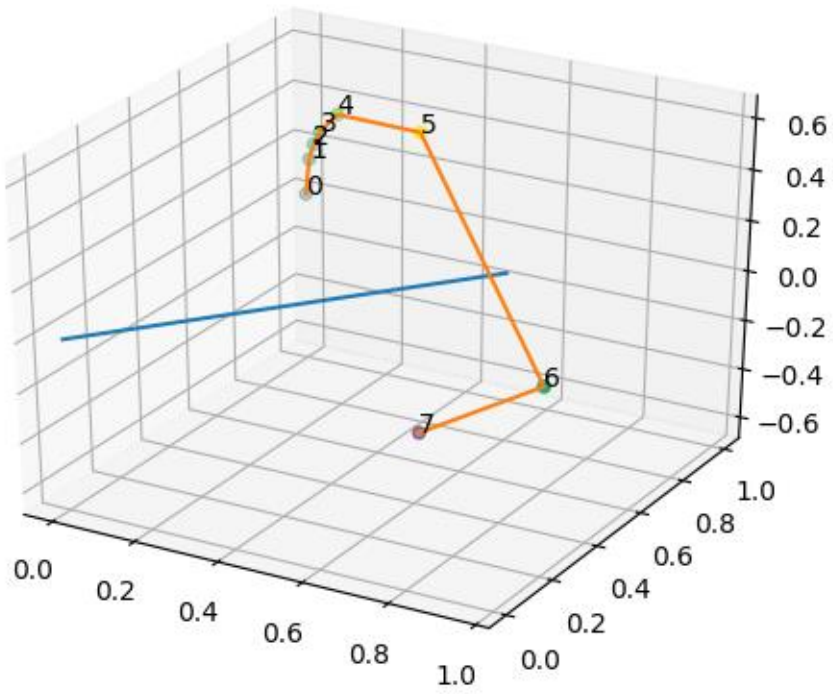
Appendix 1

Problem 2.3



s = [0.6 0.8 0.]
u = [-0.8 0.6 0.]
v = [0 0 1]
Point: [0 1 0]

Phi = 0.0:
The rotation matrix R:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Eigenvalues:
[1. 1. 1.]
Eigenvector: [1. 0. 0.]
SUV representation: 0.6s - 0.8u + 0.0v
Eigenvector: [0. 1. 0.]
SUV representation: 0.7999999999999998s + 0.6000000000000001u + 0.0v
Eigenvector: [0. 0. 1.]
SUV representation: 0.0s + 0.0u + 1.0v

Phi = 0.2617993877991494:

The rotation matrix R:
[[0.97819253 0.0163556   0.20705524]
 [0.0163556   0.9877333 -0.15529143]
 [-0.20705524 0.15529143 0.96592583]]
Eigenvalues:
[0.96592583+0.25881905j 0.96592583-0.25881905j 1.   +0.   j]
Eigenvector: [2.02615702e-15+0.56568542j 2.02615702e-15-0.56568542j
 6.00000000e-01+0.   j]
SUV representation: (2.8366198279172746e-15-0.11313708498984759j)s + (-4.052314039881822e-16-0.791959594928934j)u + (0.6000000000000006+0j)v
Eigenvector: [-1.38777878e-16-0.42426407j -1.38777878e-16+0.42426407j
  8.00000000e-01+0.   j]
SUV representation: (-1.9428902930940237e-16+0.08485281374238546j)s + (2.77555756156289e-17+0.5939696961966988j)u + (0.7999999999999993+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j -1.19297591e-16+0.j]
SUV representation: (-0.989949493661666+0j)s + (0.14142135623730945+0j)u + (-1.1929759116026628e-16+0j)v

