AA 274A: Principles of Robot Autonomy I
Problem Set 4
Group 18

Name: Abhyudit Singh Manhas (SUID: 06645995)
Name: Anish Mokkarala (SUID: 06499590)
Name: Gandharv Mahajan (SUID: 06510344)
Name: Shail Trivedi (SUID: 06501118)

Problem 1: EKF Localization

(i) The dynamics of our Turtlebot is:

$$\begin{aligned}\dot{x}(t) &= V(t)\cos(\theta(t)) \\ \dot{y}(t) &= V(t)\sin(\theta(t)) \\ \dot{\theta}(t) &= \omega(t)\end{aligned} \tag{1}$$

The continuous state variable is $\mathbf{x}(t) = \begin{bmatrix} x(t) & y(t) & \theta(t)\end{bmatrix}^T$, and the instantaneous control is $\mathbf{u}(t) = \begin{bmatrix} V(t) & \omega(t)\end{bmatrix}^T$. Also, let $\mathbf{x}(t) = \mathbf{x}_t$ and $\mathbf{x}(t-dt) = \mathbf{x}_{t-1}$. Now in equation 1, we integrate the equation for $\dot{x}(t)$ to get:

$$\int_{t-dt}^{t} \dot{x}(t)dt = x_t - x_{t-1} = \int_{t-dt}^{t} V(t)\cos(\theta(t))dt \tag{2}$$

Performing a change of variables:

$$\begin{aligned}\frac{d\theta(t)}{dt} &= \omega(t) \\ \implies dt &= \frac{d\theta(t)}{\omega(t)}\end{aligned} \tag{3}$$

Substituting 3 in 2, we get:

$$x_t - x_{t-1} = \int_{\theta_{t-1}}^{\theta_t} V(t)\cos(\theta(t))\frac{d\theta(t)}{\omega(t)} \tag{4}$$

Now assuming a zero-order hold on the control inputs, we can assume $V(t) = V$ and $\omega(t) = \omega$, i.e., we assume they are constants over the time interval of length dt and so they are taken outside the integral. Also, dropping the dependence on t for θ, we get:

$$\begin{aligned} x_t - x_{t-1} &= \frac{V}{\omega}\int_{\theta_{t-1}}^{\theta_t} \cos(\theta)d\theta \\ \implies x_t - x_{t-1} &= \frac{V}{\omega}\left[\sin(\theta)\right]\Big|_{\theta_{t-1}}^{\theta_t} \\ \implies x_t &= x_{t-1} + \frac{V}{\omega}\left(\sin(\theta_t) - \sin(\theta_{t-1})\right)\end{aligned} \tag{5}$$

Similarly, we integrate the equation for $\dot{y}(t)$ from equation 1 to get:

$$\begin{aligned}
\int_{t-dt}^{t} \dot{y}(t)dt = y_t - y_{t-1} &= \int_{t-dt}^{t} V(t)\sin(\theta(t))dt \\
\implies y_t - y_{t-1} &= \int_{\theta_{t-1}}^{\theta_t} V(t)\sin(\theta(t))\frac{d\theta(t)}{\omega(t)} \\
\implies y_t - y_{t-1} &= \frac{V}{\omega}\int_{\theta_{t-1}}^{\theta_t} \sin(\theta)d\theta \\
\implies y_t - y_{t-1} &= \frac{V}{\omega}\left[-\cos(\theta)\right]\Big|_{\theta_{t-1}}^{\theta_t} \\
\implies y_t &= y_{t-1} - \frac{V}{\omega}\left(\cos(\theta_t) - \cos(\theta_{t-1})\right) \qquad (6)
\end{aligned}$$

Integrating the equation for $\dot{\theta}(t)$ from equation 1, we get:

$$\begin{aligned}
\int_{t-dt}^{t} \dot{\theta}(t)dt = \theta_t - \theta_{t-1} &= \int_{t-dt}^{t} \omega(t)dt \\
\implies \theta_t - \theta_{t-1} &= \omega\int_{t-dt}^{t} dt \\
\implies \theta_t &= \theta_{t-1} + \omega dt \qquad (7)
\end{aligned}$$

Substituting 7 in 5 and 6, we get the transition model $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$:

$$\boxed{\begin{gathered}
x_t = x_{t-1} + \frac{V}{\omega}\left(\sin(\theta_{t-1} + \omega dt) - \sin(\theta_{t-1})\right) \\
y_t = y_{t-1} - \frac{V}{\omega}\left(\cos(\theta_{t-1} + \omega dt) - \cos(\theta_{t-1})\right) \\
\theta_t = \theta_{t-1} + \omega dt
\end{gathered}} \qquad (8)$$

The Jacobian G_x is given by:

$$G_x = \begin{bmatrix}
\frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial x_t}{\partial y_{t-1}} & \frac{\partial x_t}{\partial \theta_{t-1}} \\
\frac{\partial y_t}{\partial x_{t-1}} & \frac{\partial y_t}{\partial y_{t-1}} & \frac{\partial y_t}{\partial \theta_{t-1}} \\
\frac{\partial \theta_t}{\partial x_{t-1}} & \frac{\partial \theta_t}{\partial y_{t-1}} & \frac{\partial \theta_t}{\partial \theta_{t-1}}
\end{bmatrix}$$

$$\implies G_x = \begin{bmatrix}
1 & 0 & \frac{V}{\omega}\left(\cos(\theta_{t-1} + \omega dt) - \cos(\theta_{t-1})\right) \\
0 & 1 & \frac{V}{\omega}\left(\sin(\theta_{t-1} + \omega dt) - \sin(\theta_{t-1})\right) \\
0 & 0 & 1
\end{bmatrix} \qquad (9)$$

The Jacobian G_u is given by:

$$G_u = \begin{bmatrix} \dfrac{\partial x_t}{\partial V} & \dfrac{\partial x_t}{\partial \omega} \\ \dfrac{\partial y_t}{\partial V} & \dfrac{\partial y_t}{\partial \omega} \\ \dfrac{\partial \theta_t}{\partial V} & \dfrac{\partial \theta_t}{\partial \omega} \end{bmatrix}$$

$$\implies G_u = \begin{bmatrix} \dfrac{1}{\omega}\left(\sin\left(\theta_{t-1}+\omega dt\right)-\sin\left(\theta_{t-1}\right)\right) & \dfrac{V dt}{\omega}\cos\left(\theta_{t-1}+\omega dt\right)-\dfrac{V}{\omega^2}\left(\sin\left(\theta_{t-1}+\omega dt\right)-\sin\left(\theta_{t-1}\right)\right) \\ -\dfrac{1}{\omega}\left(\cos\left(\theta_{t-1}+\omega dt\right)-\cos\left(\theta_{t-1}\right)\right) & \dfrac{V dt}{\omega}\sin\left(\theta_{t-1}+\omega dt\right)+\dfrac{V}{\omega^2}\left(\cos\left(\theta_{t-1}+\omega dt\right)-\cos\left(\theta_{t-1}\right)\right) \\ 0 & dt \end{bmatrix} \tag{10}$$

