AA 274: Principles of Robot Autonomy

Problem Set 4: Filtering, Localization, and Mapping

## Problem 1: EKF Localization

viii)

By driving around the TurtleBot, the open loop and EKF state estimates are converged at first. But it starts to diverge when the robot hits the wall. The collision between the robot and the wall seems to cause the state estimates to diverge from each other. The three screenshots of RViz are shown below.

Figure 1 The initial state

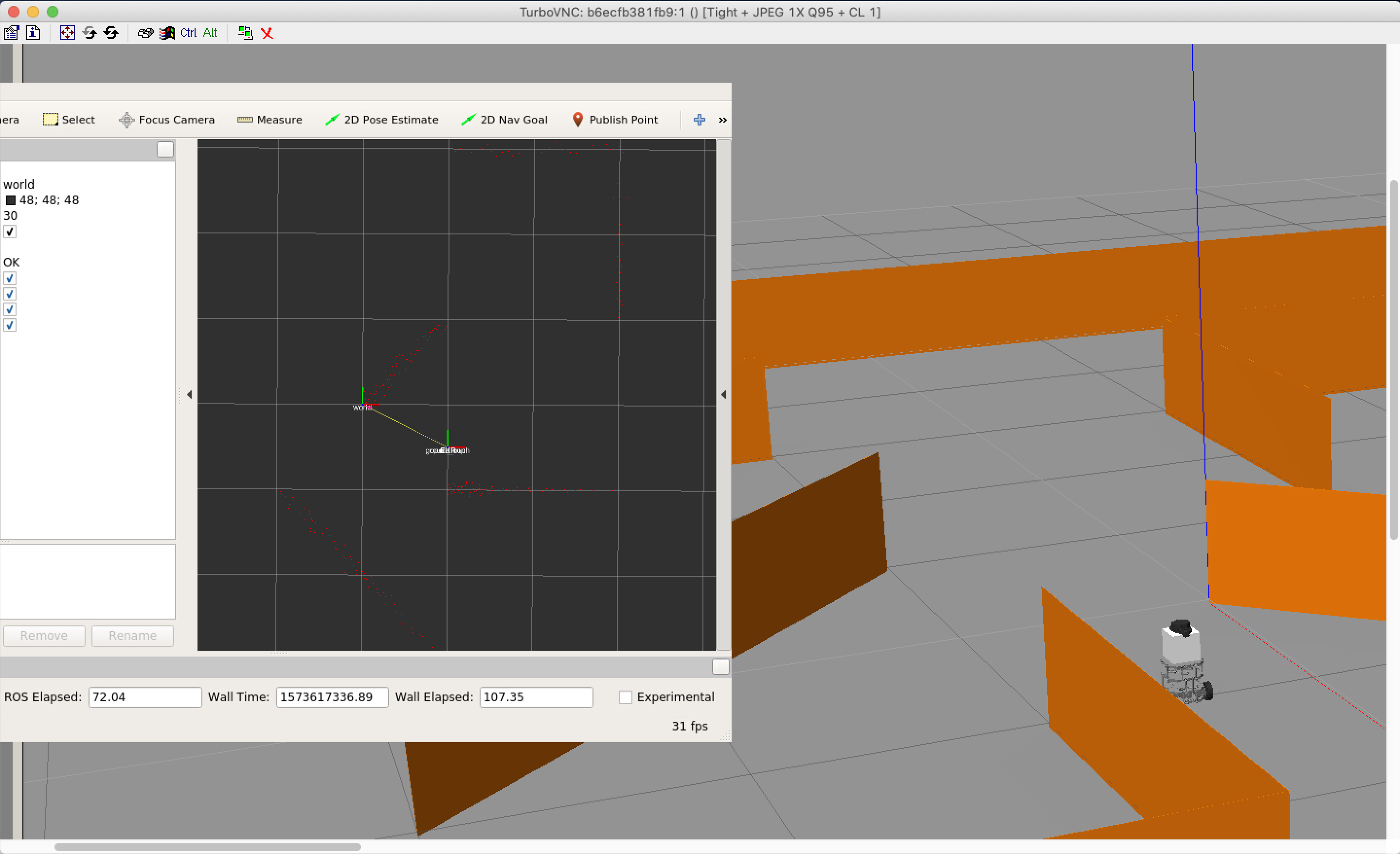


Figure 2 Far from the initial state

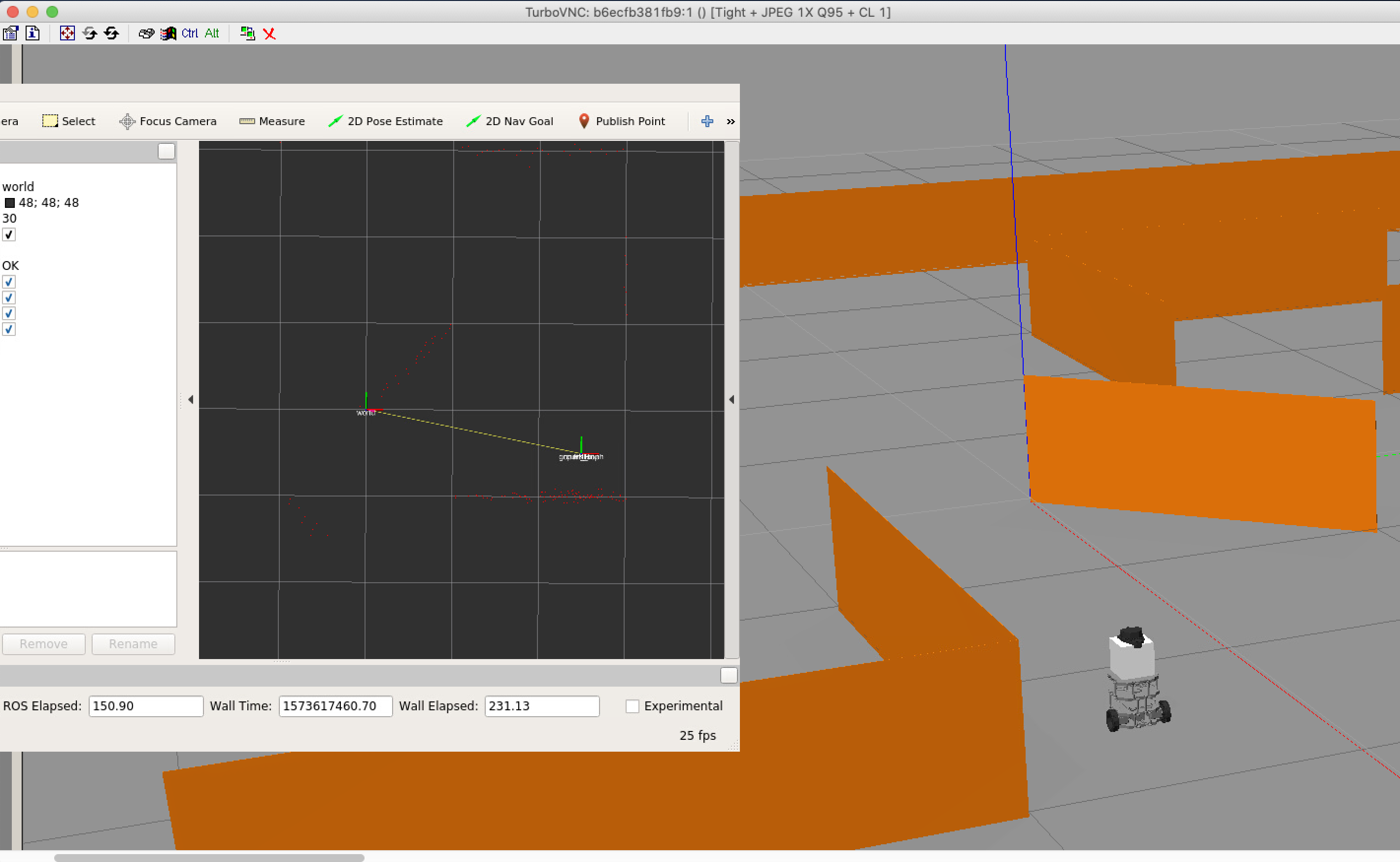
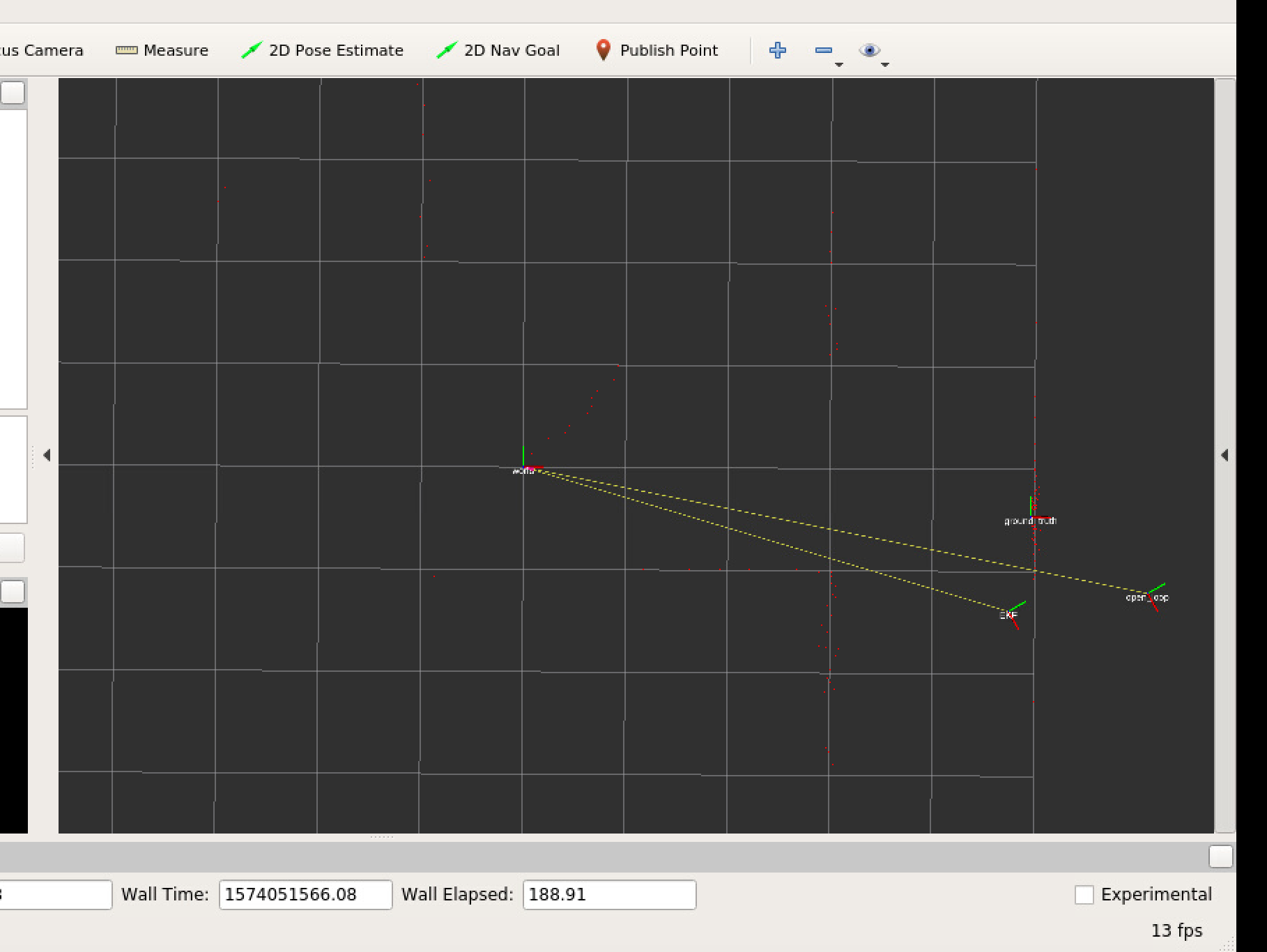


Figure 3 The state estimates start to diverge



## Problem 2: EKF Localization

iii)

By driving the TurtleBot around, the map estimates changes at first. The EKF estimate will converge to the true estimate over time but the open loop estimate will not. It seems that when robot hits the wall, the EKF and ground truth estimates starts to diverge. And when the robot runs normally around the arena, the EKF estimate will gradually converge to the true estimate.

