EECE 5550 Mobile Robotics - Section2 -Lab #1

Zhexin Xu , xu.zhex@northeastern.edu

October 1, 2023

Abstract

This is the solution of HW1. The code is implemented using c++ and mainly based on Eigen and Sophus libraries. Pangolin and matplotlib are also used for visualization. All the code can be found in:https://github.com/zhexin1904/EECE5550

1 Question1: Object pose estimation

Question1 required to get the transformation matrix $T_{SO} \in \mathrm{SE}(3)$ between the frame O and frame S, given the coordinates of p in these two frames. Firstly, we can get the homogeneous representation of the coordinates:

$$P_O = \begin{pmatrix} 2 & 0 & -1 & -1 \\ 3 & 0 & -2 & 0 \\ -3 & -3 & 2 & -2 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{1}$$

$$P_S = \begin{pmatrix} -1.3840 & -0.9608 & 1.3250 & -1.3140 \\ 4.5620 & 1.3110 & -2.3890 & 0.2501 \\ -0.1280 & -1.6280 & 1.7020 & -0.7620 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{2}$$

Based on the transformation relationship:

$$T_{SO} P_O = P_S \tag{3}$$

we can get the transformation matrix:

$$T_{SO} = P_S P_O^{-1} = \begin{pmatrix} 0.706812 & -0.612275 & 0.353612 & 0.100038 \\ 0.707219 & 0.612188 & -0.353681 & 0.249956 \\ 0 & 0.5 & 0.866 & 0.97 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

2 Question2: Lie algebras and left-invariant vector fields

(a) Given $v \in \mathbb{R}^n$, what is the corresponding left-translation map $L_v : \mathbb{R}^n \to \mathbb{R}^n$?

$$L_v(x) = v + x \tag{5}$$

where $x \in R^n$.

(b) What is the derivative dL_v of the map L_v you found in part (a)?

$$dL_v = \lim_{x \to 0} \frac{v + x}{x} = \frac{\partial x}{\partial x} = I \in R^{n \times n} \tag{6}$$

(c) Given a vector $\xi \in T_0(\mathbb{R}^n) \cong \mathbb{R}^n$ in $\mathbb{R}^n$'s Lie algebra, what is the left-invariant vector field V_ξ on $\mathbb{R}^n$ determined by ξ ? Interpret this result geometrically.

$$V_\xi(x) \triangleq d(L_x)_e(\xi) = I(\xi) = \xi \in T_0(\mathbb{R}^n) \tag{7}$$

(d) Given a matrix $A \in \mathrm{GL}(n)$, what is the corresponding left-translation map $L_A : \mathrm{GL}(n) \to \mathrm{GL}(n)$?

Matrix multiplication is the group operation of $\mathrm{GL}(n)$. Therefore, we can conclude that:

$$L_A(x) = Ax \tag{8}$$

(e) What is the derivative dL_A of the map L_A you found in part (d)?

$$dL_A = \lim_{x \to 0} \frac{Ax}{x} = A \in GL(n) \tag{9}$$

(f) The tangent space $T_I(\mathrm{GL}(n))$ of $\mathrm{GL}(n)$ at the identity $I \in \mathrm{GL}(n)$ is just $\mathbb{R}^{n \times n}$, the set of all $n \times n$ matrices. [2] Given a matrix $\Omega \in T_I(\mathrm{GL}(n))$, what is the left-invariant vector field V_Ω on $\mathrm{GL}(n)$ determined by Ω ?

$$V_\Omega \triangleq d(L_x)_e V_\omega(e) = d(L_x)_e(\Omega) = A\Omega \in GL(n) \tag{10}$$

3 Question 3: Motion on Lie groups

(a)Using(11), derive a formula for a curve $\gamma : [0, 1] \to G$ such that $\gamma(0) = x$ and $\gamma(1) = y$.

$$\gamma(t) \triangleq x \star \exp(t\omega) \tag{11}$$

Using $\gamma(0) = x$ and $\gamma(1) = y$, we can conclude that:

$$\omega = log(x^{-1}y) \tag{12}$$

As a result, curve $\gamma : [0, 1] \to G$ can be stated that:

$$\gamma(t) \triangleq xexp(tlog(x^{-1}y)) \tag{13}$$

(b)Using (14)and (15), specialize your result from part (a) to derive a formula for a curve γ that joins x to y in $\mathbb{R}^n$. Interpret this result geometrically.