Now if $|\omega|$ is too small, $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$, G_x and G_u will suffer from numerical instabilities. Thus we now evaluate their limit as $\omega \to 0$. For x_t, we have from 8:

$$x_t = x_{t-1} + \frac{V}{\omega}\left(\sin\left(\theta_{t-1}+\omega dt\right)-\sin\left(\theta_{t-1}\right)\right)$$

$$\implies x_t = x_{t-1} + \frac{V}{\omega}\left(\sin\left(\theta_{t-1}\right)\cos\left(\omega dt\right)+\cos\left(\theta_{t-1}\right)\sin\left(\omega dt\right)-\sin\left(\theta_{t-1}\right)\right) \tag{11}$$

Now making use of the fact that $\sin(\alpha) \to \alpha$ and $\cos(\alpha) \to 1$ as $\alpha \to 0$, we get from 11 as $\omega \to 0$:

$$x_t = x_{t-1} + \frac{V}{\omega}\left(\sin\left(\theta_{t-1}\right)+\omega dt\cos\left(\theta_{t-1}\right)-\sin\left(\theta_{t-1}\right)\right)$$

$$\implies x_t = x_{t-1} + \frac{V}{\omega}\cdot\omega dt\cos\left(\theta_{t-1}\right)$$

$$\implies x_t = x_{t-1} + V dt\cos\left(\theta_{t-1}\right) \tag{12}$$

Similarly as $\omega \to 0$, we get for y_t:

$$y_t = y_{t-1} - \frac{V}{\omega}\left(\cos\left(\theta_{t-1}+\omega dt\right)-\cos\left(\theta_{t-1}\right)\right)$$

$$\implies y_t = y_{t-1} - \frac{V}{\omega}\left(\cos\left(\theta_{t-1}\right)\cos\left(\omega dt\right)-\sin\left(\theta_{t-1}\right)\sin\left(\omega dt\right)-\cos\left(\theta_{t-1}\right)\right)$$

$$\implies y_t = y_{t-1} - \frac{V}{\omega}\left(\cos\left(\theta_{t-1}\right)-\omega dt\sin\left(\theta_{t-1}\right)-\cos\left(\theta_{t-1}\right)\right)$$

$$\implies y_t = y_{t-1} - \frac{V}{\omega}\left(-\omega dt\sin\left(\theta_{t-1}\right)\right)$$

$$\implies y_t = y_{t-1} + V dt\sin\left(\theta_{t-1}\right) \tag{13}$$

Thus our transition model $g(\mathbf{x}_{t-1}, \mathbf{u}_t)$ as $\omega \to 0$ is given by equations 12 and 13:

$$\boxed{\begin{gathered} x_t = x_{t-1} + V dt\cos\left(\theta_{t-1}\right) \\ y_t = y_{t-1} + V dt\sin\left(\theta_{t-1}\right) \\ \theta_t = \theta_{t-1} + \omega dt \end{gathered}} \tag{14}$$

Now taking partial derivatives with respect to $\mathbf{x}_{t-1}$ in 14, we get G_x (as $\omega \to 0$):

$$G_x = \begin{bmatrix} 1 & 0 & -V\,dt\sin(\theta_{t-1}) \\ 0 & 1 & V\,dt\cos(\theta_{t-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

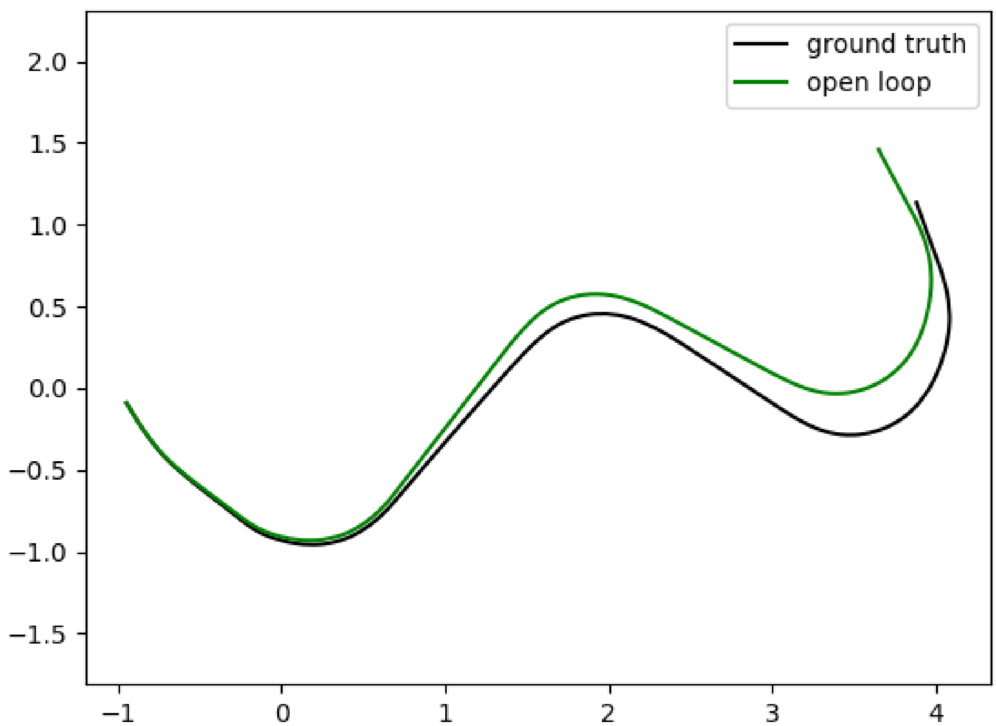
Similarly, taking partial derivatives with respect to $\mathbf{u}_t$ in 14, we get G_u (as $\omega \to 0$):

$$G_u = \begin{bmatrix} dt\cos(\theta_{t-1}) & 0 \\ dt\sin(\theta_{t-1}) & 0 \\ 0 & dt \end{bmatrix} \tag{16}$$

Note: Equations 15 and 16 can also be obtained by taking the limit $\omega \to 0$ in equations 9 and 10 respectively.

(ii) Implemented the computation of g, G_x and G_u in the `compute_dynamics()` function in `turtlebot_model.py`.
Called `compute_dynamics()` in the `transition_model()` method of the `EkfLocalization` class in `ekf.py`.
Validated our work by running `validate_localization_transition_model()` from `validate_ekf.py`.

(iii) Implemented the dynamics transition update in the `transition_update()` method of the `Ekf` class in `ekf.py`.
Validated our work by running `validate_ekf_transition_update()` from `validate_ekf.py`.
The file `ekf_open_loop.png` is shown below.