Phi = 0.39269908169872414:
The rotation matrix R:
[[0.9512829   0.03653782 0.30614675]
 [0.03653782 0.97259663 -0.22961006]
 [-0.30614675 0.22961006 0.92387953]]
Eigenvalues:
[0.92387953+0.38268343j 0.92387953-0.38268343j 1.   +0.   j]
Eigenvector: [2.77555756e-17-0.56568542j 2.77555756e-17+0.56568542j
 6.00000000e-01+0.   j]
SUV representation: (3.885780586188047e-17+0.11313708498984743j)s + (-5.551115123125781e-18+0.7919595949289329j)u + (0.6000000000000001+0j)v
Eigenvector: [-1.38777878e-16+0.42426407j -1.38777878e-16-0.42426407j
  8.00000000e-01+0.   j]
SUV representation: (-1.9428902930940237e-16-0.08485281374238557j)s + (2.77555756156289e-17-0.5939696961966994j)u + (0.7999999999999998+0j)v
Eigenvector: [7.07106781e-01+0.j 7.07106781e-01-0.j -1.22728786e-16+0.j]
SUV representation: (0.989949493661167+0j)s + (-0.14142135623730956+0j)u + (-1.227287863770378e-16+0j)v

Phi = 0.5235987755982988:
The rotation matrix R:
[[0.91425626 0.06430781 0.4   ]
 [0.06430781 0.95176915 -0.3   ]
 [-0.4   0.3   0.8660254]]
Eigenvalues:
[0.8660254+0.5j 0.8660254-0.5j 1.   +0.j]
Eigenvector: [7.07767178e-16+0.56568542j 7.07767178e-16-0.56568542j
 6.00000000e-01+0.   j]
SUV representation: (9.90874049477952e-16-0.11313708498984748j)s + (-1.4155343563970745e-16-0.791959594928933j)u + (0.6000000000000003+0j)v
Eigenvector: [-3.12250226e-16-0.42426407j -3.12250226e-16+0.42426407j

8.00000000e-01+0.   j]
SUV representation: (-4.3715031594615534e-16+0.08485281374238557j)s + (6.245004513516503e-17+0.5939696961966995j)u + (0.7999999999999997+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j -3.88353777e-16+0.j]
SUV representation: (-0.989949493661167+0j)s + (0.14142135623730956+0j)u + (-3.8835377691477215e-16+0j)v

Phi = 0.7853981633974483:
The rotation matrix R:
[[0.81254834 0.14058875 0.56568542]
 [0.14058875 0.89455844 -0.42426407]
 [-0.56568542 0.42426407 0.70710678]]
Eigenvalues:
[0.70710678+0.70710678j 0.70710678-0.70710678j 1.   +0.   j]
Eigenvector: [1.2490009e-16+0.56568542j 1.2490009e-16-0.56568542j
 6.0000000e-01+0.   j]
SUV representation: (1.7486012637846213e-16-0.11313708498984754j)s + (-2.498001805406602e-17-0.7919595949289333j)u + (0.6000000000000001+0j)v
Eigenvector: [-4.16333634e-17-0.42426407j -4.16333634e-17+0.42426407j
  8.00000000e-01+0.   j]
SUV representation: (-5.828670879282071e-17+0.08485281374238562j)s + (8.326672684688675e-18+0.5939696961967j)u + (0.7999999999999999+0j)v
Eigenvector: [-7.07106781e-01+0.j -7.07106781e-01-0.j 1.60936741e-16+0.j]
SUV representation: (-0.9899494936611666+0j)s + (0.14142135623730945+0j)u + (1.6093674092148539e-16+0j)v

Phi = 1.5707963267948966:
The rotation matrix R:
[[3.60000000e-01 4.80000000e-01 8.00000000e-01]
 [4.80000000e-01 6.40000000e-01 -6.00000000e-01]
 [-8.00000000e-01 6.00000000e-01 1.11022302e-16]]
Eigenvalues:
[1.11022302e-16+1.j 1.11022302e-16-1.j 1.00000000e+00+0.j]
Eigenvector: [-3.88578059e-16-0.56568542j -3.88578059e-16+0.56568542j
  6.00000000e-01+0.   j]
SUV representation: (-5.440092820663266e-16+0.11313708498984754j)s + (7.771561172376093e-17+0.7919595949289335j)u + (0.6+0j)v
Eigenvector: [2.22044605e-16+0.42426407j 2.22044605e-16-0.42426407j
 8.00000000e-01+0.   j]
SUV representation: (3.108624468950438e-16-0.08485281374238562j)s + (-4.440892098500625e-17-0.5939696961966998j)u + (0.8+0j)v
Eigenvector: [7.07106781e-01+0.j 7.07106781e-01-0.j -2.64436103e-16+0.j]
SUV representation: (0.9899494936611662+0j)s + (-0.14142135623730945+0j)u + (-2.6443610263447547e-16+0j)v