Figure 4 Initial state

A screenshot of a computer

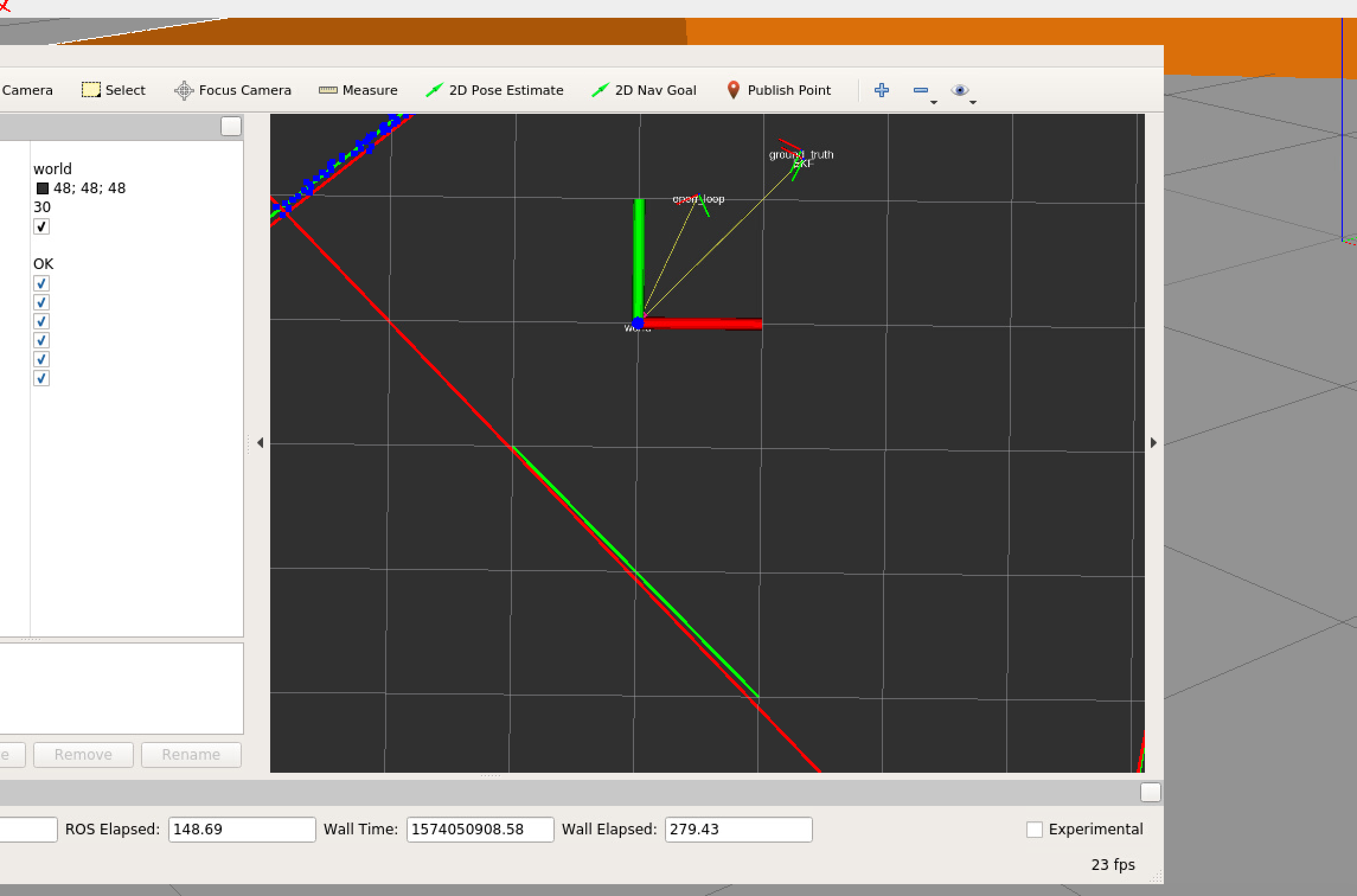
Description automatically generated

Figure 5 Away from the initial state

A screenshot of a computer

Description automatically generated

Figure 6 The map estimates have converged



# Extra Problem: Monte Carlo Localization

iv)

By driving around the TurtleBot, the open loop and MCL state estimates converge to the true estimates at first. When the robot hits the wall, both the MCL and open loop state estimates will diverge from the true estimate. But the MCL estimate will self-correct the error after a certain time interval and converge to true estimate.

When the number of particles increase, the time it takes to for MCL estimate to self-correct will decrease. More number of particles will improve the performance of the self-correctness of MCL estimate.

Figure 7 Initial state:

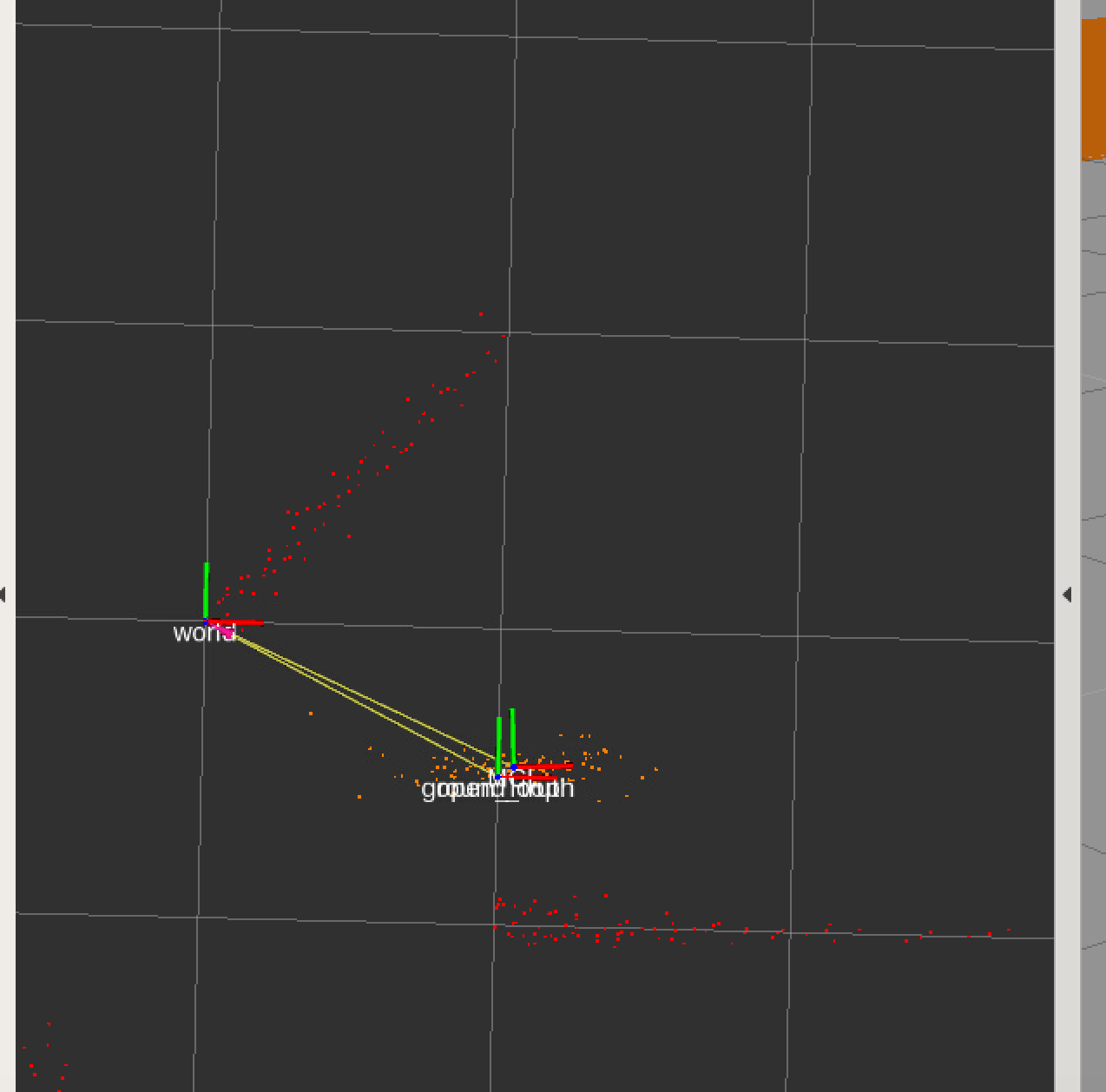


Figure 8 Away from initial state.

A picture containing colorful

Description automatically generated

Figure 9 the MCL and ground truth estimates diverge

A picture containing kite, colorful

Description automatically generated

v)

* resample()

m = np.linspace(0, self.M - 1, num=self.M)  
u = np.sum(ws) \* (r + m / self.M)  
  
ws\_sum = np.cumsum(ws)  
idx = np.searchsorted(ws\_sum, u)  
self.xs = xs[idx]  
self.ws = ws[idx]

* transition\_model()

x = self.xs[:, 0] # [M, 1]  
y = self.xs[:, 1] # [M, 1]  
th = self.xs[:, 2] # [M, 1]  
V = us[:, 0] # [M, 1]  
om = us[:, 1] # [M, 1]  
th\_t = th + om \* dt  
# for om < Epsilon  
x\_t1 = x + V \* dt \* np.cos(th\_t)  
y\_t1 = y + V \* dt \* np.sin(th\_t)  
  
x\_t1 = np.where(abs(om) >= EPSILON\_OMEGA, 0, x\_t1)  
y\_t1 = np.where(abs(om) >= EPSILON\_OMEGA, 0, y\_t1)  
  
# for om > Epsilon  
x\_t2 = (V / om) \* (np.sin(th\_t) - np.sin(th)) + x  
y\_t2 = (-V / om) \* (np.cos(th\_t) - np.cos(th)) + y  
  