(iv) Let W denote the world frame, B denote the base frame of the robot, and C denote the robot's camera frame. The robot's state ${}^W\mathbf{x}_B = \begin{bmatrix} {}^Wx_B & {}^Wy_B & {}^W\theta_B \end{bmatrix}^T$ defines the offset/yaw of the robot's base frame with respect to the world frame. Let the offset/yaw of the robot's camera frame with respect to its base frame be ${}^B\mathbf{x}_C = \begin{bmatrix} {}^Bx_C & {}^By_C & {}^B\theta_C \end{bmatrix}^T$. Using this, we can construct 3×3 homogeneous transformation matrices. The transformation matrix relating the base frame to the world frame is given by:

$$
{}^WT_B = \begin{bmatrix} \cos({}^W\theta_B) & -\sin({}^W\theta_B) & {}^Wx_B \\ \sin({}^W\theta_B) & \cos({}^W\theta_B) & {}^Wy_B \\ 0 & 0 & 1 \end{bmatrix} \tag{17}
$$

Similarly, the transformation matrix relating the camera frame to the base frame is given by:

$$
{}^BT_C = \begin{bmatrix} \cos({}^B\theta_C) & -\sin({}^B\theta_C) & {}^Bx_C \\ \sin({}^B\theta_C) & \cos({}^B\theta_C) & {}^By_C \\ 0 & 0 & 1 \end{bmatrix} \tag{18}
$$

Hence, the transformation matrix relating the camera frame to the world frame is given by:

$$
{}^WT_C = {}^WT_B {}^BT_C
$$

$$
\implies {}^WT_C = \begin{bmatrix} \cos({}^W\theta_B) & -\sin({}^W\theta_B) & {}^Wx_B \\ \sin({}^W\theta_B) & \cos({}^W\theta_B) & {}^Wy_B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos({}^B\theta_C) & -\sin({}^B\theta_C) & {}^Bx_C \\ \sin({}^B\theta_C) & \cos({}^B\theta_C) & {}^By_C \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
\implies {}^WT_C = \begin{bmatrix} \cos({}^W\theta_B + {}^B\theta_C) & -\sin({}^W\theta_B + {}^B\theta_C) & {}^Wx_B + {}^Bx_C\cos({}^W\theta_B) - {}^By_C\sin({}^W\theta_B) \\ \sin({}^W\theta_B + {}^B\theta_C) & \cos({}^W\theta_B + {}^B\theta_C) & {}^Wy_B + {}^Bx_C\sin({}^W\theta_B) + {}^By_C\cos({}^W\theta_B) \\ 0 & 0 & 1 \end{bmatrix} \tag{19}
$$

Hence, the offset and yaw of the camera frame with respect to the world frame is given by:

$$
\begin{aligned}
{}^Wx_C &= {}^Wx_B + {}^Bx_C\cos({}^W\theta_B) - {}^By_C\sin({}^W\theta_B) \\
{}^Wy_C &= {}^Wy_B + {}^Bx_C\sin({}^W\theta_B) + {}^By_C\cos({}^W\theta_B) \\
{}^W\theta_C &= {}^W\theta_B + {}^B\theta_C
\end{aligned} \tag{20}
$$

Now, the equation of a line in the world frame with parameters $({}^W\alpha, {}^Wr)$ is given by:

$$
{}^Wx\cos({}^W\alpha) + {}^Wy\sin({}^W\alpha) = {}^Wr \tag{21}
$$

The arbitrary points $({}^Wx, {}^Wy)$ in the world frame are mapped to points $({}^Cx, {}^Cy)$ in the camera frame by:

$$
\begin{bmatrix} {}^Wx \\ {}^Wy \\ 1 \end{bmatrix} = {}^WT_C \begin{bmatrix} {}^Cx \\ {}^Cy \\ 1 \end{bmatrix}
$$

$$
\implies \begin{bmatrix} {}^Wx \\ {}^Wy \\ 1 \end{bmatrix} = \begin{bmatrix} \cos({}^W\theta_C) & -\sin({}^W\theta_C) & {}^Wx_C \\ \sin({}^W\theta_C) & \cos({}^W\theta_C) & {}^Wy_C \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^Cx \\ {}^Cy \\ 1 \end{bmatrix} \tag{22}
$$

where ${}^W x_C$, ${}^W y_C$ and ${}^W\theta_C$ are given in 20. Hence, equation 22 gives us:

$$\begin{aligned}
{}^W x &= {}^W x_C + {}^C x \cos\left({}^W\theta_C\right) - {}^C y \sin\left({}^W\theta_C\right) \\
{}^W y &= {}^W y_C + {}^C x \sin\left({}^W\theta_C\right) + {}^C y \cos\left({}^W\theta_C\right)
\end{aligned} \tag{23}$$

Substituting 23 in 21, we get:

$$\begin{aligned}
&\left({}^W x_C + {}^C x \cos\left({}^W\theta_C\right) - {}^C y \sin\left({}^W\theta_C\right)\right)\cos\left({}^W\alpha\right) + \left({}^W y_C + {}^C x \sin\left({}^W\theta_C\right) + {}^C y \cos\left({}^W\theta_C\right)\right)\sin\left({}^W\alpha\right) = {}^W r \\
&\implies {}^W x_C \cos\left({}^W\alpha\right) + {}^W y_C \sin\left({}^W\alpha\right) + {}^C x \cos\left({}^W\alpha - {}^W\theta_C\right) + {}^C y \sin\left({}^W\alpha - {}^W\theta_C\right) = {}^W r \\
&\implies {}^C x \cos\left({}^W\alpha - {}^W\theta_C\right) + {}^C y \sin\left({}^W\alpha - {}^W\theta_C\right) = {}^W r - {}^W x_C \cos\left({}^W\alpha\right) - {}^W y_C \sin\left({}^W\alpha\right)
\end{aligned} \tag{24}$$

From equation 24 we can observe that equivalent parameters $({}^C\alpha, {}^C r)$ in the camera frame for the line given by 21 are:

$$\boxed{\begin{aligned}
{}^C\alpha &= {}^W\alpha - {}^W\theta_C \\
{}^C r &= {}^W r - {}^W x_C \cos\left({}^W\alpha\right) - {}^W y_C \sin\left({}^W\alpha\right)
\end{aligned}} \tag{25}$$

The Jacobian H is given by:

$$H = \begin{bmatrix} \frac{\partial {}^C\alpha}{\partial {}^W x_B} & \frac{\partial {}^C\alpha}{\partial {}^W y_B} & \frac{\partial {}^C\alpha}{\partial {}^W \theta_B} \\ \frac{\partial {}^C r}{\partial {}^W x_B} & \frac{\partial {}^C r}{\partial {}^W y_B} & \frac{\partial {}^C r}{\partial {}^W \theta_B} \end{bmatrix} \tag{26}$$

