Northeastern University
EECE 5550 Mobile Robotics Lab 5 submission

Yash Mewada

Dec 9,2022

1 Solution 1 - Camera Pose Estimation

1.1 Find the derivative map of the camera projection to the point as a function of camera pose

Given the point $C_p \in R^3$ the coordinates of a point p as measured in the camera's body-center coordinate frame. Representing this point in a vector form as the given map π. This can be written as...

$$C_p = \begin{pmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} \\ \frac{y}{\sqrt{x^2+y^2+z^2}} \\ \frac{z}{\sqrt{x^2+y^2+z^2}} \end{pmatrix} = \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} \tag{1}$$

Where $\sqrt{x^2+y^2+z^2}$ is the $\|C_p\|$ and this being in S^2 i.e unit circle will always lead to 1. In order to find the derivative of the map we have to take the Jacobian of (1).

$$\mathbf{d}\pi = \begin{pmatrix} \frac{\partial X'}{\partial x} & \frac{\partial X'}{\partial y} & \frac{\partial X'}{\partial z} \\ \frac{\partial Y'}{\partial x} & \frac{\partial Y'}{\partial y} & \frac{\partial Y'}{\partial z} \\ \frac{\partial Z'}{\partial x} & \frac{\partial Z'}{\partial y} & \frac{\partial Z'}{\partial z} \end{pmatrix} \tag{2}$$

$$\frac{\partial X'}{\partial x} = \frac{1}{\sqrt{x^2+y^2+z^2}} - \frac{x^2}{(x^2+y^2+z^2)^{3/2}}$$

$$\frac{\partial X'}{\partial y} = -\frac{x\,y}{(x^2+y^2+z^2)^{3/2}}$$

$$\frac{\partial X'}{\partial z} = -\frac{x\,z}{(x^2+y^2+z^2)^{3/2}}$$

Using the above partial differentiation the derivative map can be written as below.

$$\mathbf{d}\pi = \begin{pmatrix} \frac{1}{\sqrt{x^2+y^2+z^2}} - \frac{x^2}{(x^2+y^2+z^2)^{3/2}} & -\frac{x\,y}{(x^2+y^2+z^2)^{3/2}} & -\frac{x\,z}{(x^2+y^2+z^2)^{3/2}} \\ -\frac{x\,y}{(x^2+y^2+z^2)^{3/2}} & \frac{1}{\sqrt{x^2+y^2+z^2}} - \frac{y^2}{(x^2+y^2+z^2)^{3/2}} & -\frac{y\,z}{(x^2+y^2+z^2)^{3/2}} \\ -\frac{x\,z}{(x^2+y^2+z^2)^{3/2}} & -\frac{y\,z}{(x^2+y^2+z^2)^{3/2}} & \frac{1}{\sqrt{x^2+y^2+z^2}} - \frac{z^2}{(x^2+y^2+z^2)^{3/2}} \end{pmatrix} \tag{3}$$

If we look closely and simplify this equation we can get a somewhat below form...

$$\mathbf{d}\pi = \frac{1}{\|C_p\|}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \frac{1}{\|C_p\|^3}\begin{pmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{pmatrix}$$

$$\mathbf{d}\pi = \frac{I_3}{\|C_p\|} - \frac{C_p.C_p^T}{\|C_p\|^3} \tag{4}$$

The reason $C_p.C_p^T$ has a transpose is due to the fact that C_p is a 1x3 vector and vector multiplication, in this case, will not exist.

1.2 Derive an expression for C_p, the coordinates of point p expressed in the camera's body-centric coordinate frame

We saw in the above section that the derivative map can be written in form of (4). Now we are given another coordinate in the world frame as W_p which can be represented in the world frame as X_{cw}.

$$X_{cw} = \begin{pmatrix} R_{cw} & t_{cw} \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} R_{cw} & t_{cw} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

$$X_{cp} = \begin{pmatrix} R_{cw} & t_{cw} \\ 0 & 1 \end{pmatrix} W_p \tag{5}$$

Where in the above equation W_p is the point in the world frame and $X_{cp} = (R_{cw}W_p + t_{cw})$ is the point from the world frame seen from the camera body-centric frame.