$$\exp : \mathbb{R}^n \to \mathbb{R}^n \tag{14}$$

$$\exp(\xi) = \xi \tag{15}$$

Based on(15), we can state that:

$$\log(\xi) = \xi \tag{16}$$

Thus, it can be inferred that:

$$\gamma(t) \triangleq x \star t(x^{-1} \star y) \tag{17}$$

We know that as for the group in $\mathbb{R}^n$, group operation is vector addition. As a result, it can be stated that:

$$\gamma(t) \triangleq x + t(-x + y) \tag{18}$$

(c)Apply the formula you derived in part (a) to calculate the "midpoint" $\gamma_{\mathrm{SE}(3)}(1/2)$ on the curve $\gamma_{\mathrm{SE}(3)} : [0, 1] \to \mathrm{SE}(3)$ from X_0 to X_1.

$$(t_1, R_1) \star (t_2, R_2) = (R_1 t_2 + t_1, R_1 R_2)\,. \tag{19}$$

We can directly use the multiplication in SE3 to calculate the mid point:

$$\gamma(t) \triangleq x \star exp(0.5 * log(x^{-1} \star y) \tag{20}$$

where $\star$ is the multiplication in SE3 and $*$ is the multiplication of scalar. Here we use Eigen and Lie group library Sophus to conduct calculations, and a brief description of the code is give below:

```
1    Sophus::SE3d SE3_X0(orthogonalize(R0), t0);
2    Sophus::SE3d SE3_X1(orthogonalize(R1), t1);
3    Sophus::SE3d SE3_mid;
4    SE3_mid=SE3_X0*Sophus::SE3d::exp(0.5*((SE3_X0.inverse()*SE3_X1).log()));
```

Among them, function *orthogonalize*() is used to make sure that the rotation matrix is orthogonal, which is required by Sophus.

$$mid = \begin{pmatrix} 0.612391 & 0.0794515 & 0.786552 & 1.67103 \\ 0.353562 & 0.862364 & -0.362384 & 2.59874 \\ -0.707087 & 0.500016 & 0.500013 & 1.34425 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(d)Using the formula that you derived in part (a), compute the "midpoint" $\gamma_P(1/2)$ of the curve $\gamma_P : [0, 1] \to P$ from X_0 to X_1 in P.

$$(t_1, R_1) \star_P (t_2, R_2) = (t_1 + t_2, R_1 R_2)\,. \tag{21}$$

Different from the operation defined in (19), here we have to conduct operation seperatly according to the definition in (21). We also use Sophus to conduct calculations related to Lie group, and a brief description of the code is give below:

```
1     Sophus::SO3d SO3_X0(orthogonalize(R0)), SO3_X1(orthogonalize(R1));
2     Sophus::SE3d CurveP_mid;
3     CurveP_mid.so3() = SO3_X0 * Sophus::SO3d::exp(0.5 * ((SO3_X0).inverse()*
SO3_X1).log());
4     CurveP_mid.translation() = t0 + (0.5 * (-t0 +  t1));
```

Mid point in $\gamma_P(1/2)$ of the curve γ_P is still defined as an element in SE3, but we use self-defined operation as formula (21), which can be interpreted as to update the rotation part $\in SO(3)$ and the translation part $\in R^3$ separately. 0.612391 0.0794515 0.786552 1.5 0.353562 0.862364 -0.362384 2.5 -0.707087 0.500016 0.500013 1.5 0 0 0 1

$$mid = \begin{pmatrix} 0.612391 & 0.0794515 & 0.786552 & 1.5 \\ 0.353562 & 0.862364 & -0.362384 & 2.5 \\ -0.707087 & 0.500016 & 0.500013 & 1.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(e)Plot the transnational components of the curves $\gamma_{\mathrm{SE}(3)}$ and γ_P from parts (c) and (d) over two intervals: (i) $t \in [0, 1]$ and (ii) $t \in [0, 30]$. Describe these curves qualitatively.

Here we can still use the formulation in (c) and (d) to compute the trajectory, and a brief description of the algorithm is give below:

```
1 SE3_mid=SE3_X0*Sophus::SE3d::exp(0.5*((SE3_X0.inverse()*SE3_X1).log()));
2 CurveP_add.so3()=SO3_X0*Sophus::SO3d::exp(dt*((SO3_X0).inverse()*SO3_X1).log());
3 CurveP_add.translation() = t0 + (dt * (-t0 +  t1));
4 dt = dt + 0.05;
```

As for the qualitative description, we simply use matplotlib to visualize these trajectory firstly.