Phi = 3.141592653589793:
The rotation matrix R:
[[-2.80000000e-01 9.60000000e-01 9.79717439e-17]
 [9.60000000e-01 2.80000000e-01 -7.34788079e-17]

[-9.79717439e-17 7.34788079e-17 -1.00000000e+00]]
Eigenvalues:
[-1. 1. -1.]
Eigenvector: [-0.8   0.6   0.38635862]
SUV representation: -1.1102230246251565e-16s + 1.0000000000000002u + 0.3863586229535932v
Eigenvector: [0.6   0.8   -0.28976897]
SUV representation: 0.9999999999999999s + 1.1102230246251565e-16u + -0.2897689672151948v
Eigenvector: [-6.12323400e-17 -8.16431199e-17 8.75648879e-01]
SUV representation: -1.020538999289461e-16s + -2.465190328815662e-32u + 0.8756488794650756v

Phi = 4.71238898038469:
The rotation matrix R:
[[3.60000000e-01 4.80000000e-01 -8.00000000e-01]
 [4.80000000e-01 6.40000000e-01 6.00000000e-01]
 [8.00000000e-01 -6.00000000e-01 -2.22044605e-16]]
Eigenvalues:
[-4.4408921e-16+1.j -4.4408921e-16-1.j 1.0000000e+00+0.j]
Eigenvector: [-6.9388939e-17+0.56568542j -6.9388939e-17-0.56568542j
 -6.0000000e-01+0.   j]
SUV representation: (-9.714451465470118e-17-0.11313708498984754j)s + (1.387778780781445e-17-0.7919595949289335j)u + (-0.6+0j)v
Eigenvector: [8.32667268e-17-0.42426407j 8.32667268e-17+0.42426407j
 -8.00000000e-01+0.   j]
SUV representation: (1.1657341758564142e-16+0.08485281374238562j)s + (-1.665334536937735e-17+0.5939696961967j)u + (-0.8+0j)v
Eigenvector: [7.07106781e-01+0.j 7.07106781e-01-0.j -9.81955080e-17+0.j]
SUV representation: (0.9899494936611668+0j)s + (-0.1414213562373095+0j)u + (-9.819550802290167e-17+0j)v