1.3 Compute the derivative of point p's coordinates X_{cp} in the camera's frame as a function of camera pose X_{cw}

We found in the above solution how the points from the world coordinate frame can be expressed in form of the camera's body-centric frame. Using the Kronecker product rule of identity we can compute the derivative as...

$$\frac{\partial X_{cp}}{\partial X_{cw}} = \frac{\partial (R_{cw}W_p + t_{cw})}{\partial X_{cw}}$$

Where W_p is the world point coordinates we derived in (5). Further solving this equation as...

$$\frac{\partial X_{cp}}{\partial X_{cw}} = [\frac{\partial X_{cp}}{\partial R_{cw}} \quad \frac{\partial X_{cp}}{\partial t_{cw}}]$$

$$\frac{\partial X_{cp}}{\partial t_{cw}} = 0 + \frac{\partial t_{cw}}{\partial t_{cw}} = I_3$$

Using the identity rule...

$$R_{cw}W_p = (W_p^T \otimes I_3) vec(R_{cw})$$

$$\frac{\partial [(W_p^T \otimes I_3) vec(R_{cw}) + t_{cw}]}{\partial R_{cw}} = (W_p^T \otimes I_3) \tag{6}$$

$$(W_p^T \otimes I_3) = \begin{pmatrix} x_w & 0 & 0 & y_w & 0 & 0 & z_w & 0 & 0 \\ 0 & x_w & 0 & 0 & y_w & 0 & 0 & z_w & 0 \\ 0 & 0 & x_w & 0 & 0 & y_w & 0 & 0 & z_w \end{pmatrix}$$

$$\frac{\partial X_{cp}}{\partial X_{cw}} = [(W_p^T \otimes I_3) \quad I_3]$$

$$W_p^T = \begin{pmatrix} x_w & y_w & z_w \end{pmatrix}$$

1.4 Write down an expression for ψ

As ψ is the map that takes the point coordinates from the world frame and camera frame and returns the image of that point in the camera, this map can be expressed as...

$$\psi = \pi(X_{cw}W_p)$$

$$\psi = \pi(R_{cw}W_p + t_{cw}) \tag{7}$$

1.5 What is the derivative of l_k with respect to u

From the given information, and using the derivatives from previous solutions we can represent it as ...

$$l_k = \arccos(u^\top \cdot uk)^2$$

$$\frac{\partial l_k}{\partial u} = \frac{-2}{\sqrt{(1-(u^\top \cdot uk)^2)}} \cdot \arccos(u^\top \cdot uk) \cdot uk \tag{8}$$

1.6 What is the derivative of L with respect to X_{cw}

Given $L(X_{cw}) = \Sigma d_{S^2}(U_k, U_k')^2$...

$$\frac{\partial L(X_{cw})}{\partial X_{cw}} = [\frac{\partial L(X_{cw})}{\partial R_{cw}} \quad \frac{\partial L(X_{cw})}{\partial t_{cw}}]$$

$$\frac{\partial \Sigma \ \ l_k(u_k)}{\partial t_{cw}} = \Sigma \frac{\partial l_k(u_k)}{\partial t_{cw}}$$

Where Σ is the summation from 1 to m.

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial u}{\partial t_{cw}}$$

Multiplying and diving it by ∂u.

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial t_{cw}}$$

Here $\partial u = \partial \pi(c_p) = \partial \pi$ due to the fact that the change in a landmark position point projected on the image plane is equal to the change in position of the point $c_p \in R^3$ projected on the image plane.
Again multiplying and diving by $\partial(c_p)$ we can rewrite the equation again as...

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} \frac{\partial c_p}{\partial t_{cw}} \tag{9}$$

But $\frac{\partial c_p}{\partial t_{cw}} = I_3$, from (6) hence again rewriting this equation. So,

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} I_3$$

$$\frac{\partial L(X_{cw})}{\partial R_{cw}} = \Sigma \frac{\partial l_k(u_k)}{\partial R_{cw}}$$

Repeating the same process as done above we can rewrite this equation as...

