AA 274A: Principles of Robot Autonomy I
Problem Set 4 - Group Part

Name: Alberto Guiggiani, Avrum Noor, Josie Oetjen, Karthik Pythireddi

SUID: alberto6, avrum, josieoet, karthik09

November 28, 2022

Problem 1

(i) We use the rule for integration by substitution, which states that

$$\int_a^b f(g(x))g'(x)\mathrm{d}x = \int_{g(a)}^{g(b)} f(g(x))\mathrm{d}g(x).$$

Defining $x = t$, $g(x) = \theta$, and $g'(x) = \omega$, we can then write g_x as

$$\begin{aligned} g_x &= x_{t-1} + \int_{t-1}^{t} V(t)\cos(\theta(t))\mathrm{d}t \\ &= x_{t-1} + V_t \int_{t-1}^{t} \cos(\theta(t))\frac{\omega_t}{\omega_t}\mathrm{d}t \\ &= x_{t-1} + \frac{V_t}{\omega_t}\int_{\theta_{t-1}}^{\theta_{t-1}+\omega\mathrm{d}t} \cos(\theta)\mathrm{d}\theta \\ &= x_{t-1} + \frac{V_t}{\omega_t}\left(\sin(\theta_{t-1}+\omega\mathrm{d}t) - \sin(\theta_{t-1})\right), \end{aligned}$$

where we added the term $\frac{\omega_t}{\omega_t}$ in the second line, which is allowed because it cancels out, to transfer the theorem to our problem.

Similarly, for g_y we obtain

$$\begin{aligned} g_y &= y_{t-1} + \int_{t-1}^{t} V(t)\sin(\theta(t))\mathrm{d}t \\ &= y_{t-1} + V_t \int_{t-1}^{t} \sin(\theta(t))\frac{\omega_t}{\omega_t}\mathrm{d}t \\ &= y_{t-1} + \frac{V_t}{\omega_t}\int_{\theta_{t-1}}^{\theta_t+\omega\mathrm{d}t} \sin(\theta)\mathrm{d}\theta \\ &= y_{t-1} + \frac{V_t}{\omega_t}\left(-\cos(\theta_{t-1}+\omega\mathrm{d}t) + \cos(\theta_{t-1})\right) \\ &= y_{t-1} + \frac{V_t}{\omega_t}\left(\cos(\theta_{t-1}) - \cos(\theta_{t-1}+\omega\mathrm{d}t)\right), \end{aligned}$$

and for g_θ

$$\begin{aligned} g_\theta &= \theta_{t-1} + \int_{t-1}^{t} \omega(t)\mathrm{d}t \\ &= \theta_{t-1} + \omega(t)\mathrm{d}t. \end{aligned}$$

Now, we define the Jacobians where we denote $\theta_{t-1} = \theta$ to simplify notation. We have

$$\begin{aligned} G_{\vec{x}} &= \begin{bmatrix} \frac{\partial g_x}{\partial x} & \frac{\partial g_x}{\partial y} & \frac{\partial g_x}{\partial \theta} \\ \frac{\partial g_y}{\partial x} & \frac{\partial g_y}{\partial y} & \frac{\partial g_y}{\partial \theta} \\ \frac{\partial g_\theta}{\partial x} & \frac{\partial g_\theta}{\partial y} & \frac{\partial g_\theta}{\partial \theta} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \frac{V}{\omega}(\cos(\theta+\omega\mathrm{d}t) - \cos(\theta)) \\ 0 & 1 & \frac{V}{\omega}(\sin(\theta+\omega\mathrm{d}t) - \sin(\theta)) \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

$$G_{\vec{u}} = \begin{bmatrix} \frac{\partial g_x}{\partial V} & \frac{\partial g_x}{\partial \omega} \\ \frac{\partial g_y}{\partial V} & \frac{\partial g_y}{\partial \omega} \\ \frac{\partial g_\theta}{\partial V} & \frac{\partial g_\theta}{\partial \omega} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\sin(\theta+\omega \mathrm{d}t)-\sin(\theta)}{\omega} & -\frac{V}{\omega^2}(\sin(\theta+\omega \mathrm{d}t)-\sin(\theta)) + \frac{V}{\omega}\cos(\theta+\omega \mathrm{d}t)\mathrm{d}t \\ \frac{\cos(\theta)-\cos(\theta+\omega \mathrm{d}\theta)}{\omega} & -\frac{V}{\omega^2}(\cos(\theta)-\cos(\theta+\omega \mathrm{d}t)) + \frac{V}{\omega}\sin(\theta+\omega \mathrm{d}t)\mathrm{d}t \\ 0 & \mathrm{d}t \end{bmatrix}.$$

