3DCV HW2 report

r09922115 朱世耘

Problem 1

1-1

我參考這篇 ECCV2018 的paper: Lambda Twist: An Accurate Fast Robust Perspective Three Point (P3P) Solver. 使用了paper中的演算法：

Algorithm 1 Lambda Twist P3P

function MIX($\boldsymbol{n}, \boldsymbol{m}$) $= (\boldsymbol{n}, \boldsymbol{m}, \boldsymbol{n} \times \boldsymbol{m})$
function P3P($\boldsymbol{y}_{1:3}, \boldsymbol{x}_{1:3}$)
  Normalize $\boldsymbol{y}_i = \boldsymbol{y}_i / |\boldsymbol{y}_i|$
  Compute a_{ij} and b_{ij} according to (3)
  Construct $\mathbf{D}_1$ and $\mathbf{D}_2$ from (5) and (6)
  Compute a real root γ to (8)-(10) of the cubic equation
  $\mathbf{D}_0 = \mathbf{D}_1 + \gamma \mathbf{D}_2$
  $[\mathbf{E}, \sigma_1, \sigma_2] =$ EIG3X3KNOWN0 ($\mathbf{D}_0$). See Algorithm 2
  $s = \pm\sqrt{\frac{-\sigma_2}{\sigma_1}}$
  Compute the $\tau_k > 0, \tau_k \in \mathbb{R}$ for each s using Eqn (14) with coefficients in Eqn (15)
  Compute $\boldsymbol{\Lambda}_k$ according to Eqn (16), $\lambda_{3k} = \tau_k \lambda_{2k}$ and Eqn (13), $\lambda_{1k} > 0$
  $\mathbf{X}_{\text{inv}} = (\text{mix}(\boldsymbol{x}_1 - \boldsymbol{x}_2, \boldsymbol{x}_1 - \boldsymbol{x}_3))^{-1}$
  for each valid $\boldsymbol{\Lambda}_k$ **do**
    Gauss-Newton-Refine($\boldsymbol{\Lambda}_k$), see Section 3.8
    $\mathbf{Y}_k =$ MIX($\lambda_{1k}\boldsymbol{y}_1 - \lambda_{2k}\boldsymbol{y}_2, \lambda_{1k}\boldsymbol{y}_1 - \lambda_{3k}\boldsymbol{y}_3$)
    $\mathbf{R}_k = \mathbf{Y}_k \mathbf{X}_{\text{inv}}$
    $\boldsymbol{t}_k = \lambda_{1k}\boldsymbol{y}_1 - \mathbf{R}_k \boldsymbol{x}_1$
  Return all $\mathbf{R}_k, \boldsymbol{t}_k$

這篇的作者宣稱他們的方法是目前最快，而且accuracy最高的，他的概念是去解出 $Rx + T = \lambda \times y$ 中的 λ ，但是我重現了他的code之後（我的python程式和官方的c++的output是一樣的），發現他return的R和T並不是內部參數的R、T，後來試了一下還是不知道怎麼用得到的R、T來算相機位置，因此先把[R|T]當成3D到2D的投影矩陣，但是會有很大的誤差，因此1-2和2-1會用opencv的p3p再做一次。

