Assignment 5
Image Registration

Problem 1: Develop a matrix equation for minimizing the cost in image registration (without shear) when the correspondences between the points in two images I1 and I2 are known. Summation indicates the sum over all points in a shape.

$$\text{Cost} = \sum (I_1 - T(I_2))^2 = \sum \left(\begin{bmatrix} x1 \\ y1 \end{bmatrix} - \left(\begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} x2 \\ y2 \end{bmatrix} + \begin{bmatrix} t1 \\ t2 \end{bmatrix} \right) \right)^2 = \sum \begin{bmatrix} x_1 - (ax_2 + by_2 + t_1) \\ y_1 - (-bx_2 + ay_2 + t_2) \end{bmatrix}^2$$

$$\text{Cost} = \sum (x_1 - (ax_2 + by_2 + t_1))^2 + (y_1 - (-bx_2 + ay_2 + t_2))^2$$

a) To find the optimal transformation that will align image 2 to image 1, take the partial derivatives of the above cost with respect to a, b, t1 and t2 and set these to 0. Express the four resulting equations in matrix form.

$\partial C/\partial a = 0$

$\partial C/\partial b = 0$

$\partial C/\partial t_1 = 0$

$\partial C/\partial t_2 = 0$

For example, the first equation will appear as:

$$\partial C/\partial a = -2x_2 (x_1 - ax_2 - by_2 - t_1) - 2y_2(y_1 + bx_2 - ay_2 - t_2) = 0$$

$$-2 x_1x_2 + 2ax_2^2 + 2bx_2y_2 + 2x_2t_1 - 2y_2y_1 - 2bx_2y_2 + 2ay_2^2 + 2t_2y_2 = 0$$

$$(2x_2^2 + 2y_2^2)a + 0\, b + 2x_2t_1 + 2y_2t_2 = 2x_1x_2 + 2y_1y_2$$

In matrix form, it can be written as

$$\sum \begin{bmatrix} 2x_2^2 + 2y_2^2 & 0 & 2x_2 & 2y_2 \\ & & & \end{bmatrix} \begin{bmatrix} a \\ b \\ t_1 \\ t_2 \end{bmatrix} = \sum \begin{bmatrix} 2x_1x_2 + 2y_1y_2 \\ \end{bmatrix}$$

Complete the above matrix equation for the four partial derives. The above matrix equation can be expressed as:

$$At = b$$

where t is the unknown transformation vector consisting of a, b, t1, and t2. It can be easily solved as:

$$t = A^{-1}b$$

Problem #2: Program the Image registration for two shapes where the correspondences are known in terms of a few points. The two shapes consists of the following points:

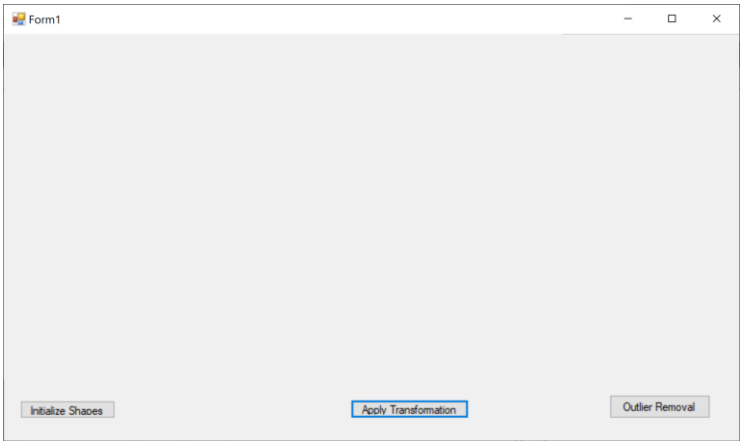
Shape 1:

```
self.shape1.append(QPoint(20,30))
self.shape1.append(QPoint(120,50))
self.shape1.append(QPoint(160,80))
self.shape1.append(QPoint(180,300))
self.shape1.append(QPoint(100,220))
self.shape1.append(QPoint(50,280))
self.shape1.append(QPoint(20,140))
```

Shape 2 points are obtained by applying the transformation to shape 1 as:

```
# transform shape1 to shape2
transf = Transformation()
transf.A = 1.05
transf.B = 0.05
transf.T1 = 15  # shift in X
transf.T2 = 22  # shift in Y
self.shape2 = self.apply_transformation(transf, self.shape1)
self.shape2[2] = QPoint(self.shape2[2].x() + 10, self.shape2[2].y() + 3)
# change one point
# add outliers to both shapes
pt_outlier1 = QPoint(200, 230)
self.shape1.append(pt_outlier1)
pt_outlier2 = QPoint(270, 160)
self.shape2.append(pt_outlier2)
```

Solution: We will use the Qt Designer to create a simple UI in Python as shown below.