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} \frac{\partial c_p}{\partial R_{cw}}$$

We know that $\frac{\partial c_p}{\partial R_{cw}}$ is equal to (6), rewriting...

$$\Sigma \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} (W_p^T \otimes I_3) \tag{10}$$

Hence (9) and (10) we can in a matrix we get the derivative of L with respect X_{cw}. Also $dl_k = dl_k(u_k)$

$$\frac{\partial L(X_{cw})}{\partial X_{cw}} = \Sigma [\frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} (W_p^T \otimes I_3) \quad \frac{\partial l_k}{\partial u} \frac{\partial \pi(c_p)}{\partial c_p} I_3] \tag{11}$$

1.7 Manifold optimization problem using Manopt python.

Using the example code and the cost and product manifold logics we derived from the above solutions, below is the implementation of this in a simple python code which optimizes i.e minimizes the distance between the assumed camera pose and the real camera pose.

```
import autograd.numpy as np
import pymanopt
import pymanopt.manifolds
import pymanopt.optimizers


dim = 3
manifoldr3 = pymanopt.manifolds.Euclidean(dim,1)
SO3 = pymanopt.manifolds.SpecialOrthogonalGroup(dim)
SE2 = pymanopt.manifolds.product.Product([SO3,manifoldr3])
wp = np.array([[1,0,0,0],
               [0,1,0,0],
               [0,0,1,0]])
#Position of Landmarks in image of camera
uk = np.array([[0.866,-0.945,-0.189,-0.289],
               [0.289,0.189,-0.567,-0.866],
               [-0.408,-0.267,0.802,-0.408]])
@pymanopt.function.autograd(SE2)
def cost(rot,tr):
    s = 0
    for i in range(4):
        cp = np.dot(rot,wp[:,[i]]) + tr
        cp = cp/np.linalg.norm(cp)
        cp = cp.T @ uk[:,[i]]
        temp = cp[0,0]
        if (temp ¿= 1.0):
            s += np.arccos(1.0)**2
        elif (temp ¡= -1.0):
            s += np.arccos(-1.0)**2
        else:
            s += np.arccos(temp)**2
    return s

problem = pymanopt.Problem(SE2, cost)

optimizer = pymanopt.optimizers.SteepestDescent()
result = optimizer.run(problem)

print("Rotation:", result.point[0])
print("Translation:", result.point[1])
```

Listing 1: Example pymanopt

Below is the output of the number of iterations it ran to optimize the pose of the camera along with the actual pose of the camera.

```
Optimizing...
Iteration    Cost                      Gradient norm
---------    ----------------------    --------------
   1         +1.1966442695837371e+01    4.86509882e+00
   2         +6.6312052149468066e+00    7.62305742e+00
   3         +1.3664994956629473e+00    4.43256039e+00
   4         +1.0045103871865182e+00    3.50516590e+00
   5         +5.1998252406565992e-01    1.94605270e+00
   6         +2.3427401534678116e-01    1.10595815e+00
   7         +1.5102458135219349e-01    1.51311745e+00
   8         +1.1639456176866425e-01    1.45785186e+00
   9         +3.4757129007832434e-02    3.91371284e-01
  10         +1.9301087751863545e-02    5.48344092e-01
  11         +8.7641478109826802e-03    2.77385086e-01
  12         +5.4431687166723195e-03    2.35072693e-01
  13         +3.1059051236712173e-03    1.63635488e-01
  14         +2.3678901708911815e-03    1.98087068e-01
  15         +7.8802188026617296e-04    5.99118459e-02
  16         +3.9910908602930903e-04    7.54397887e-02
  17         +2.5102837993958488e-04    5.66283473e-02
  18         +1.5405059124141838e-04    1.83210750e-02
  30         +1.2274285818722247e-04    3.50586015e-03
  31         +1.2261135910363692e-04    3.43804116e-03
  32         +1.2261048364917038e-04    1.34027025e-02
  33         +1.2260974702021552e-04    3.43750512e-03
  34         +1.2260901043049098e-04    3.43713535e-03
  35         +1.2260882628589094e-04    3.43704292e-03
  36         +1.2260880326784370e-04    3.43703137e-03
```

```
  37         +1.2260880039056003e-04    3.43702992e-03
Terminated - min step˙size reached after 37 iterations, 0.13 seconds.

Rotation: [[ 0.70809897 -0.70611179 -0.00140785]
            [ 0.70610204  0.70807516  0.0070346 ]
            [-0.00397035 -0.00597528  0.99997427]]
Translation: [[-0.17565098]
            [-0.52795075]
            [-0.2476308 ]]
```

Listing 2: Optimisation output

The above code describes how the distance between the actual points of the landmark on the image plane S^2 and the points the camera sees on the same plane is optimized by minimizing it using the gradient descent function. Note that the distance between two points on a sphere is $\arccos p1.p2$, where p1 is the point that the camera sees on its plane and p2 is the point that the actual landmarks appear on that image plane, here sphere.