[]
    if abs(x1-x0)>=abs(y1-y0):
        if x0>=x1:
            for x in range(x0,x1-1,-1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x,int(y)))
        else:
            for x in range(x0,x1+1):
                y = (x-x0)*(y1-y0)/(x1-x0) + y0
                points.append((x,int(y)))
    else:
        if y0>=y1:
            for y in range(y0,y1-1,-1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x),y))
        else:
            for y in range(y0,y1):
                x = (x1-x0)*(y-y0)/(y1-y0) + x0
                points.append((int(x),y))
    return points


def drawLine(canvas, x0, y0, x1, y1, color=(255, 255, 255)):
    if True:
        xys = getLine(x0, y0, x1, y1)
        for xy in xys:
            x, y = xy
            canvas[y, x, :] = color


def deg2rad(deg):
```

```
    rad = deg * np.pi / 180.
    return rad

def getRegularNGon(ngon):
    delta = 360. / ngon
    points = []
    for i in range(ngon):
        degree = i * delta
        radian = deg2rad(degree)
        x = np.cos(radian)
        y = np.sin(radian)
        points.append( (x, y, 1) )
    #
    points = np.array(points)
    return points

def drawLinePQ(canvas, p, q, color):
    drawLine(canvas, p[0], p[1], q[0], q[1], color)
    return

def drawPolygon(canvas, pts, color, axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k,0], pts[k,1],
                         pts[k+1,0], pts[k+1,1], color)
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True: # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color=(255, 128, 128))
    #
    return

def drawStar(canvas, pts, color, axis=False):
    drawLine(canvas, pts[0,0], pts[0,1], pts[2,0], pts[2,1], color)
    drawLine(canvas, pts[0,0], pts[0,1], pts[3,0], pts[3,1], color)
    drawLine(canvas, pts[1,0], pts[1,1], pts[3,0], pts[3,1], color)
    drawLine(canvas, pts[1,0], pts[1,1], pts[4,0], pts[4,1], color)
    drawLine(canvas, pts[2,0], pts[2,1], pts[4,0], pts[4,1], color)
    return

def erasePolygon(canvas, pts, color=(0,0,0), axis=False):
    for k in range(pts.shape[0]-1):
        drawLine(canvas, pts[k,0], pts[k,1],
                         pts[k+1,0], pts[k+1,1], color)
    drawLinePQ(canvas, pts[-1], pts[0], color)

    if axis == True: # center - pts[0]
        center = np.array([0., 0, 0])
        for p in pts:
```

```
            center += p
        center = center / pts.shape[0]
        center = center.astype('int')
        print(center)
        drawLinePQ(canvas, center, pts[0], color)
    return

def eraseStar(canvas, pts, color=(0,0,0), axis=False):
    drawLine(canvas, pts[0,0], pts[0,1], pts[2,0], pts[2,1], color)
    drawLine(canvas, pts[0,0], pts[0,1], pts[3,0], pts[3,1], color)
    drawLine(canvas, pts[1,0], pts[1,1], pts[3,0], pts[3,1], color)
    drawLine(canvas, pts[1,0], pts[1,1], pts[4,0], pts[4,1], color)
    drawLine(canvas, pts[2,0], pts[2,1], pts[4,0], pts[4,1], color)
    return