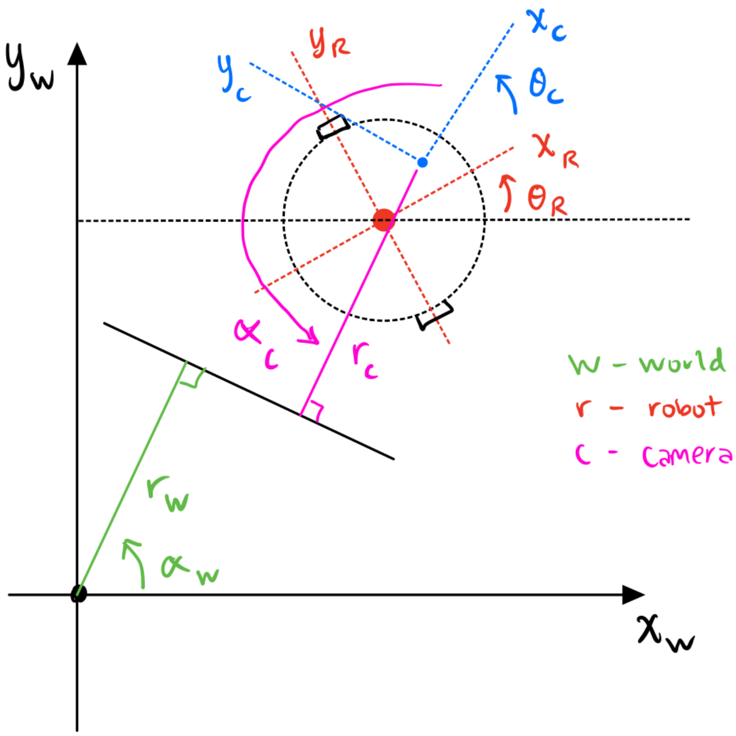
We substitute 20 in 25 to get:

$$\begin{aligned}
{}^C\alpha &= {}^W\alpha - {}^W\theta_B - {}^B\theta_C \\
{}^C r &= {}^W r - {}^W x_B \cos\left({}^W\alpha\right) - {}^W y_B \sin\left({}^W\alpha\right) - {}^B x_C \cos\left({}^W\theta_B - {}^W\alpha\right) + {}^B y_C \sin\left({}^W\theta_B - {}^W\alpha\right)
\end{aligned} \tag{27}$$

Finally, taking partial derivatives in 27, we get the Jacobian H:

$$H = \begin{bmatrix} 0 & 0 & -1 \\ -\cos\left({}^W\alpha\right) & -\sin\left({}^W\alpha\right) & {}^B x_C \sin\left({}^W\theta_B - {}^W\alpha\right) + {}^B y_C \cos\left({}^W\theta_B - {}^W\alpha\right) \end{bmatrix} \tag{28}$$

Implemented the coordinate change for a map entry between the world frame and camera frame in the `transform_line_to_scanner_frame()` function in `turtlebot_model.py`.
Completed the `compute_predicted_measurements()` method of the `EkfLocalization` class in `ekf.py`.
Validated our work by running `validate_localization_compute_predicted_measurements()` from `validate_ekf.py`.
The annotated diagram `diagram_annotation.png` is shown below.



(v) Algorithm in pseudocode:

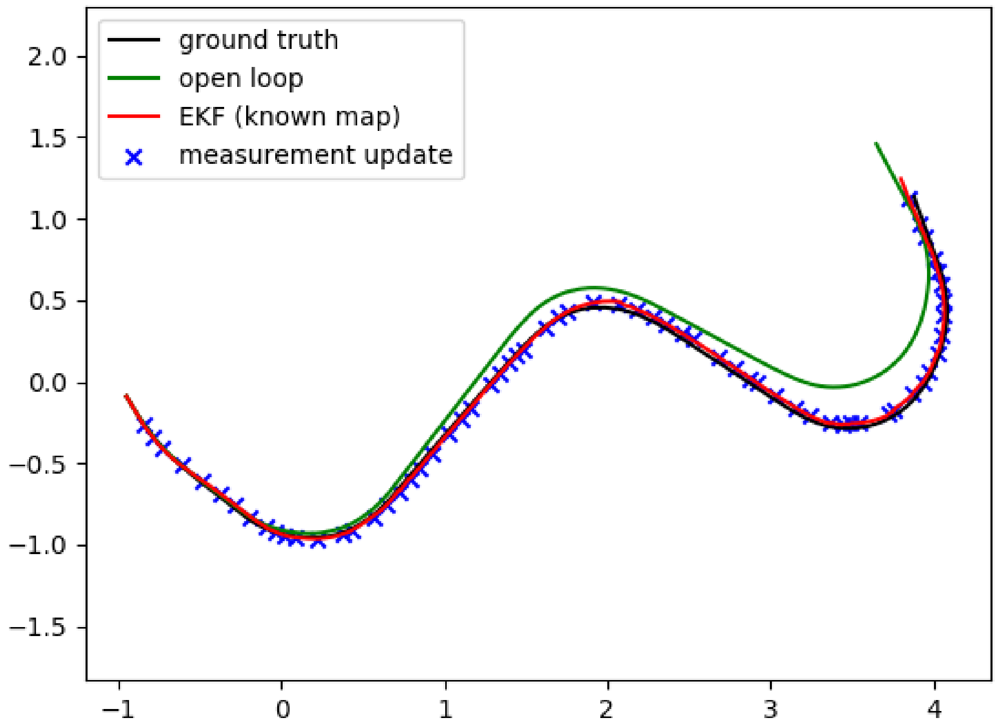
Implemented the measurement association process in `compute_innovations()` method of the `EkfLocalization` class in `ekf.py`.
Validated our work by running `validate_localization_compute_innovations()` from `validate_ekf.py`.

(vi) Implemented the assembly of joint measurements in the `measurement_model()` method of the `EkfLocalization` class in `ekf.py`.

(vii) Implemented the measurement update in the `measurement_update()` method of the `Ekf` class in `ekf.py`.
Validated our work by running `validate_ekf_localization()` from `validate_ekf.py`.
The file `ekf_localization.png` is shown below.

Algorithm 1 Pseudocode for compute_innovations()

```
v_list = [ ]                ▷ Initialize empty lists for innovation vectors, covariance matrices, & Jacobians
Q_list = [ ]
H_list = [ ]

for i ≤ I do                                          ▷ Loop through number of observed lines
    z_i ← z_raw[i]                         ▷ Extract lines & covariance matrices from scanner data
    Q_i ← Q_raw[i]
    d_ij^min ← inf                                 ▷ Set initial minimum Mahalanobis distance

    for j ≤ J do                                      ▷ Loop through number of predicted lines
        h_j ← hs[j]                        ▷ Extract line parameters & Jacobian for predicted line
        H_j ← Hs[j]
        v_ij ← z_i − h_j     ▷ Calculate innovation vector, innovation covariance, & Mahalanobis distance
        S_ij ← H_j Σ H_j^T + Q_i

        if d_ij < d_ij^min then   ▷ Set new minimum Mahalanobis distance, innovation vector, & Jacobian
            d_ij^min ← d_ij
            v_ij^min ← v_ij
            H_j^min ← H_j
        end if
    end for

    if d_ij < g^2 then              ▷ Check if minimum Mahalanobis distance meets validation gate
        V_list add V_ij^min
        Q_list add Q_i
        H_list add H_j^min
    end if

end for
return v_list, Q_list, H_list
```