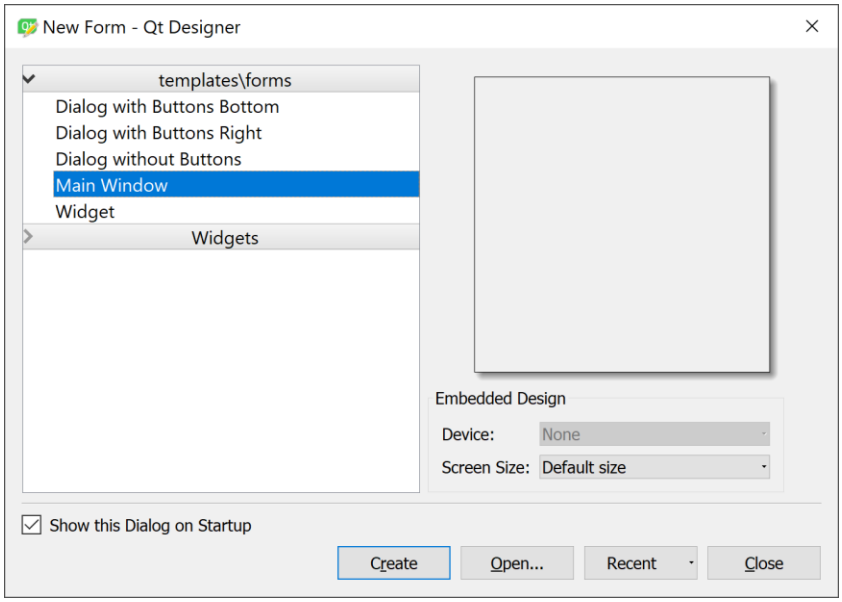
Launch command prompt as an administrator. Then issue the following command (where pytorch1x is the name of your python environment):

activate pytorch1x

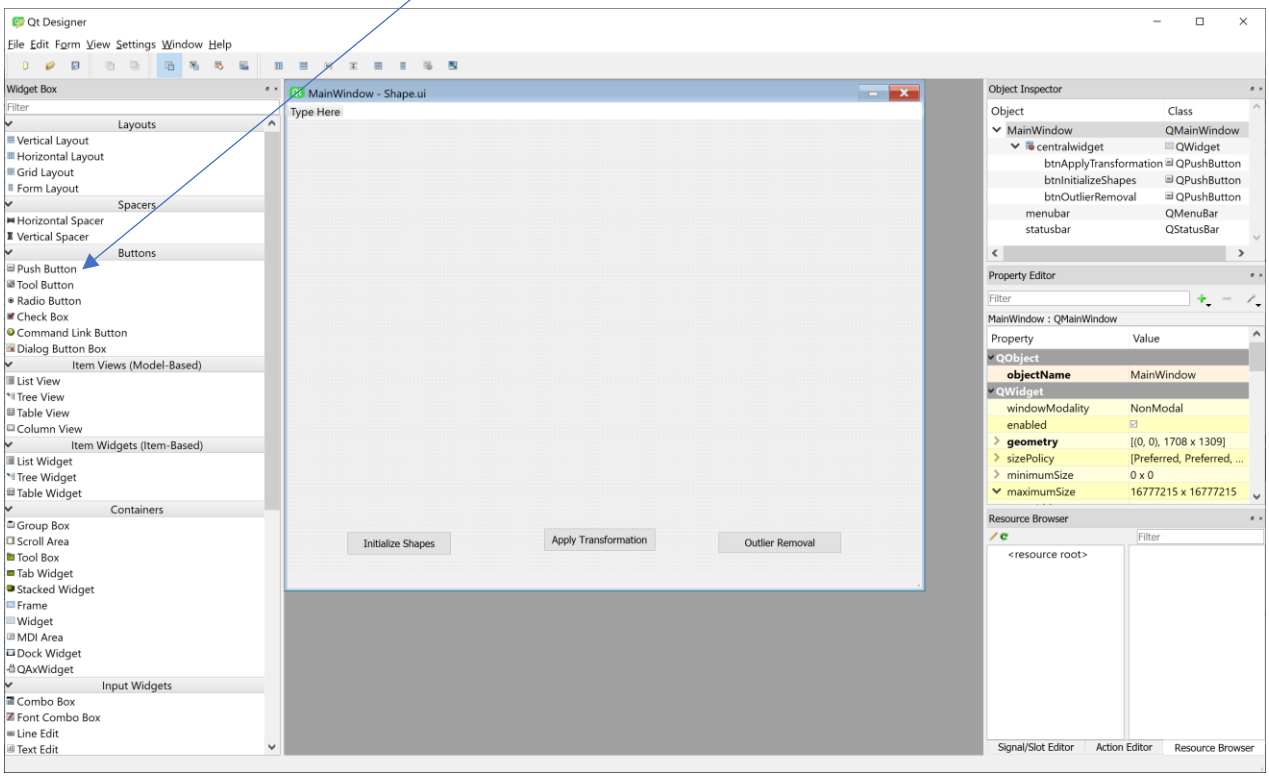
Then install Qt Designer using pip as:

pip install PyQt5Designer

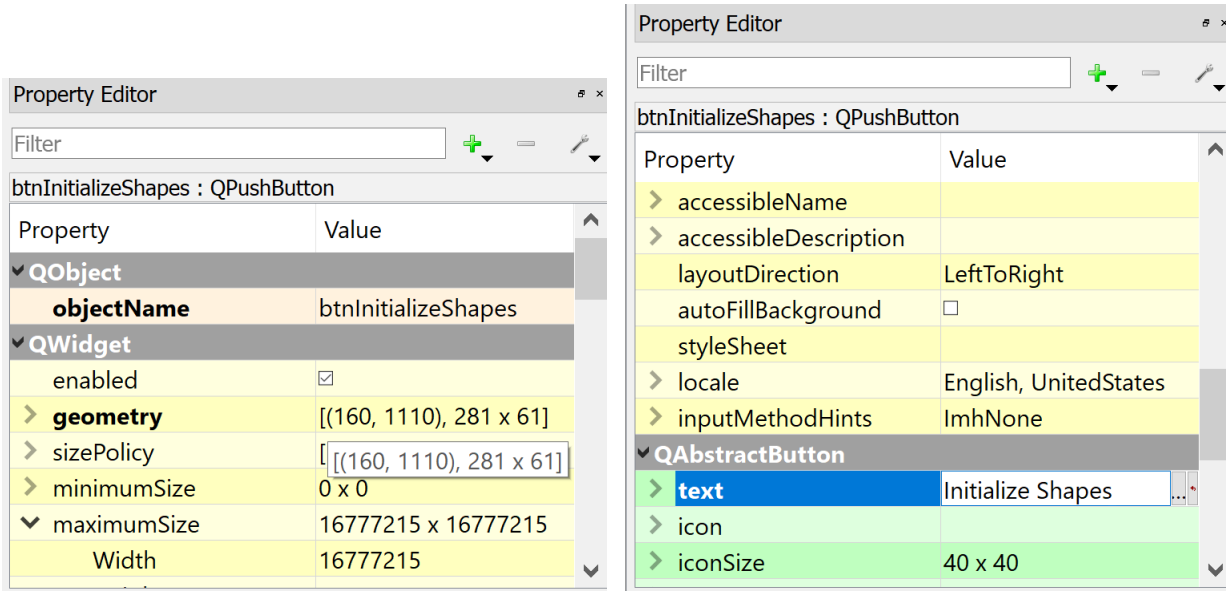
Once PyQt5Designer is installed, launch it by typing **designer** in the command prompt. Choose create main window.



Then drag and drop three push buttons on the window as shown below.



Fo each button, select it and then give it an object name, e.g., btnInitializeShapes for the first button, btnApplyTransformation for the second button, and btnOutlierRemoval for the third button. Similarly, give the button a proper text property as shown below.



Save the designer in a file e.g., shape.ui
Then from the command prompt move to the folder where you saved the shape.ui and issue the following command.

pyuic5 -x shape.ui -o ShapeUI.py

This will produce a python file for the UI called ShapeUI.py.