def rotatePoints(degree, points):
    R = makeR(30)
    qT = R @ points.T
    points = qT.T
    return points
```

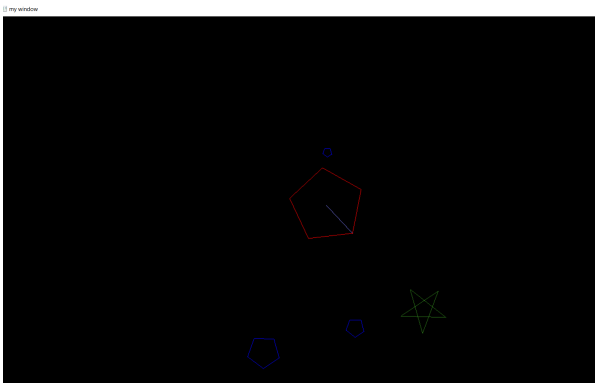
Similar to the previous two codes, the beginning part of this code is where I declare a bunch of different functions that will later play a part. The first step is creating a line, which will form the shapes of our solar system. The star is created by drawing lines within the polygon and then erasing different parts.

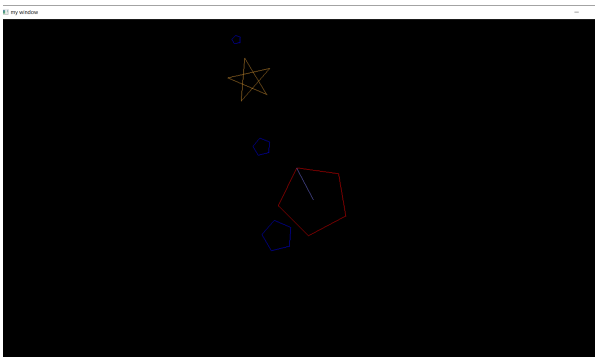
```
def main():
    width, height = 1400, 1000
    canvas = np.zeros( (height, width, 3), dtype='uint8')
    alpha = 5
    beta = 10
    teta = 5

    while True:
        # 2. regular n-gon
        color = np.random.randint(0, 256, size=3)
        ngon = 5 # np.random.randint(3, 13)
        starPts = getRegularNGon(ngon) # vertices of the N-gon
        polyPts = starPts.copy()  # hard copy
        planetPts = starPts.copy()  # hard copy
        rocketPts = starPts.copy()
        moonPts = starPts.copy()

        # Poly
        polyPts *= 80
        polyPts[:, 2] /= 80

        Tc = makeT(700,400)
```

```
        R1 = makeR(alpha)
        R2 = makeR(-alpha)
        polyPts = (Tc @ R1 @ polyPts.T).T
        polyPts = polyPts.astype('int')
        drawPolygon(canvas, polyPts, (0, 0, 255), axis=True)

        # Star
        starPts = starPts * 50  # scaling
        starPts[:,2] /= 50

        T1 = makeT(300,0)
        R3 = makeR(-beta)
        Hstar = Tc @ R1 @ T1 @ R2 @ R3
        starPts = (Hstar @ starPts.T).T
        starPts = starPts.astype('int')
        drawStar(canvas, starPts, color)
        #

        #planetM
        planetPts *= 20
        planetPts[:,2] /= 20

        Tm = makeT(150,0)
        R4 = makeR(teta)
        R5 = makeR(-teta)
        planetPts = (Hstar @ R4 @ Tm @ R5 @ planetPts.T).T
        planetPts = planetPts.astype('int')
        drawPolygon(canvas, planetPts, (255,0,0))



        rocketPts *= 10
        rocketPts[:,2] /= 20

        Tm = makeT(500,0)
        R6 = makeR(alpha)
        R7 = makeR(-alpha)
        rocketPts = (Hstar @ R6 @ Tm @ R7 @ rocketPts.T).T
        rocketPts = rocketPts.astype('int')
        drawPolygon(canvas, rocketPts, (255,0,0))


        moonPts *= 35
        moonPts[:,2] /= 35

        Tm = makeT(350,0)
        R8 = makeR(teta)
        R9 = makeR(-teta)
        moonPts = (Hstar @ R8 @ Tm @ R9 @ moonPts.T).T
        moonPts = moonPts.astype('int')
        drawPolygon(canvas, moonPts, (255,0,0))


        cv2.imshow("my window", canvas)
```

```
        if cv2.waitKey(20) == 27: break

        erasePolygon(canvas, polyPts, axis=True)
        eraseStar(canvas, starPts)
        erasePolygon(canvas, planetPts)
        erasePolygon(canvas, rocketPts)
        erasePolygon(canvas, moonPts)

        alpha += 3
        beta += 3
        teta += -2
    #
#

if __name__ == "__main__": # __
    main()
```

The main part of the code is where the code is executed. I first created different shapes, utilizing my drawline function and my erasePolygon function to get all of the shapes I needed. Then by calculating the angle I can have the shapes, depedning on initial coordinate position, I can have the shapes move. This gives the effect of rotation, amongst planets, and or objects. I created five shapes, to represent each aspect, the sun, Venus, earth, moon, and a rocket wondering around.





Me and some classmates got together to attempt the three codes, thank you.