Now, to compute the derivatives when $|\omega|$ is close to zero, we take the $\lim_{\omega \to 0} \omega$ and use the L'Hôpital rule where we take the derivative w.r.t θ:

$$\lim_{\omega \to 0} g_x = V \mathrm{d}t \cos(\theta) + x_{t-1},$$
$$\lim_{\omega \to 0} g_y = V \mathrm{d}t \sin(\theta) + y_{t-1},$$
$$\lim_{\omega \to 0} G_{\vec{x}} = \begin{bmatrix} 1 & 0 & -V \mathrm{d}t \sin(\theta) \\ 0 & 1 & V \mathrm{d}t \cos(\theta) \\ 0 & 0 & 1 \end{bmatrix}$$
$$\lim_{\omega \to 0} G_{\vec{u}} = \begin{bmatrix} \mathrm{d}t \cos(\theta) & -V \mathrm{d}t^2 \sin(\theta) \\ \mathrm{d}t \sin(\theta) & V \mathrm{d}t^2 \cos(\theta) \\ 0 & \mathrm{d}t \end{bmatrix}.$$

(ii) See `compute_dynamics()` in `turtlebot_model.py`.

(iii) See `transition_update()` in `turtlebot_model.py`.

(iv) See `transform_line_to_scanner_frame()` in `ekf.py` and `compute_predicted_measurements()` in `ekf.py`.

We show the annotated Figure in Figure 1.

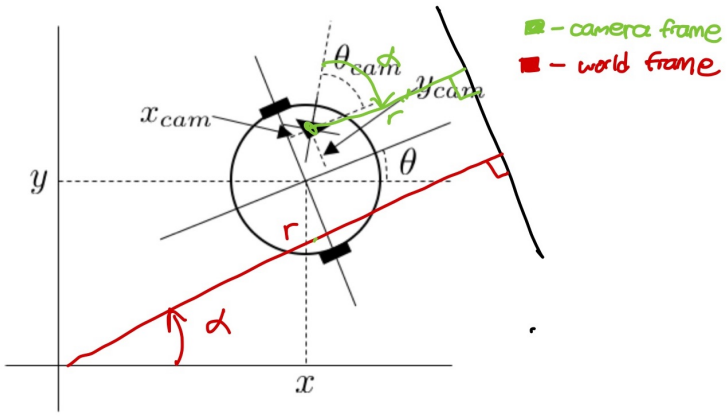

Figure 1: Annotated Figure.
The figure shows the annotated Figure with a line and its parameters (α, r) in the world and camera frame.

(v) We show how we would implement `compute_innovations()` in Algorithm 1.

Algorithm 1: Compute Innovations Algorithm

```
Initialize v_list, Q_list, H_list to empty
for each observed line do
    Initialize innovation vector to none
    Initialize optimal distance to a minimum threshold
    for each predicted line do
        Compute the Mahalanobis distance
        if Distance is smaller than optimal distance then
            Set optimum to found
            Update innovation vector to the one associated with the current predicted line
    if Optimum found then
        Append innovation, covariance matrix, Jacobian to v_list, Q_list, H_list
```

(vi) See `measurement_model()` in `ekf.py`.

(vii) See `measurement_update()` in `ekf.py`.

(viii) Without adjusting the uncertainty in the system, our robots starts nice and after some time, it slightly deviates from the ground truth. In a next step, we adjust the parameters in NoiseParams as follows:

`Sigma0`: $0.02 \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$,
`var_theta`: 0.05,
`var_rho`: 0.1.