Now create a python application called ShapeTransformation and add the existing file ShapeUI.py to it.

Add a class to the project Transformation with the following code in it.

```
from PyQt5.QtCore import QPoint

class Transformation(object):
    def __init__(self) -> None:
        self.A = 0
        self.B = 0
        self.T1 = 0
        self.T2 = 0

    def apply_transformation(self, shape_list):  # T is the transformation
        TList = []
        for pt in shape_list:
            xprime = self.A * pt.x() + self.B * pt.y() + self.T1
            yprime = self.B * pt.x() * -1 + self.A * pt.y() + self.T2
            pTrans = QPoint(int(xprime), int(yprime))
            TList.append(pTrans)
        return TList;
```

Add a class called ComputeTransformation to the project with the following code in it. This class has the important method of compute_transformation_matrix that computes the transformation matrix given two sets of correspondence points.

```
import numpy as np
from Transformation import Transformation

class ComputeTransformation(object):
    @staticmethod
    def compute_transformation_matrix(shp1, shp2):
        A = np.zeros((4, 4))
        B = np.zeros((4, 1))
        for i in range(0,len(shp1)):
            A[0, 0] += 2 * shp2[i].x() * shp2[i].x() + 2 * shp2[i].y() *
shp2[i].y()
            A[0, 1] += 0
            A[0, 2] += 2 * shp2[i].x()
            A[0, 3] += 2 * shp2[i].y()

            A[1, 0] += 0
            A[1, 1] += 2 * shp2[i].x() * shp2[i].x() + 2 * shp2[i].y() *
shp2[i].y()
            A[1, 2] += 2 * shp2[i].y()
            A[1, 3] += -2 * shp2[i].x()

            A[2, 0] += -2 * shp2[i].x()
            A[2, 1] += -2 * shp2[i].y()
            A[2, 2] += -2;
            A[2, 3] += 0

            A[3, 0] += -2 * shp2[i].y()
            A[3, 1] += 2 * shp2[i].x()
            A[3, 2] += 0
            A[3, 3] += -2

            B[0, 0] += 2 * shp1[i].x() * shp2[i].x() + 2 * shp1[i].y() *
shp2[i].y()
            B[1, 0] += 2 * shp1[i].x() * shp2[i].y() - 2 * shp2[i].x() *
shp1[i].y()
            B[2, 0] += -2 * shp1[i].x()
            B[3, 0] += -2 * shp1[i].y()

        Ainv = np.linalg.inv(A)
        Res = np.dot(Ainv,B)
        T = Transformation()
        T.A = Res[0, 0]
        T.B = Res[1, 0]
        T.T1 = Res[2, 0]
        T.T2 = Res[3, 0]
        return T

    @staticmethod
    def compute_cost(P1List, P2List, T):
        cost = 0
        for i in range(0,len(P1List)):
            xprime = T.A * P2List[i].x() + T.B * P2List[i].y() + T.T1
            yprime = -1 * T.B * P2List[i].x() + T.A * P2List[i].y() + T.T2
            cost += ((P1List[i].x() - xprime) * (P1List[i].x() - xprime) +
                    (P1List[i].y() - yprime) * (P1List[i].y() - yprime))
            return cost;
```