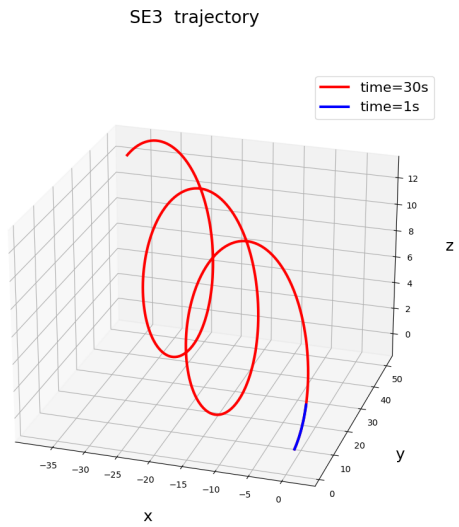

Figure 1: Trajectory of SE3.

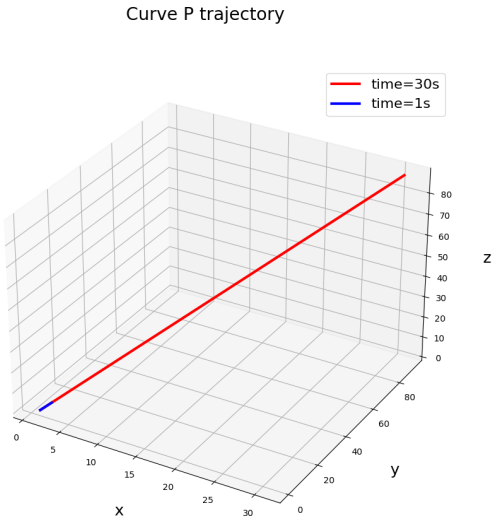

Figure 2: Trajectory of Curve p.

However, the orientation can not be visualized intuitively, especially for the operation of curve p . We use Pangolin to realized the visualization of 6 DOF(blue represent Z axis, red represent X axis and Green represent Y axis(Pangolin use right-handed coordinates)):

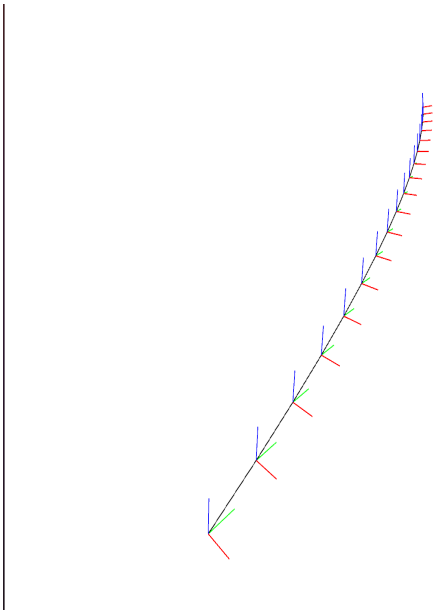

Figure 3: Trajectory of SE3 after 5 S.

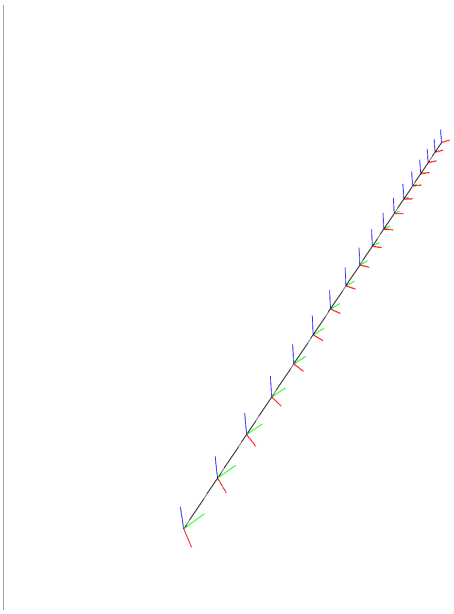

Figure 4: Trajectory of Curve P after 5 S.