我做ransac的方式是每次找三個點得到轉換的矩陣，把所有的3D點轉到圖片上，再去算inlier的數量，inlier最多的時候就把這個轉換矩陣存下來，總共做500次。

```python
def p3p_v2(points3D, points2D, cameraMatrix, distCoeffs):
    # preprocessing
    x = deepcopy(points3D)
    y = deepcopy(points2D)
    y = np.concatenate((y, np.ones((3, 1))), axis=1)

    x1, x2, x3 = x
    y1, y2, y3 = y

    # normalize y
    y1 = y1/np.linalg.norm(y1, 2)
    y2 = y2/np.linalg.norm(y2, 2)
    y3 = y3/np.linalg.norm(y3, 2)

    # compute a_ij, b_ij
    b12 = -2.0*np.dot(y1, y2)
    b13 = -2.0*np.dot(y1, y3)
    b23 = -2.0*np.dot(y2, y3)

    d12 = x1-x2
    d13 = x1-x3
    d23 = x2-x3

    a12 = np.linalg.norm(d12, 2)**2
    a13 = np.linalg.norm(d13, 2)**2
    a23 = np.linalg.norm(d23, 2)**2

    # get D1,D2, and find r
    c31 = -0.5*b13
    c23 = -0.5*b23
    c12 = -0.5*b12
    blob = (c12*c23*c31-1.0)

    s31_squared = 1.0 - c31*c31
    s23_squared = 1.0 - c23*c23
    s12_squared = 1.0 - c12*c12

    p3 = (a13*(a23*s31_squared - a13*s23_squared))
    p2 = 2.0*blob*a23*a13 + a13*(2.0*a12 + a13) * s23_squared + a23*(a23 - a12)*s31_squared
    p1 = a23*(a13 - a23)*s12_squared - a12*a12*s23_squared - 2.0*a12*(blob*a23 + a13*s23_squared)
    p0 = a12*(a12*s23_squared - a23*s12_squared)

    roots = np.roots([p3, p2, p1, p0])
    for r in roots[::-1]:
        if np.isreal(r) == True:
            root = np.real(r)
            break
    # solve eigenvalue [ E, sigma_1, sigma_2]

    A00 = a23*(1.0 - root)
    A01 = (a23*b12)*0.5
    A02 = (a23*b13*root)*(-0.5)
    A11 = a23 - a12 + a13*root
    A12 = b23*(a13*root - a12)*0.5
    A22 = root*(a13 - a23) - a12

    A=np.array([[A00,A01,A02],[A01,A11,A12],[A02,A12,A22]])
    sigmas, E = np.linalg.eig(A)
    new_sigmas, new_E=order_eigenvalue(sigmas,E)
    sigma = max(0,-1*new_sigmas[1]/new_sigmas[0])

    # s = +- (-sigma_2/sigma_1)**(1/2)
    s = np.zeros((2))
    s[0] = np.sqrt(sigma)
    s[1] = -1*np.sqrt(sigma)

    # find valid tau
    good_lamb = []

    for i in range(2):
        avaliable_root = []
        w0 = (new_E[1][0]-s[i]*new_E[1][1])/(s[i]*new_E[0][1]-new_E[0][0])
        w1 = (new_E[2][0]-s[i]*new_E[2][1])/(s[i]*new_E[0][1]-new_E[0][0])

        aa = (a13-a12)*w1*w1-a12*b13*w1-a12
        bb = a13*b12*w1-a12*b13*w0-2*w0*w1*(a12-a13)
        cc = (a13-a12)*w0*w0+a13*b12*w0+a13

        two_roots = np.roots([aa, bb, cc])
        for r in two_roots:
            if np.isreal(r) == True:
                if r >= 0:
                    avaliable_root.append(np.real(r))

        tau = np.array(avaliable_root)
```

```
    # compute the t_k > 0 and get A_k
        for t in tau:
            lamb2 = np.sqrt(a23/(t*(b23+t)+1.0))
            lamb3 = lamb2*t
            lamb1 = w0*lamb2+w1*lamb3

            if lamb1 >= 0:
                good_lamb.append([lamb1, lamb2, lamb3])

    # compute X_inv
    X=mix(d12, d13)
    X_inv = np.linalg.inv(X)

    answer = []
    # for each valid A, calculate R, T
    for lambs in good_lamb:

        # compute Y_k
        z1 = lambs[0]*y1-lambs[1]*y2
        z2 = lambs[0]*y1-lambs[2]*y3
        Y = mix(z1, z2)

        # compute R_k
        R = Y@X_inv

        # compute T_k
        T = lambs[0]*y1-R@x1
        T=T.reshape((3,1))

        answer.append([R,T])

    return answer
```

```
def ransac(R,T,points3D, points2D, num,K,p):
    cnt=0
    l=[]
    for i in range(num):
        x=np.array(points3D[i]).reshape(3,1)
        y=points2D[i]
        tmp=R@x+T
        tmp/=tmp[-1]
        inlier=calculate_distance(y,tmp,100)
        if inlier==1:
            l.append(i)
        cnt+=inlier
    return cnt
```

1-2

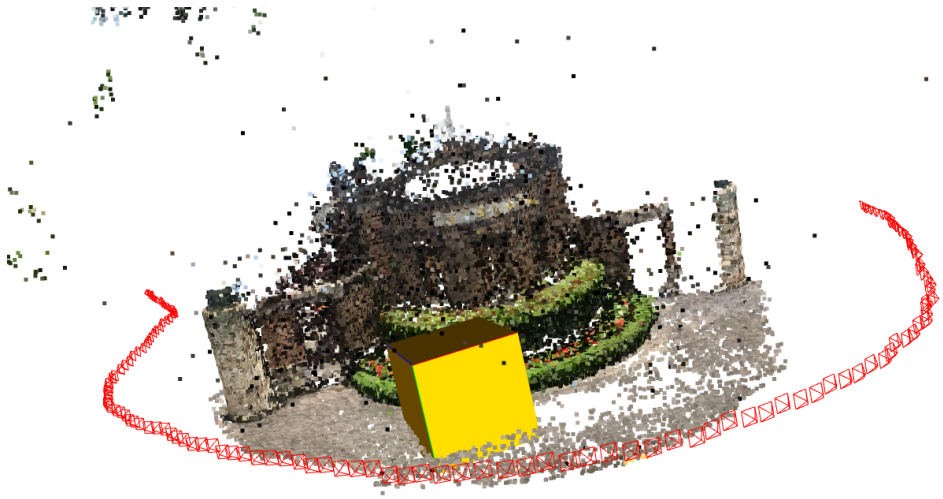
這題要對每張圖片找出拍攝的位置，並且畫在3D的空間中，我使用o3d.geometry.LineSet()來畫線，相機位置的部分是用前面算出來的結果，轉成矩陣後concatenate後作反矩陣再拆開得到相機位置和角度。

```
def find_camera_position(rvec, tvec):
    r_matrix=R.from_rotvec(rvec.reshape(1,3)).as_matrix().reshape(3,3)
    t_matrix = tvec.reshape(3,1)
    R_T=np.concatenate((r_matrix, t_matrix), axis=1)
    tmp=np.array([[0,0,0,1]])
    R_T=np.concatenate((R_T,tmp),axis=0)
    R_inverse=np.linalg.inv(R_T)
    R_matrix=R_inverse[:3,:3]
    T_matrix=R_inverse[:3,3]
    return R_matrix,T_matrix
```

```
def draw_camera(vis,data):
    p=data['position']
    r=data['rotation']
    model=o3d.geometry.LineSet()
    model.points=o3d.utility.Vector3dVector([[0,0,0],[1,1,1],[1,-1,1],[-1,-1,1],[-1,1,1]])
    model.lines=o3d.utility.Vector2iVector([[0,1],[0,2],[0,3],[0,4],[1,2],[2,3],[3,4],[4,1]])
    color=np.array([1,0,0])
    model.colors=o3d.utility.Vector3dVector(np.tile(color,(8,1)))
    model.scale(0.05,np.zeros(3))
    model.rotate(r)
    model.translate(p)
    vis.add_geometry(model)
    return model
```