Now, our robot performs worse and we deviate from the truth quicker and stronger.

(ix) We show the initial state in Figure 2, when the TurtleBot has moved far from the initial state in Figure 3, and when the state estimates diverge in Figure 4.
We observe that when driving towards a wall at constant speed, we observe an increase in divergence between the EKF and the open loop. Similar happens when we we drive parallel to a wall. Spinning in place is one of the trickiest manoeuvres. Then, we see a big divergence in the angles.

Problem 2

(i) For the code, see `transition_model()` in `ekf.py`.
The dimension of the state vector is $(3 + 2J, 1)$, where J is the number of map elements.

(ii) See `measurement_model()`, `compute_innovations()`, and `compute_predicted_measurements()` for the `EkfSlam` class in `ekf.py`.

(iii) We show the initial state in Figure 5, when the TurtleBot has moved far from the initial state and the map estimate has changed in Figure 6, and when the map estimates have converged in Figure 7.
Here, we observe that the EKF and the ground truth do not diverge a lot. We tried spinning, driving towards or parallel to a wall, increasing and decreasing the speed. However, we observe a small divergence when driving with higher speed, because the estimation takes some time to compute. When slowing down, the estimate tends to converge again. The open loop still diverges under all motions.

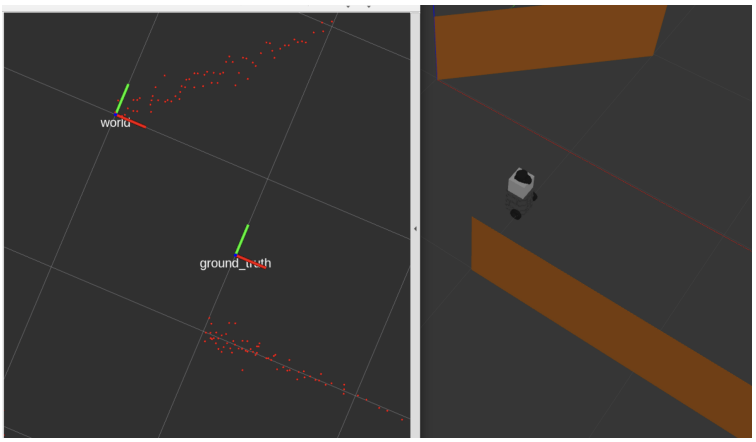

Figure 2: Initial state.
The figure shows the initial state of our robot.

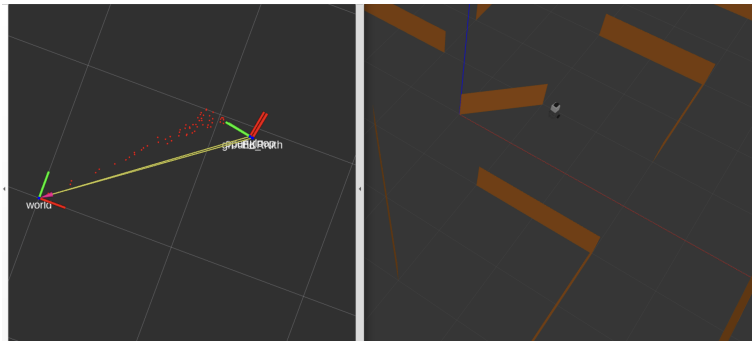

Figure 3: Far from initial state.
The figure shows when the TurtleBot has moved far from the initial state.

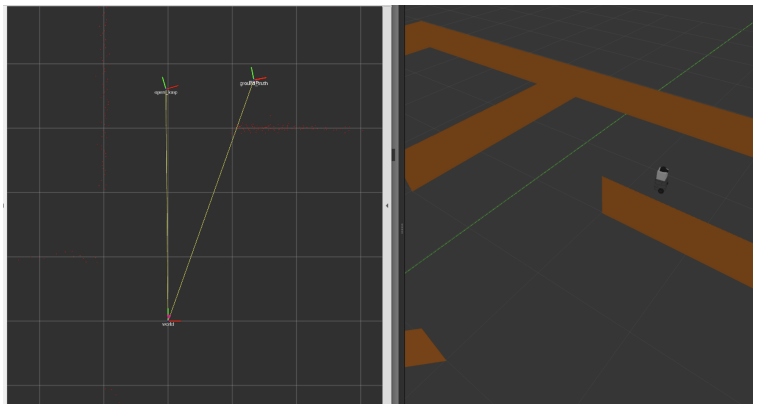

Figure 4: Divergence.
The figure shows when the TurtleBot when the state estimates diverge.