Add a class called Outliers with the following code in it. The code here removes a pair of correspondence points from the two shapes if these results in the highest reduction in cost i.e., distance between correspondence points.

```
from Transformation import Transformation
from ComputeTransformation import ComputeTransformation

class Outliers(object):
    @staticmethod
    def RemoveOutliersFromCorrespondencePoints(shape1, shape2, numPointsToRemove):
        P1ListCopy = shape1.copy()
        P2ListCopy = shape2.copy()

        for i in range(numPointsToRemove):
            P1ListFil, P2ListFil =
Outliers.RemoveOneOutlierFromCorrespondencePoints(P1ListCopy, P2ListCopy)
            P1ListCopy = P1ListFil.copy()
            P2ListCopy = P2ListFil.copy()
        return P1ListCopy, P2ListCopy

    @staticmethod
    def RemoveOneOutlierFromCorrespondencePoints(P1List, P2List):
        matchCount = len(P1List)
        T = ComputeTransformation.compute_transformation_matrix(P1List, P2List)
        cost = ComputeTransformation.compute_cost(P1List, P2List, T)

        # make a copy of the original point lists
        P1ListCopy = P1List.copy()
        P2ListCopy = P2List.copy()
        P1ListFiltered = []
        P2ListFiltered = []

        bestIndexToRemove = -1
        highestCost = 9999999;
        costNew = 0;
        for i in range(len(P1ListCopy)):
            P1ListFiltered = []
            P2ListFiltered = []
            # find the best point that if removed reduces the cost significantly
            for j in range(len(P1ListCopy)):
                if (j != i):
                    P1ListFiltered.append(P1ListCopy[j])
                    P2ListFiltered.append(P2ListCopy[j])
            TR = ComputeTransformation.compute_transformation_matrix
(P1ListFiltered, P2ListFiltered)
            costNew = ComputeTransformation.compute_cost(P1ListFiltered,
P2ListFiltered, TR)
            if (costNew < highestCost):
                 highestCost = costNew;
                 bestIndexToRemove = i

        # update filtered lists based on best index that is to be removed
        P1ListFiltered = []
        P2ListFiltered = []
        for i in range(len(P1ListCopy)):
            # find the best point that if removed reduces the cost significantly
            if (i != bestIndexToRemove):
                P1ListFiltered.append(P1ListCopy[i])
                P2ListFiltered.append(P2ListCopy[i])
        return P1ListFiltered, P2ListFiltered
```

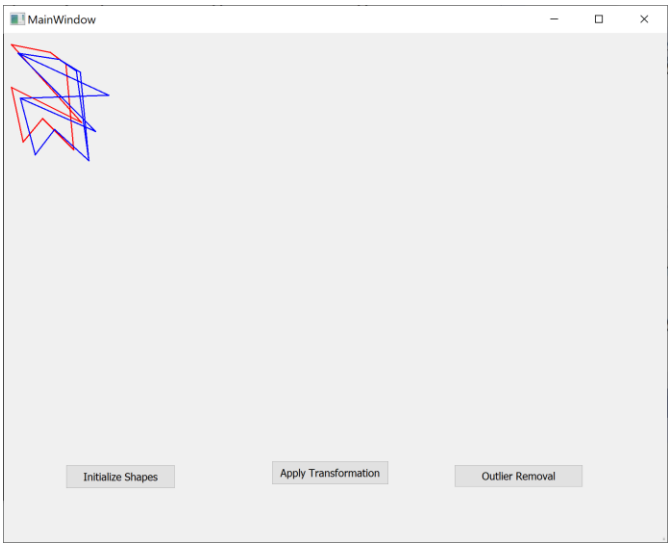
Type the following code in the main file i.e., ShapeTransformation.py.

```
import sys

from ShapeUI import Ui_MainWindow
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt, QLine, QPoint
from Transformation import Transformation
from ComputeTransformation import ComputeTransformation
from Outliers import Outliers


class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)

self.btnInitializeShapes.clicked.connect(self.btnInitializeShapes_clicked)

self.btnApplyTransformation.clicked.connect(self.btnApplyTransformation_clicked)
        self.btnOutlierRemoval.clicked.connect(self.btnOutlierRemoval_clicked)
        self.shape1 = []
        self.shape2 = []

    def btnInitializeShapes_clicked(self):
        print("initialize shapes..")
        self.shape1.append(QPoint(20,30))
        self.shape1.append(QPoint(120,50))
        self.shape1.append(QPoint(160,80))
        self.shape1.append(QPoint(180,300))
        self.shape1.append(QPoint(100,220))
        self.shape1.append(QPoint(50,280))
        self.shape1.append(QPoint(20,140))


        # transform shape1 to shape2
        transf = Transformation()
        transf.A = 1.05
        transf.B = 0.05
        transf.T1 = 15  # shift in X
        transf.T2 = 22  # shift in Y
        self.shape2 = transf.apply_transformation(self.shape1)
        self.shape2[2] = QPoint(self.shape2[2].x() + 10, self.shape2[2].y() + 3)
# change one point
        # add outliers to both shapes
        pt_outlier1 = QPoint(200, 230)
        self.shape1.append(pt_outlier1)
        pt_outlier2 = QPoint(270, 160)
        self.shape2.append(pt_outlier2)

        self.update()  # trigger paint event
```