result:
紅色的角錐代表相機位置和角度，可以看得出來是有很大的error。

opencv result:



1-3

我用投影片和助教說的方式來算pose error，最後再算median，函式和結果如下：

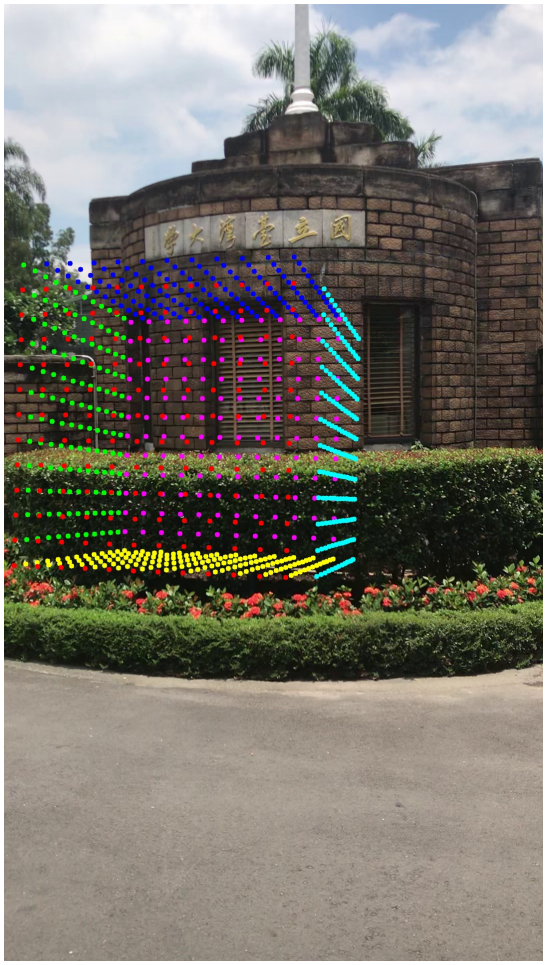
```
def differences(rotq,tvec,rotq_gt,tvec_gt):
    d_t=np.linalg.norm(tvec-tvec_gt,2)
    nor_rotq=rotq/np.linalg.norm(rotq)
    nor_rotq_gt=rotq_gt/np.linalg.norm(rotq_gt)
    dif_r=np.clip(np.sum(nor_rotq*nor_rotq_gt),0,1)
    d_r=np.degrees(np.arccos(2*dif_r*dif_r-1))

    return d_r, d_t
```

pose error	4.8872
rotation error	141.4336

Problem 2

2-1

我先把立方體上的點的位置標出來，然後做排序，從遠而近的畫圓（當然也要先算出3D點在圖片上的位置），這裡我用了cv2.circle來幫忙畫圓。函式和結果圖如下：



```python
def make_points(cube_vertices,index,color):
    ratio=12
    points=[]
    o=cube_vertices[index[0]]
    v1=cube_vertices[index[1]]-cube_vertices[index[0]]
    v2=cube_vertices[index[2]]-cube_vertices[index[0]]
    for i in range(ratio):
        for j in range(ratio):
            point=o+(i/ratio)*v1+(j/ratio)*v2
            point=point.tolist()
            point.append(color)
            points.append(point)

    return points
```

```python
def draw_points(rimg, transform_matrix, position, points_pool):
    position = np.array(position)
    ordered_points = sorted(points_pool, key=lambda point: np.linalg.norm(
        np.array(point[:3])-position, 2), reverse=True)

    for point in ordered_points:
        color = point[3]
        p = deepcopy(point)
        p[3] = 1
        pos = transform_matrix@np.array(p)
        pos /= pos[2]
        # print(pos)
        rimg = draw_one_point(rimg, pos[:2], color)

    return rimg
```