x\_t2 = np.where(abs(om) < EPSILON\_OMEGA, 0, x\_t2)  
y\_t2 = np.where(abs(om) < EPSILON\_OMEGA, 0, y\_t2)  
  
x\_t = x\_t1 + x\_t2  
y\_t = y\_t1 + y\_t2  
  
g = np.c\_[x\_t, y\_t, th\_t]

* compute\_innovations()

alpha = self.map\_lines[0, :] # [1, J]  
r = self.map\_lines[1, :] # [1, J]  
  
x\_cam, y\_cam, th\_cam = self.tf\_base\_to\_camera # float  
# compute pose of camera in world frame  
  
hs = np.zeros((self.M, 2, self.map\_lines.shape[1]))  
for i in range(self.M):  
 x = self.xs[i, 0] # [M, 1]  
 y = self.xs[i, 1] # [M, 1]  
 th = self.xs[i, 2] # [M, 1]  
 x\_cam\_w = x\_cam \* np.cos(th) - y\_cam \* np.sin(th) + x # [1, 1]  
 y\_cam\_w = x\_cam \* np.sin(th) + y\_cam \* np.cos(th) + y  
  
 alpha\_pre = alpha - th - th\_cam  
 r\_pre = r - x\_cam\_w \* np.cos(alpha) - y\_cam\_w \* np.sin(alpha)  
  
 # normalize  
 idx = r\_pre < 0  
 r\_pre[idx] \*= -1  
 alpha\_pre[idx] += np.pi  
 alpha\_pre = (alpha\_pre + np.pi) % (2 \* np.pi) - np.pi  
  
 hs[i] = np.array([alpha\_pre, r\_pre])