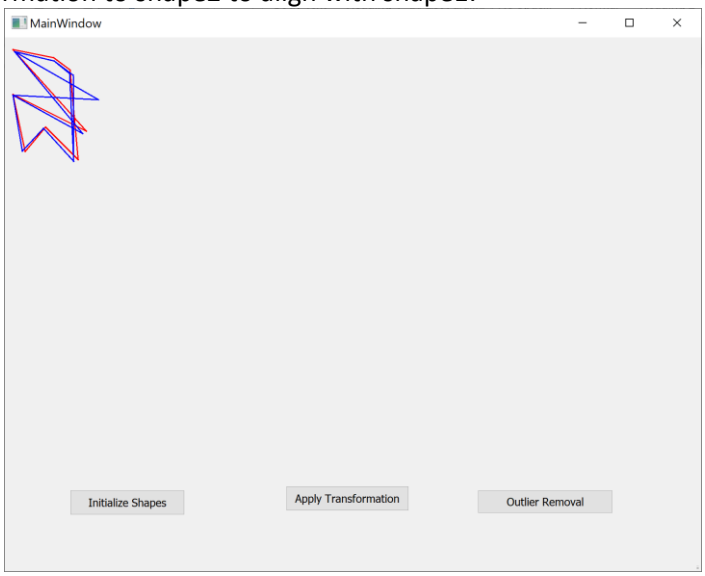
```
    def btnApplyTransformation_clicked(self):
        T = ComputeTransformation.compute_transformation_matrix(self.shape1,
self.shape2)
        cost = ComputeTransformation.compute_cost(self.shape1, self.shape2, T)
        msg = QMessageBox()
        msg.setText("cost = " + str(cost))
        msg.exec_()
        shape2t = T.apply_transformation(self.shape2)
        self.shape2 = shape2t
        self.update()  # trigger paint event

    def btnOutlierRemoval_clicked(self):
        shape1_filtered = []
        shape2_filtered = []
        numPointsToRemove = 2
        shape1_filtered, shape2_filtered =
Outliers.RemoveOutliersFromCorrespondencePoints(self.shape1, self.shape2,
numPointsToRemove)
        # shape1_filtered and shape2_filtered will contain points without
outliers

        # compute new transformation between source points and dest points based
        # on removal of outlier points
        T = ComputeTransformation.compute_transformation_matrix(shape1_filtered,
shape2_filtered)
        cost = ComputeTransformation.compute_cost(shape1_filtered,
shape2_filtered, T)
        shape2t = T.apply_transformation(shape2_filtered)
        self.shape2 = shape2t
        self.shape1 = shape1_filtered
        self.update()  # trigger paint event

    def paintEvent(self,event):
        QMainWindow.paintEvent(self, event)
        painter = QPainter(self)
        if len(self.shape1) > 0:
            pt_prev = self.shape1[0]
            pt_first = pt_prev
            for pt_current in self.shape1:
                pen = QPen(Qt.red, 3)
                painter.setPen(pen)
                painter.drawLine(pt_prev.x(), pt_prev.y(), pt_current.x(),
pt_current.y())
                pt_prev = pt_current
            painter.drawLine(pt_current.x(), pt_current.y(),pt_first.x(),
pt_first.y())

        if len(self.shape2) > 0:
            pt_prev = self.shape2[0]
            pt_first = pt_prev
            for pt_current in self.shape2:
                pen = QPen(Qt.blue, 3)
                painter.setPen(pen)
                painter.drawLine(pt_prev.x(), pt_prev.y(), pt_current.x(),
pt_current.y())
                pt_prev = pt_current
            painter.drawLine(pt_current.x(), pt_current.y(),pt_first.x(),
pt_first.y())
```

```
def main():
    # run the following command to convert ui file generated by designer in qt
to py
    # pyuic5 -x shape.ui -o ShapeUI.py
    app = QApplication(sys.argv)
    main_window = MainWindow()
    main_window.show()
    sys.exit(app.exec_())

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

Test the program by clicking on the buttons from left to right to see how the middle button does the shape alignment after calculating the transformation matrix for shape2 that will align it with shape1. The right most button removes two outliers from the two shapes and then computes the transformation to perfectly align the two shapes.
After initialize shapes.



After applying transformation to shape2 to align with shape1.



After outlier removals.