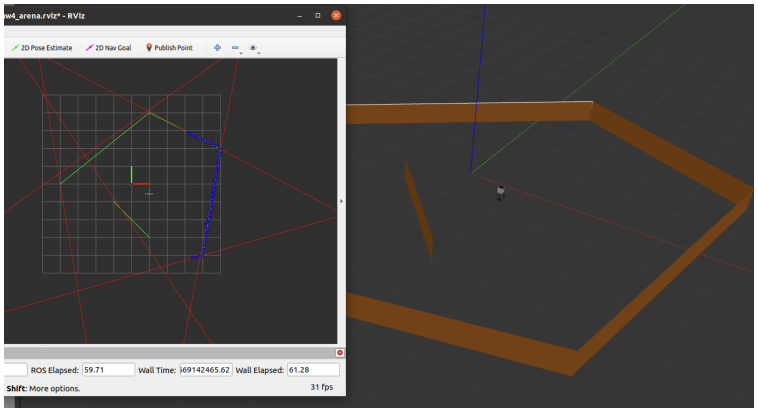

Figure 5: SLAM Initial state.
The figure shows the initial state of our robot.

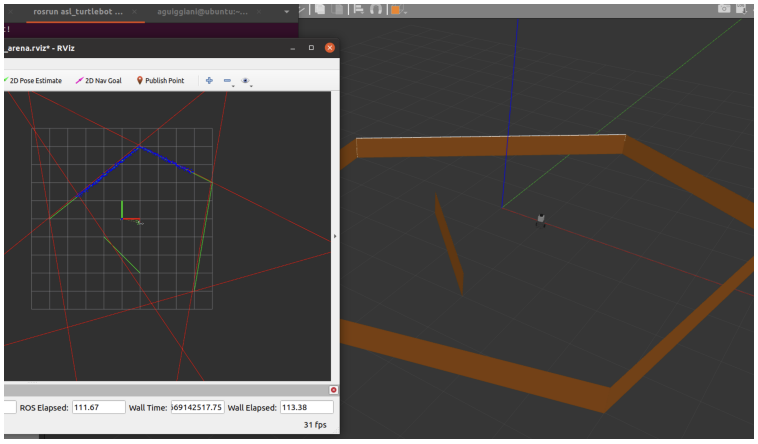

Figure 6: SLAM far from initial state.
The figure shows when the TurtleBot has moved far from the initial state and the map estimate has changed.

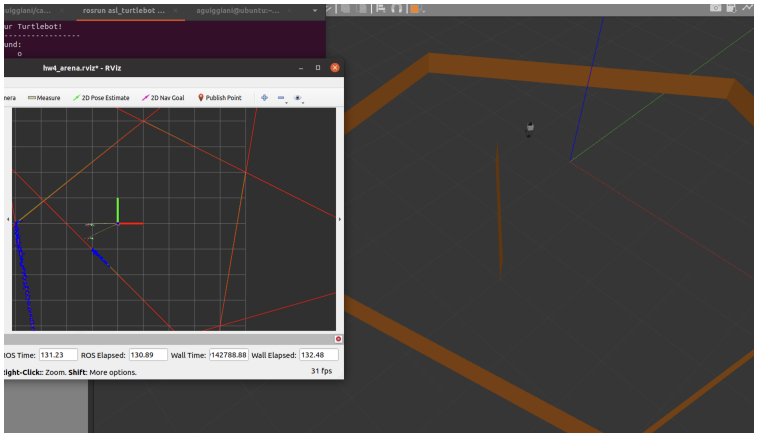

Figure 7: SLAM Divergence.
The figure shows when the TurtleBot when the map estimates have converged.