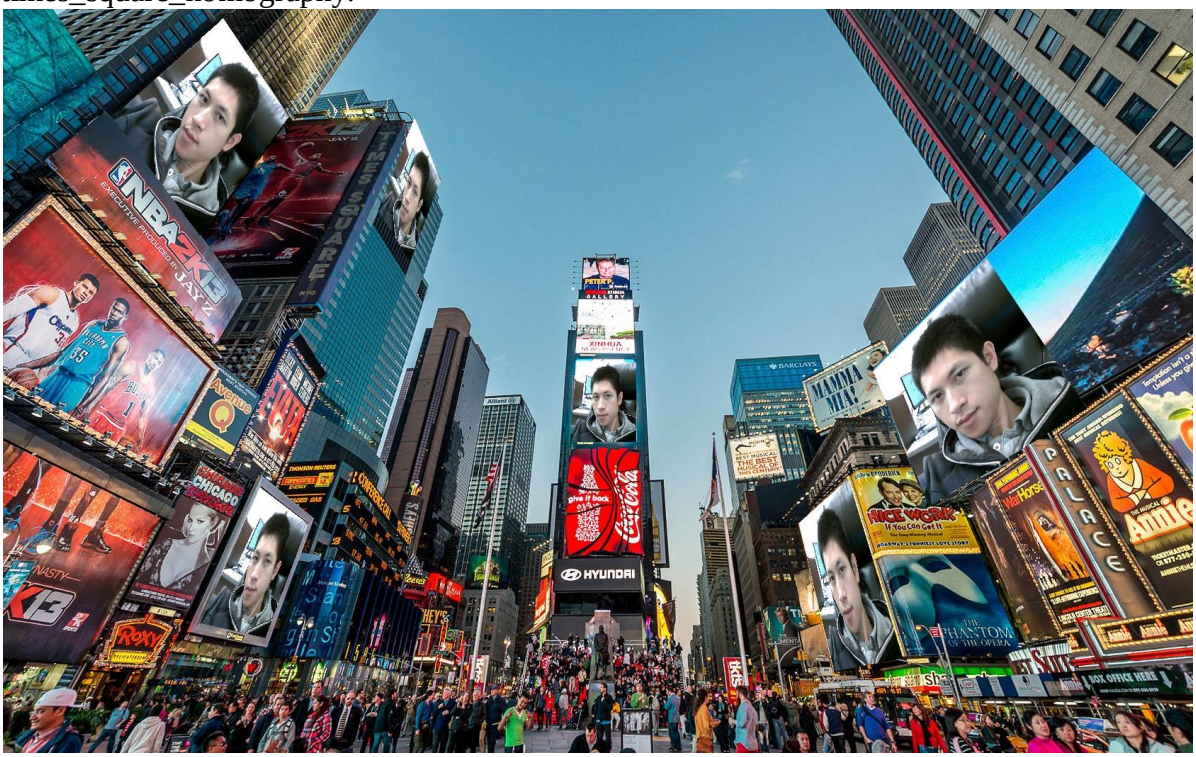
Appendix 2

Problem 3.4

times_square_affine:

hometown_affine:

times_square_homography:



hometown_homography:

Problem 3.5

computer_screen_rectified_homography:

flagellation_rectified_homography:

computer_screen_rectified_affine:

lagellation_rectified_affine:

Appendix 3

Coding of problem 2: compute_R.py and rot_to_ax_phi.py

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

def compute_R(phi, s):
        S = [0, -s[2], s[1],
                s[2], 0, -s[0],
                -s[1], s[0], 0]
        S = np.array(S).reshape(3, 3)
        I = np.identity(3)
        R = I + np.sin(phi) * S + (1 - np.cos(phi)) * S.dot(S)
        return R


def find_suv_coordinates(s, u, v, vector):
        SUV = np.stack((s, u, v))
        SUV = SUV.T
        return LA.inv(SUV).dot(vector)


def rot_to_ax_phi(R):
        trR = R[0, 0] + R[1, 1] + R[2, 2]
        phi = np.arccos(0.5 * (trR - 1))
        ax = 1 / (2 * np.sin(phi)) \
                * np.array([R[2,1] - R[1,2], R[0,2] - R[2,0], R[1,0] - R[0,1]])
        return phi, ax


if __name__ == "__main__":
        origin = np.array([0, 0, 0])
        s = np.array([0.6, 0.8, 0])
        u = np.array([-0.8, 0.6, 0])
        v = np.array([0, 0, 1])
        p = np.array([0, 1, 0])
        p_list = []
        phi_list = [0, np.pi/12, np.pi/8, np.pi/6,
                                np.pi/4, np.pi/2, np.pi, 3*np.pi/2]
        phi_arr = np.array(phi_list)

        print("s = " + str(s))
        print("u = " + str(u))
        print("v = " + str(v))
        print("Point: " + str(p))
```

```
for phi in phi_arr:
        R = compute_R(phi, s)
        print("Phi = " + str(phi) + ":")
        print("The rotation matrix R:")
        print(R)
        p_new = R.dot(p)
        p_list.append(p_new)
        eigenvalues, eigenvectors = LA.eig(R)
        print("Eigenvalues:")
        print(eigenvalues)
        for vector in eigenvectors:
                print("Eigenvector: " + str(vector))
                x, y, z = find_suv_coordinates(s, u, v, vector)
                print("SUV representation: " + str(x) + "s + " + str(y) + "u + " + str(z) + "v")
        print()

rotation_axis = np.stack((origin, s))
rotated_points = np.stack(p_list)

fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot3D(rotation_axis[:,0], rotation_axis[:,1], rotation_axis[:,2])
c = rotated_points[:,0] + rotated_points[:,1] + rotated_points[:,2]
ax.plot3D(rotated_points[:,0], rotated_points[:,1], rotated_points[:,2])
ax.scatter(rotated_points[:,0], rotated_points[:,1], rotated_points[:,2], c=c)
for i in range(len(phi_list)):
        ax.text(rotated_points[i,0], rotated_points[i,1], rotated_points[i,2], str(i))
plt.show()
```

Appendix 4

Coding of problem 3: affine_solve.py, homography_solve.py and homography_transform.py

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt

times_square = "./hw1_package/hw1_package/images/times_square.jpg"
hometown = "./hw1_package/hw1_package/images/Rach-Gia-City-worth-exploring.jpg"
my_pic = "./hw1_package/hw1_package/images/my_pic.jpg"
computer_screen = "./hw1_package/hw1_package/images/computer_screen.png"
flagellation = "./hw1_package/hw1_package/images/the_flagellation.jpg"

times_square_surfaces = [[[497, 810], [499, 893], [611, 801], [610, 894]],
                                                    [[155, 581], [241,617], [284, 530], [355, 575]],
                                                    [[676, 363], [727, 430], [866, 279], [887, 370]],
                                                    [[726, 1124], [664, 1193], [934, 1174], [916,
1270]],
                                                    [[511, 1229], [352, 1388], [701, 1308], [564,
1530]],
                                                    [[14, 311], [152, 403], [153, 155], [319, 278]]]

hometown_surfaces = [[[300, 1344], [236, 1561], [619, 1348], [586, 1565]],
                                                [[1081, 827], [1065, 1147], [1267, 828], [1228, 1147]]]

computer_screen_surface = [[203, 634], [302, 864], [575, 690], [826, 903]]
computer_screen_surface_rectified = [[0, 0], [0, 300], [200, 0], [200, 300]]

flagellation_surface = [[627, 280], [626, 567], [676, 84], [676, 578]]
flagellation_surface_rectified = [[0, 0], [0, 200], [600, 0], [600, 200]]

my_pic_surface = [[0, 0], [0, 719], [719, 0], [719, 719]]

def affine_solve(u, v):
        X = []

        for j in range(u.shape[1]):
                X_temp = [[u[0,j], u[1,j], 1, 0, 0, 0],
                                        [0, 0, 0, u[0,j], u[1,j], 1]]
                X.extend(X_temp)

        X = np.array(X)
        y = v.T.flatten()
        h = LA.inv(X.T.dot(X)).dot(X.T).dot(y)
        h = np.concatenate((h, np.array([0, 0, 1])))
        H = h.reshape(3, 3)
        return H
```

```
def homography_solve(u, v):
	X = []

	for j in range(u.shape[1]):
		X_temp = [[u[0,j], u[1,j], 1, 0, 0, 0, -u[0,j]*v[0,j], -u[1,j]*v[0,j]],
					[0, 0, 0, u[0,j], u[1,j], 1, -u[0,j]*v[1,j], -u[1,j]*v[1,j]]]
		X.extend(X_temp)

	X = np.array(X)
	y = v.T.flatten()
	h = LA.inv(X.T.dot(X)).dot(X.T).dot(y)
	h = np.concatenate((h, np.array([1])))
	H = h.reshape(3, 3)
	return H

def homography_transform(u, H):
	U = np.concatenate([u, np.ones([1, u.shape[1]])])
	V = H.dot(U)
	for j in range(V.shape[1]):
		V[:, j] = V[:, j] / V[2, j]
	v = V[:-1,:].astype(int)
	print(v)
	return v


if __name__ == "__main__":
	times_square_im = plt.imread(times_square)
	hometown_im = plt.imread(hometown)
	my_pic_im = plt.imread(my_pic)
	x_size, y_size, _ = my_pic_im.shape

	times_square_im_homo = np.copy(times_square_im)
	times_square_im_homo.setflags(write=1)
	for surface in times_square_surfaces:
		u = np.array(my_pic_surface).T
		v = np.array(surface).T
		H = homography_solve(u, v)
		U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
		V = homography_transform(U, H)

		for i in range(U.shape[1]):
			times_square_im_homo[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

	plt.imsave("times_square_homography.jpg", times_square_im_homo, format="jpg")
	plt.imshow(times_square_im_homo)
	plt.show()

	hometown_im_homo = np.copy(hometown_im)
```

```