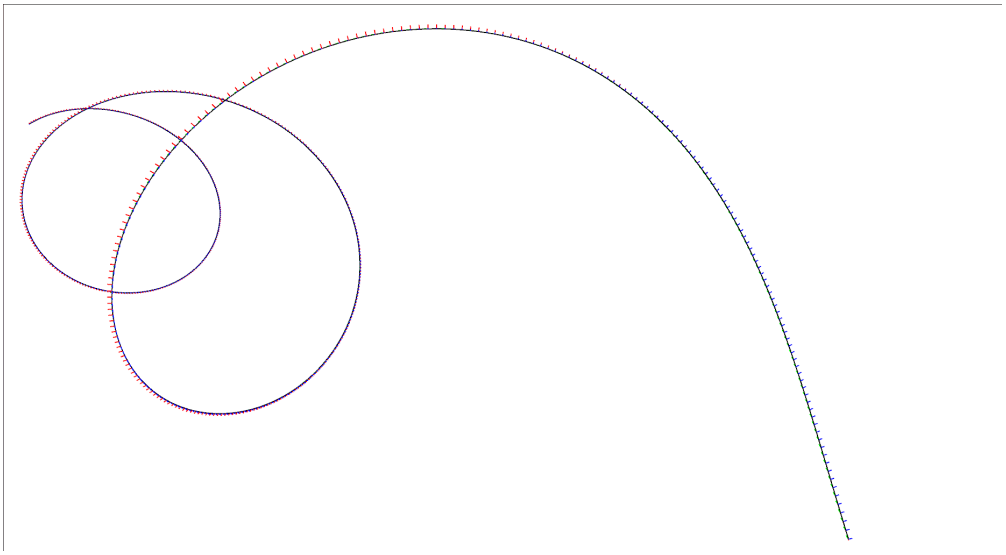

Figure 5: Trajectory of SE3 after 30 S.

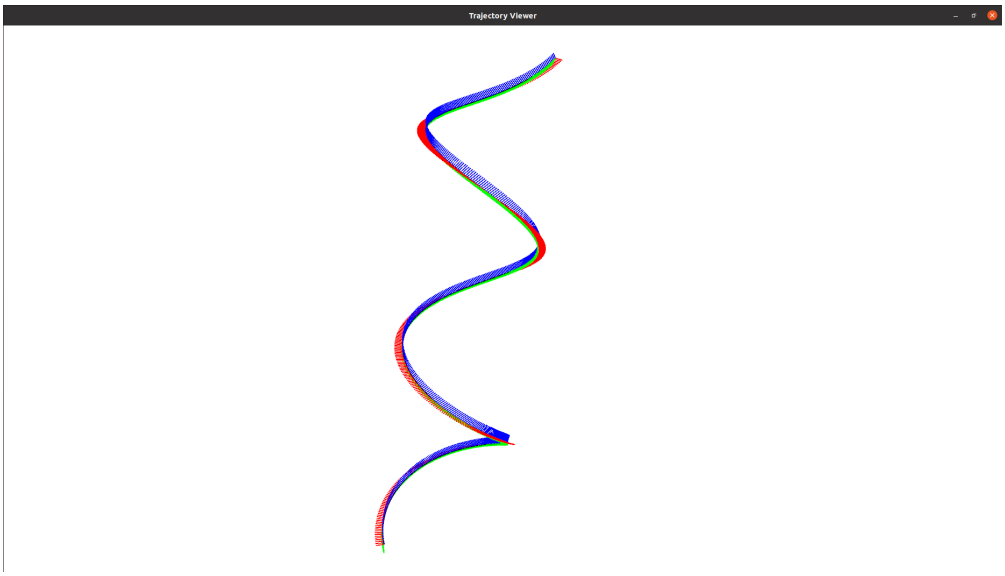

Figure 6: Trajectory of SE3 after 30 S(amplify the coordinate axis, for better visualization).

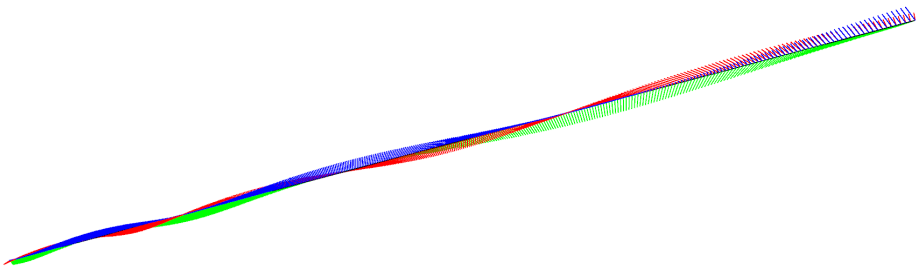

Figure 7: Trajectory of Curve P after 30 S(amplify the coordinate axis, for better visualization).