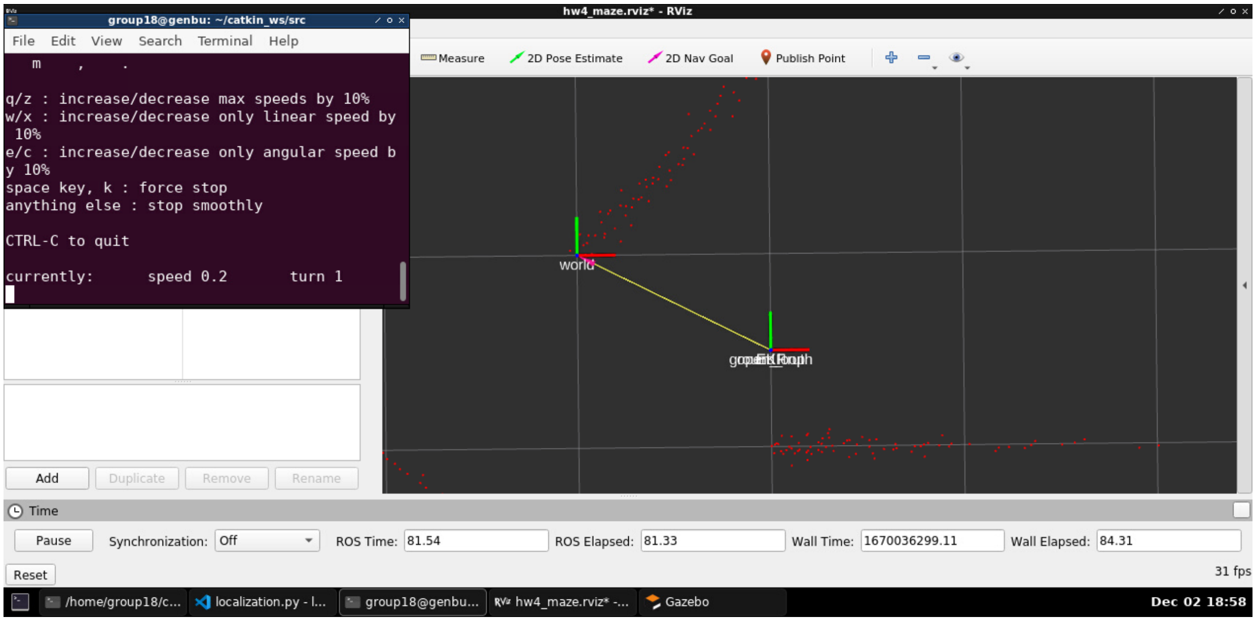


(viii) The uncertainities we injected were:

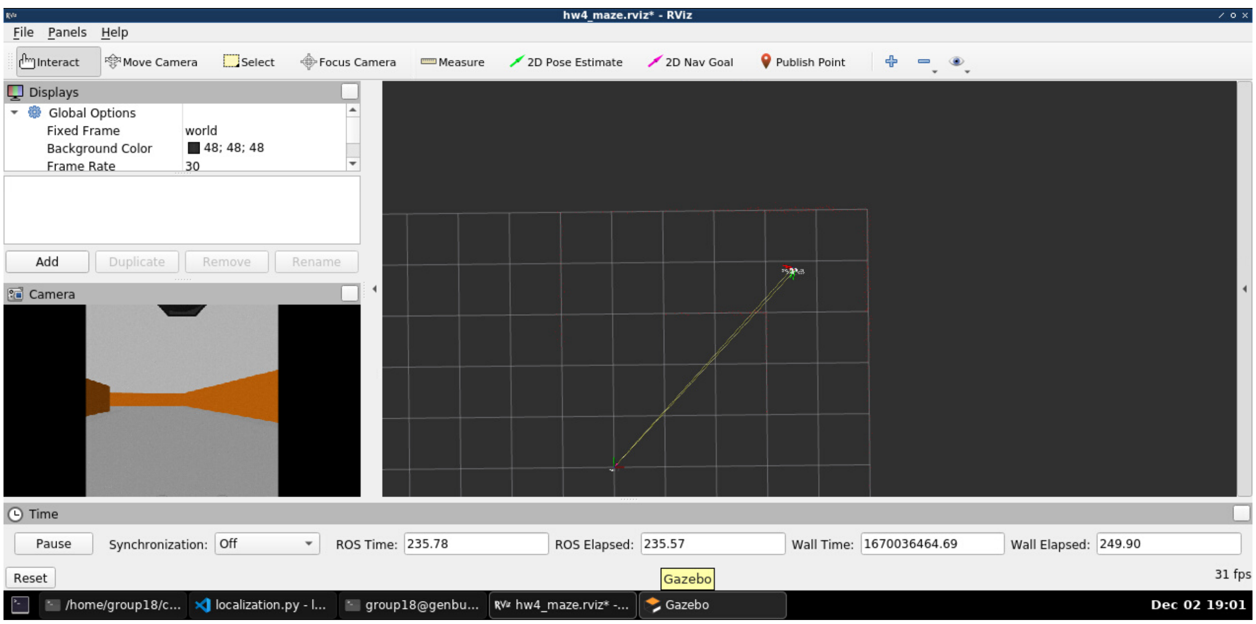
(a) Perturbing the initial position: We perturbed the initial position by 0.25 units in both x and y and observed that the EKF state estimates converge to the ground truth. However, if we give a large enough perturbation (of around 1 unit in both x and y), then the EKF state estimates and ground truth diverge quite quickly.

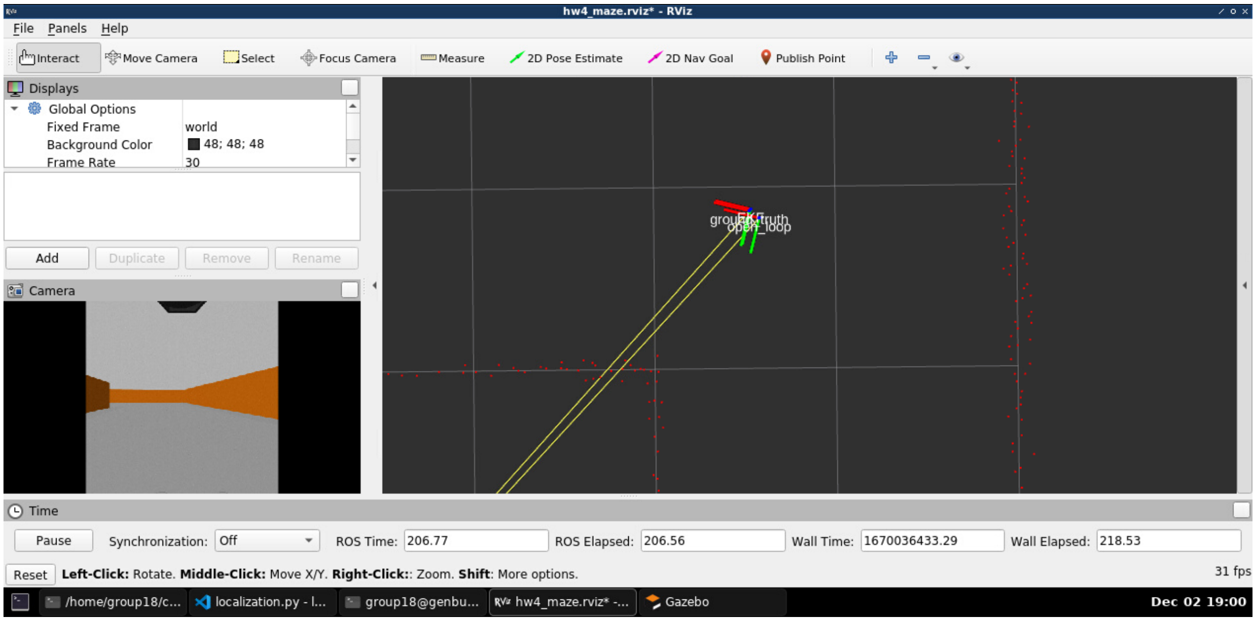
(b) Add Gaussian noise to the control/scanner measurements: We increased the values of the variables `Sigma0`, `R`, `var_theta` and `var_rho` by a factor of 50, in `maze_sim_parameters.py`. For this also we observed that the EKF state estimates converge to the ground truth.