	hometown_im_homo.setflags(write=1)
	for surface in hometown_surfaces:
		u = np.array(my_pic_surface).T
		v = np.array(surface).T
		H = homography_solve(u, v)
		U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
		V = homography_transform(U, H)

		for i in range(U.shape[1]):
			hometown_im_homo[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

	plt.imsave("Rach-Gia-City-worth-exploring_homography.jpg", hometown_im_homo,
format="jpg")
	plt.imshow(hometown_im_homo)
	plt.show()

	times_square_im_affine = np.copy(times_square_im)
	times_square_im_affine.setflags(write=1)
	for surface in times_square_surfaces:
		u = np.array(my_pic_surface[:-1]).T
		v = np.array(surface[:-1]).T
		H = affine_solve(u, v)
		U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
		V = homography_transform(U, H)

		for i in range(U.shape[1]):
			times_square_im_affine[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

	plt.imsave("times_square_affine.jpg", times_square_im_affine, format="jpg")
	plt.imshow(times_square_im_affine)
	plt.show()

	hometown_im_affine = np.copy(hometown_im)
	hometown_im_affine.setflags(write=1)
	for surface in hometown_surfaces:
		u = np.array(my_pic_surface[:-1]).T
		v = np.array(surface[:-1]).T
		H = affine_solve(u, v)
		U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
		V = homography_transform(U, H)

		for i in range(U.shape[1]):
			hometown_im_affine[V[0,i], V[1,i], :] = my_pic_im[U[0,i], U[1,i], :]

	plt.imsave("Rach-Gia-City-worth-exploring_affine.jpg", hometown_im_affine, format="jpg")
	plt.imshow(hometown_im_affine)
	plt.show()

	#### Rectify computer screen and flagellation ####
```

```
	computer_screen_im = plt.imread(computer_screen)
	flagellation_im = plt.imread(flagellation)

	x_size, y_size = 200, 300
	computer_screen_im_rectified_homo = np.ones([x_size, y_size, 3], "float")
	u = np.array(computer_screen_surface_rectified).T
	v = np.array(computer_screen_surface).T
	H = homography_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		computer_screen_im_rectified_homo[U[0,i], U[1,i], :] = computer_screen_im[V[0,i],
V[1,i], :-1]

	plt.imsave("computer_screen_rectified_homography.jpg",
computer_screen_im_rectified_homo, format="jpg")
	plt.imshow(computer_screen_im_rectified_homo)
	plt.show()

	computer_screen_im_rectified_affine = np.ones([x_size, y_size, 3], "float")
	u = np.array(computer_screen_surface_rectified[:-1]).T
	v = np.array(computer_screen_surface[:-1]).T
	H = affine_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		computer_screen_im_rectified_affine[U[0,i], U[1,i], :] = computer_screen_im[V[0,i],
V[1,i], :-1]

	plt.imsave("computer_screen_rectified_affine.jpg", computer_screen_im_rectified_affine,
format="jpg")
	plt.imshow(computer_screen_im_rectified_affine)
	plt.show()


	x_size, y_size = 600, 200
	flagellation_im_rectified_homo = np.ones([x_size, y_size, 3], "uint8") * 255
	u = np.array(flagellation_surface_rectified).T
	v = np.array(flagellation_surface).T
	H = homography_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		flagellation_im_rectified_homo[U[0,i], U[1,i], :] = flagellation_im[V[0,i], V[1,i], :]
```

```
	plt.imsave("flagellation_rectified_homography.jpg", flagellation_im_rectified_homo,
format="jpg")
	plt.imshow(flagellation_im_rectified_homo)
	plt.show()

	x_size, y_size = 600, 200
	flagellation_im_rectified_affine = np.ones([x_size, y_size, 3], "uint8") * 255
	u = np.array(flagellation_surface_rectified[:-1]).T
	v = np.array(flagellation_surface[:-1]).T
	H = affine_solve(u, v)
	U = np.array([[i, j] for i in range(x_size) for j in range(y_size)]).T
	V = homography_transform(U, H)

	for i in range(U.shape[1]):
		flagellation_im_rectified_affine[U[0,i], U[1,i], :] = flagellation_im[V[0,i], V[1,i], :]

	plt.imsave("flagellation_rectified_affine.jpg", flagellation_im_rectified_affine, format="jpg")
	plt.imshow(flagellation_im_rectified_affine)
	plt.show()
```