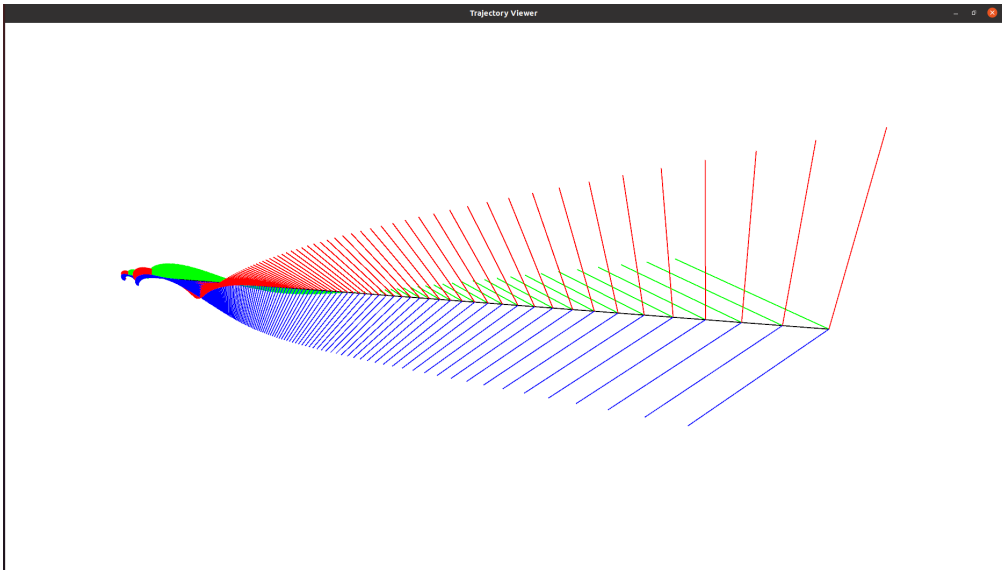

Figure 8: Trajectory of Curve P after 30 S(amplify the coordinate axis, for better visualization).

Intuitively, trajectory of SE3 is the same as we think about affine transformation. However, because of the definition of the operation in Curve P, the trajectory seems like a camera moves along a straight line, and Continuously rotate it's frame at the same time.

4 Question 4: Implementation of a differential drive robot

Consider the kinematic model of a differential drive robot, for which the wheel radius is 6 cm and the distance between the wheels are 30 cm. Suppose that the left and right wheels rotate 8 rad/s and 10rad/s, respectively. Assuming that the initial configuration is $(x_0, y_0, \theta_0) = (0, 0, 0)$,

(a) What is the pose and orientation of the robot (in terms of affine transformation matrix form) after 5 s ?

This problem can be solved as a kinematic problem in SE3. Typically, we can use the equations below to formulate the kinematics(We do not have acceleration in this case):

$$\begin{aligned} \mathbf{v}_k^w &= \mathbf{R}_b^w \mathbf{v}_k^b \\ \mathbf{p}_{b_{k+1}}^w &= \mathbf{p}_{b_k}^w + \mathbf{v}_k^w \Delta t + \frac{1}{2}\mathbf{a}^w \Delta t^2 \\ \mathbf{a}_k^b &= \mathbf{R}_{b_k}^{w\top} \mathbf{a}^w \\ \mathbf{v}_{k+1}^b &= \mathbf{v}_k^b + \mathbf{a}_k^b \Delta t \\ \mathbf{R}_{b_{k+1}}^w &= \mathbf{R}_{b_k}^w \exp(\boldsymbol{\omega} \Delta t) \end{aligned} \tag{22}$$

Usually, there are two ways can be used to update the rotation part, which consists of directly updating in SO3, and updating in quaternion.

Updating in SO3 is similar as Question3 (c). Using Sophus library to representing SE3, we update the translation part and orientation part seperatly. Discreet integration time dt is set to 0.05s.

```
V_w = T_wb.so3() * Car.V_b; //convert speed to world frame
P_w = P_w + V_w * dt; //update position(translation)
T_wb.translation() = P_w;
T_wb.so3() = T_wb.so3() * Sophus::SO3d::exp(Car.W_b * dt); // update orientation
dt = dt + 0.05;
```

The result of trajectory which updates the rotation part using SO3 after 5s is :

$$P_{5S}(SO3) = \begin{pmatrix} -0.434188 & -0.900822 & 0 & 1.23547 \\ 0.900822 & -0.434188 & 0 & 1.92399 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The key difference between the SO3 and quaternion in updating the rotation is the angel. We can give these two key formula:

$$\dot{\boldsymbol{R}} = \boldsymbol{R}\boldsymbol{\omega}^{\wedge} \tag{23}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q}\boldsymbol{\omega} \tag{24}$$