(ix) We observed that the EKF state estimates and open loop estimates diverge when we take a sharp turn or go very close to a wall. The screenshots in RViz are shown below:
(1) The initial state:

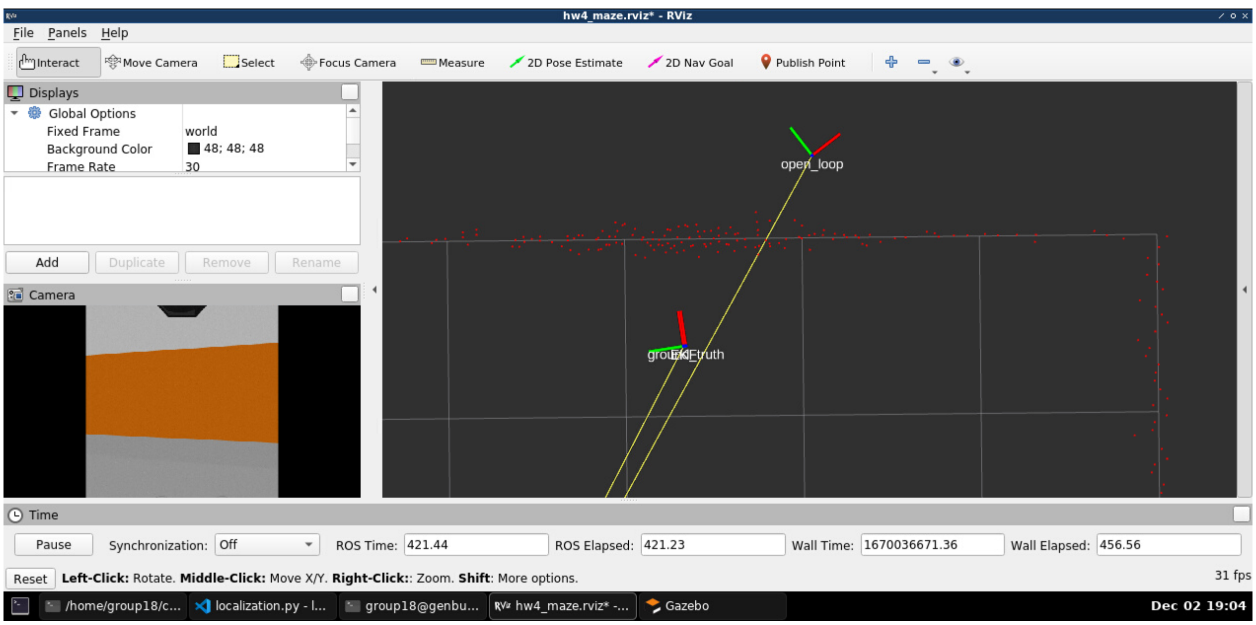


(2) The TurtleBot has moved far from the initial state:





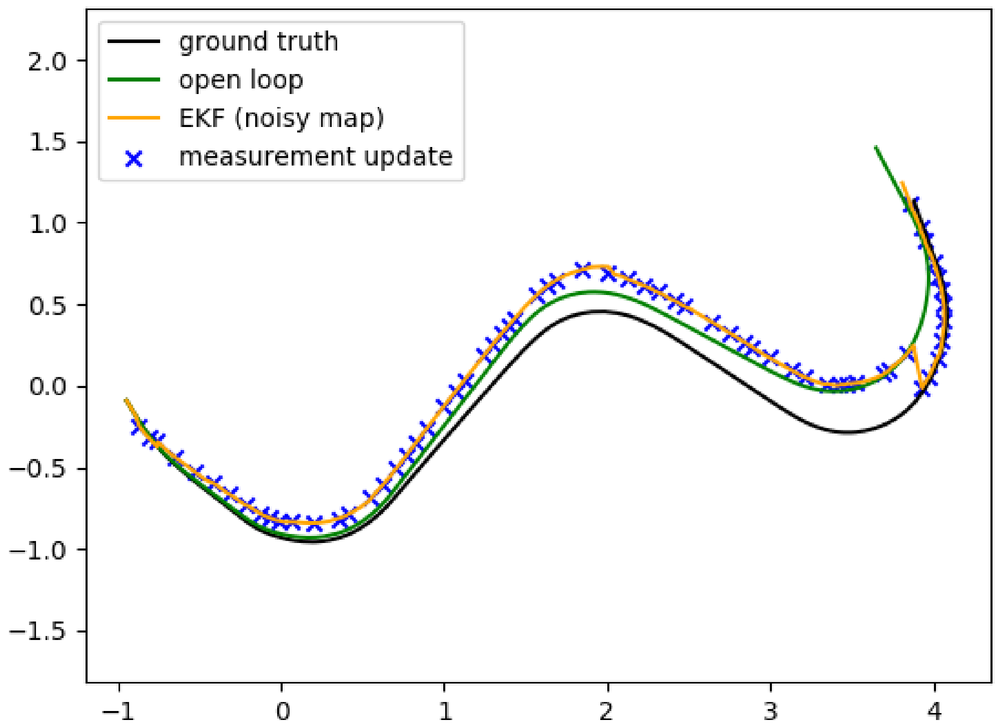
(3) When the EKF state estimates and open loop estimates diverge:



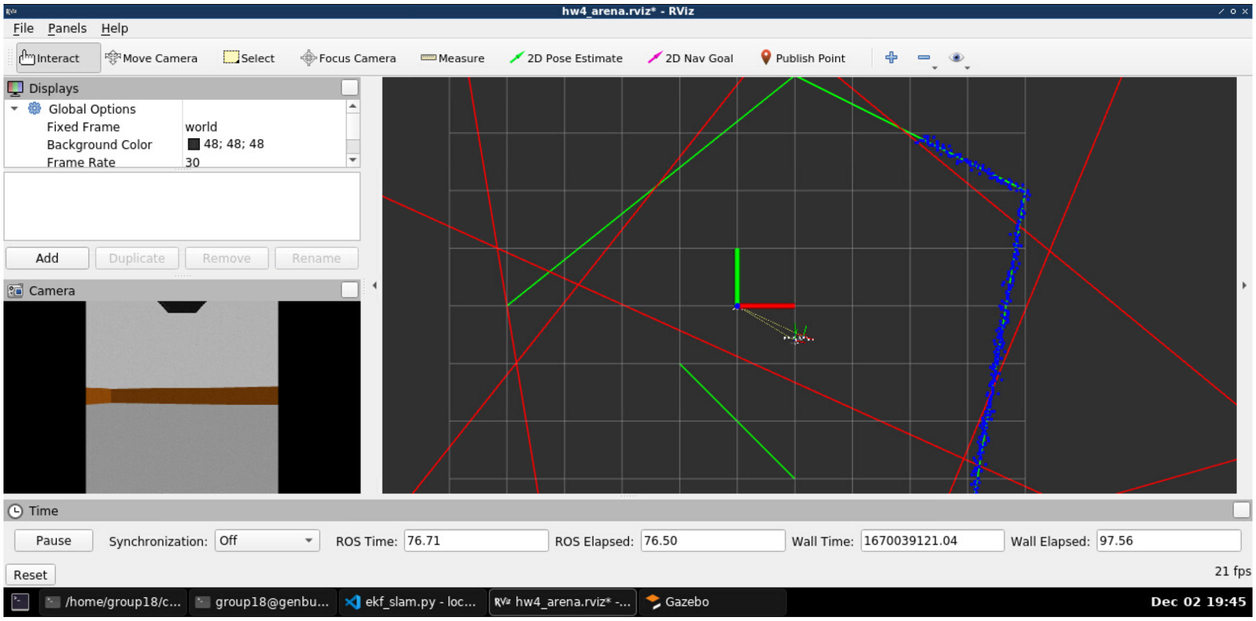
Problem 2: EKF SLAM

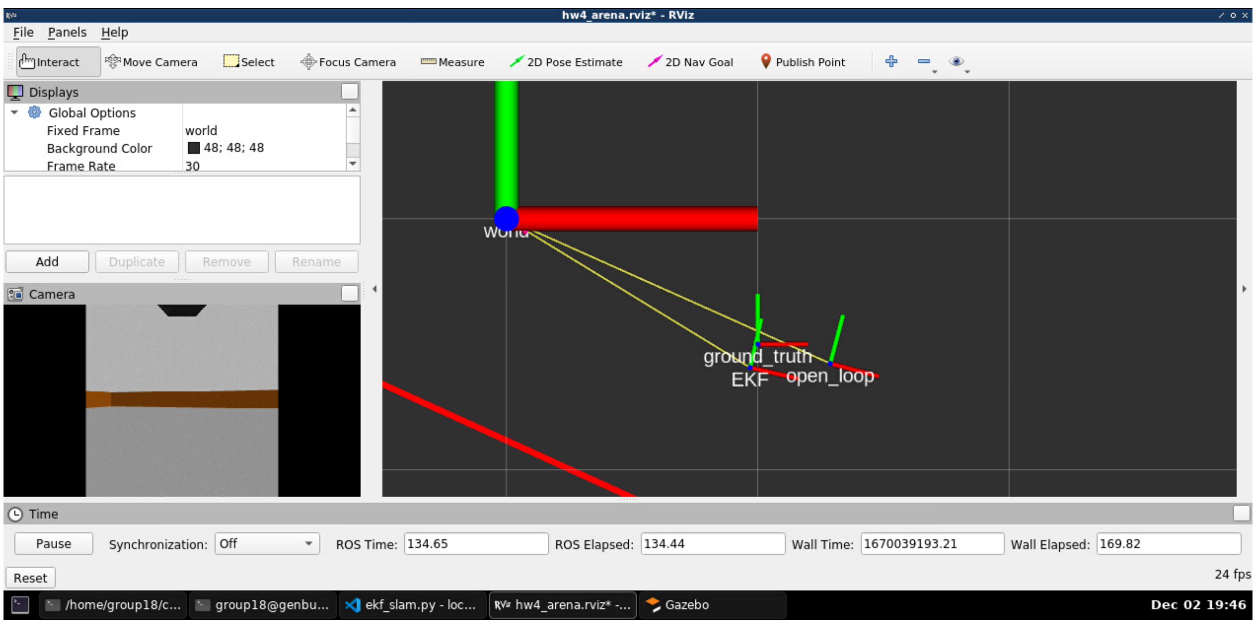
(i) Implemented the computation of g, G_x and G_u in the `transition_model()` method of the `EkfSlam` class in `ekf.py`.
The dimension of the state vector $\mathbf{x}(t)$ is $3 + 2J$, where J is the number of map elements.

(ii) Reimplemented `measurement_model()`, `compute_innovations()` and `compute_predicted_measurements()` in the `EkfSlam` class in `ekf.py`.
Validated our work by running `validate_ekf_slam()` from `validate_ekf.py`.
The file `ekf_slam.png` is shown below.

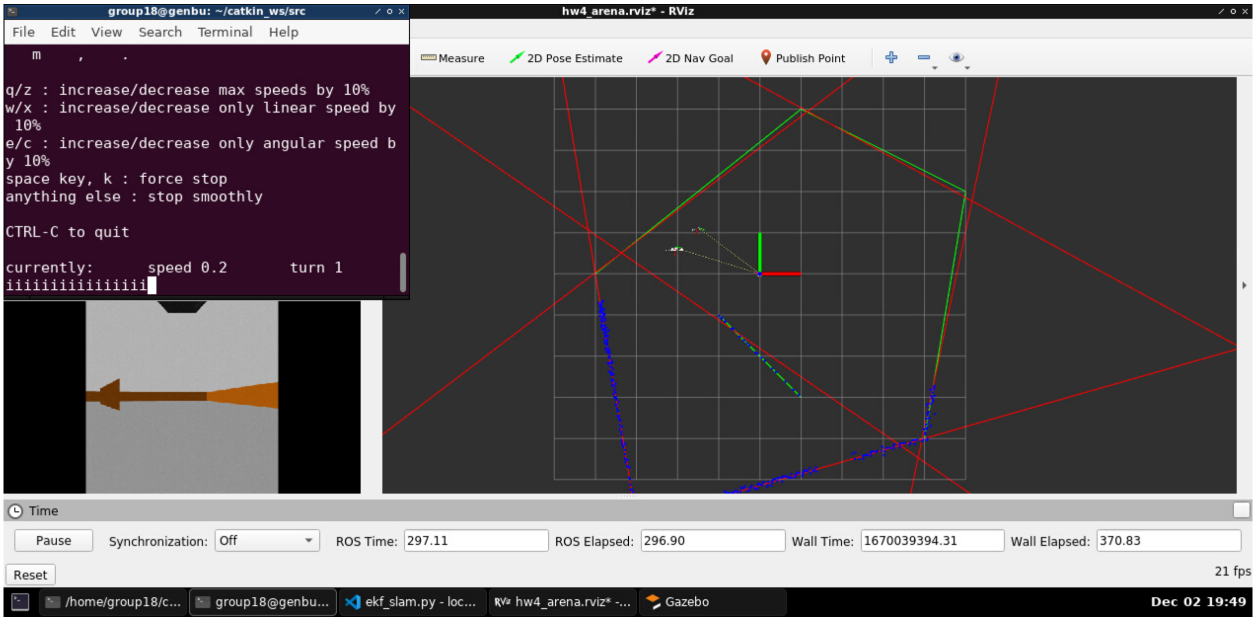


(iii) We just drive around the arena to make the map estimates converge to the right one. We observed that the EKF state estimates and ground truth diverge with sharp turns or collisions with the walls. The screenshots in RViz are shown below:
(1) The initial state:

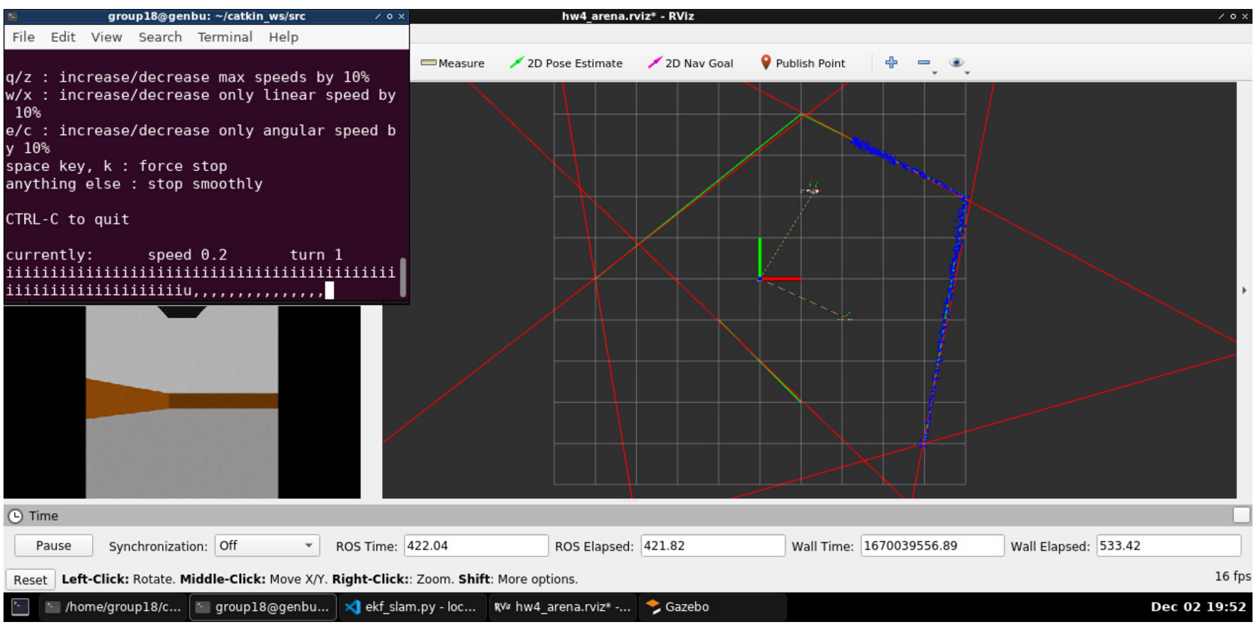


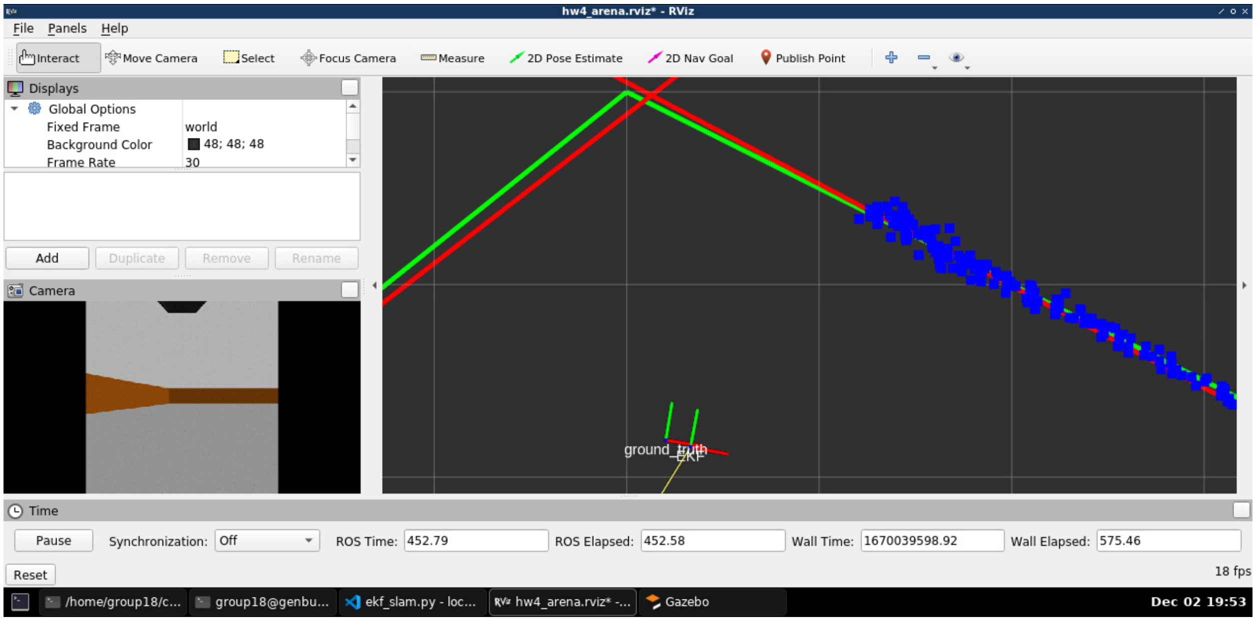


(2) The TurtleBot has moved away from the initial state:



(3) The map estimates have converged:




hw4_arena.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
world
Background Color
48; 48; 48
Frame Rate
30
Add
Duplicate
Remove
Rename
Camera
ground_truth
EKF
Time
Pause
Synchronization:
Off
ROS Time:
452.79
ROS Elapsed:
452.58
Wall Time:
1670039598.92
Wall Elapsed:
575.46
Reset
18 fps
/home/group18/c...
group18@genbu...
ekf_slam.py - loc...
hw4_arena.rviz* -...
Gazebo
Dec 02 19:53