The updating form in kinematics can be stated as:

$$\boldsymbol{R}_{t+1} = \boldsymbol{R}_t \exp(\boldsymbol{\omega}\Delta t) \tag{25}$$

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t \exp(\frac{1}{2}[0, \boldsymbol{\omega}]^{\top}\Delta t) \tag{26}$$

In other words, the angular velocity, or the angle(after integration) used in updating quaternion is just the half of that in updating SO3. Detailed derivation can be found in State Esitimation for Robotics [1].

```
V_w = T_wb.so3() * Car.V_b; //convert speed to world frame
P_w = P_w + V_w * dt; //update position(translation)
T_wb.translation() = P_w;
Eigen::Quaterniond q;
q = T_wb.unit_quaternion() * Eigen::Quaterniond(1, 0.5*Car.W_b(0)*dt, 0.5*Car.W_

q.normalize();
T_wb.so3() = Sophus::SO3d(q);
dt = dt + 0.05;
```

The result of trajectory which updates the rotation part using quaternion after 5s is :

$$P_{5S}(quatrnion) = \begin{pmatrix} -0.434248 & -0.900793 & 0 & 1.23539 \\ 0.900793 & -0.434248 & 0 & 1.92401 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Compared with the results, we can found that these two methods in updating rotation achieve almost the same accuracy. This maybe related to the setting of discrete integration time, which is very small in our experiment.

(b) Plot the resulting trajectory.

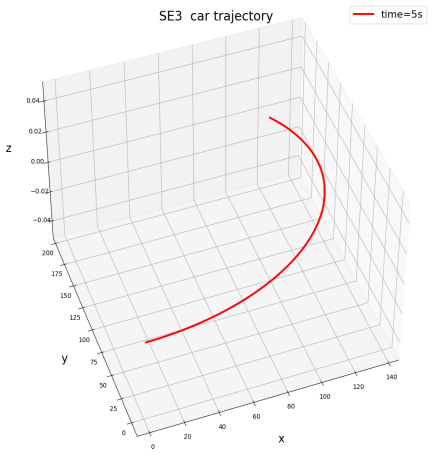

Figure 9: Trajectory of differential robot visualized in matplotlib.

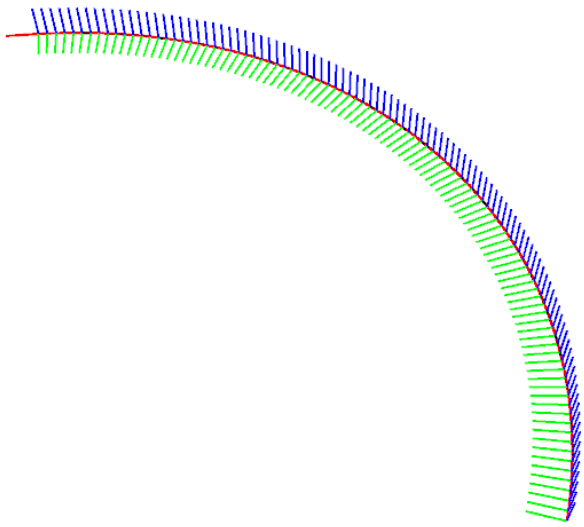

Figure 10: Trajectory of differential robot visualized in Pangolin.

(c) Copy/paste your code to your assignment.

Complete implementation of the code can be found in:https://github.com/zhexin1904/EECE5550

Using SO3 to update the rotation part.

```
1 V_w = T_wb.so3() * Car.V_b; //convert speed to world frame
```

```
P_w = P_w + V_w * dt; //update position(translation)
T_wb.translation() = P_w;
Eigen::Quaterniond q;
q = T_wb.unit_quaternion() * Eigen::Quaterniond(1, 0.5*Car.W_b(0)*dt, 0.5*Car.W_
q.normalize();
T_wb.so3() = Sophus::SO3d(q);
dt = dt + 0.05;
```

Using quaternion to update the rotation part.

```
V_w = T_wb.so3() * Car.V_b; //convert speed to world frame
P_w = P_w + V_w * dt; //update position(translation)
T_wb.translation() = P_w;
T_wb.so3() = T_wb.so3() * Sophus::SO3d::exp(Car.W_b * dt);
dt = dt + 0.05;
```

References

[1] Barfoot, Timothy D. *State estimation for robotics*, (Cambridge University Press, 2017).