3-Dimensional Robot Localization and Control using Particle Filtering

Haochen Ding

1 Introduction

Localization refers to a robot's ability to determine its position within an environment, which is essential for autonomous tasks such as path planning, obstacle avoidance, and decision-making.

Robot localization is achieved through the estimation of internal states of the robot system, which in this project are defined as the robot's position and velocity in three-dimensional space. A foundational approach to state estimation is Bayesian filtering, a probabilistic method that infers the state of a system by combining prior knowledge (system dynamics) with real-time observations. However, traditional Bayes filters such as the Kalman filter assume linear, Gaussian systems, which limits their effectiveness in more complex, nonlinear environments.

To address these limitations, we applied particle filtering, a Monte Carlo-based implementation of Bayesian filtering introduced in the 1990s. Particle filters approximate the posterior distribution of the system's states using a set of weighted samples (particles), making them particularly well-suited for nonlinear and non-Gaussian models. Each particle represents a possible state, and through repeated prediction and update steps, the filter converges toward the true state.

In this project, we implement a particle filtering algorithm to estimate the position and velocity of a feedback-controlled robot navigating a simulated 3-D space. To provide observation data for the particle filter, we developed a simulation of the robot's system dynamics in MATLAB. The robot is guided by a feedback controller designed to ensure convergence to a predefined target state. Our work demonstrates how particle filtering can be effectively integrated into robotic localization systems, especially when traditional assumptions of linearity or Gaussian noise are violated.

2 Overview of the Particle Filter Algorithm

2.1 Bayesian Filtering

Bayesian filtering is a technique for estimating the state of a system. In Bayesian filtering, the state of a system is considered as a probability distribution, or belief, that is gradually updated as time evolves.

Each cycle of Bayesian filtering consists of two steps. In the prediction step, the prior belief is first propagated through the system dynamics using the Chapman-Kolmogorov equation:

$$p(x_t \mid x_0) = \int p(x_t \mid x_{t-1})\, p(x_{t-1} \mid x_0)\, dx_{t-1},$$

assuming the Markov property.

After we have a predicted belief and actual observation data, in the update step, we use Bayes' rule to update the expected belief by calculating the possibility of getting that observation given the predicted belief:

$$p(x_t \mid z_{1:t}) = \frac{p(z_t \mid x_t)\, p(x_t \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})}$$

2.2 Particle Filtering

The idea of particle filtering is to approximate the system's belief using the Monte Carlo method. It represents the probability distribution using a set of weighted samples of the system states.

After initializing the particles, they are propagated through the system dynamics in each cycle, similar to Bayesian filtering. In the update step, the weights of the particles are updated based on how likely they match the actual observation. The states are estimated by taking the weighted mean over all particles.

Algorithm 1 Particle Filtering Robot Localization

```
1:  **Input:** Initialize ranges. Generate $\{z_{i0}, \omega_{i0}\}_{i=1}^{P}$.
2:  **Output:** State vector estimate $\hat{z}_{N-1}$.
3:  for $n = 0$ to $N-1$ do
4:      for $i = 1$ to $P$ do
5:          Compute the weight of the $i$-th particle:
6:          $\omega_{in} \propto \frac{1}{\text{distance}(z_{in}, r_{0:n})}$
7:          Normalize weight: $\overline{\omega}_{in} = \frac{\omega_{in}}{\sum_{i=1}^{P} \omega_{in}}$
8:          Calculate updated mean and covariance:
9:          $\mu_{n+1} = \sum_{i=1}^{P} \overline{\omega}_{in} z_{in}$
10:         $\Sigma_{n+1} = \sum_{i=1}^{P} \overline{\omega}_{in} (z_{in} - \mu_n)(z_{in} - \mu_n)^T$
11:         Propagate particles through the robot's dynamics:
12:         $z_{in+1} = f(z_{in}, \text{control input}, \text{noise})$
13:     end for
14: end for
```

In order to avoid particle degradation as the system propagates, we applied systematic resampling after the update step, which replaces highly improbable particles with particles with higher weights. Specifically, systematic resampling ensures selection from all parts of the particle space, which improves the filter's performance when the system is highly dynamic.

3 Robot System Dynamics and Controller

We simulated the system dynamics of our 3-D robot using MATLAB to provide the observation required for the particle filter. The following state-space model can be used to describe the system dynamics of the robot:

$$\mathbf{p}_{t+1} = \mathbf{A}\mathbf{p}_t + \mathbf{B}\mathbf{a} + \mathbf{w}$$

, where the state vector $\mathbf{p} = [x_x, v_x, x_y, v_y, x_z, v_z]^T$ denotes the positions and velocities of the robot in all three dimensions. The state transition matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and the control input matrix $\mathbf{B}$ is defined as

$$\mathbf{B} = \begin{bmatrix} \frac{dt^2}{2} & dt & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{dt^2}{2} & dt & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{dt^2}{2} & dt \end{bmatrix}.$$

In order to achieve stable destination control on our robot, we simulated a feedback controller, determining the control input using the state estimates. The control input $\mathbf{a}$, which in our case is the acceleration of the robot, is modeled as follows:

$$\mathbf{a} = [a_x, a_y, a_z]^T \ , \quad \text{where} \quad \begin{cases} a_x = k_{px}p_x + k_{vx}v_x \\ a_y = k_{py}p_y + k_{vy}v_y \\ a_z = k_{pz}p_z + k_{vz}v_z \end{cases} , \quad \text{with all gains negative.}$$

Having all gains smaller than zero ensures a stable feedback control system that converges to a destination instead of going to infinity. To simulate randomness in real-life conditions, we added Gaussian noise to the dynamics of the system, which is represented by the term $\mathbf{w}$. The result of our simulation is shown below.

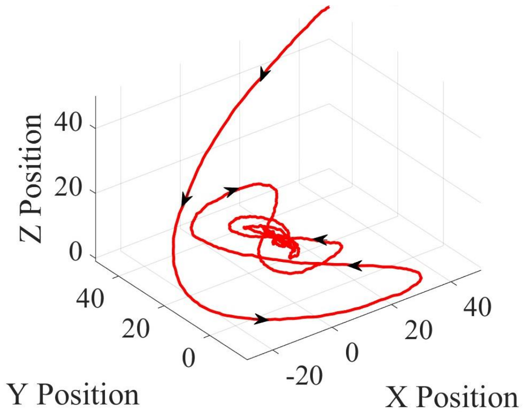


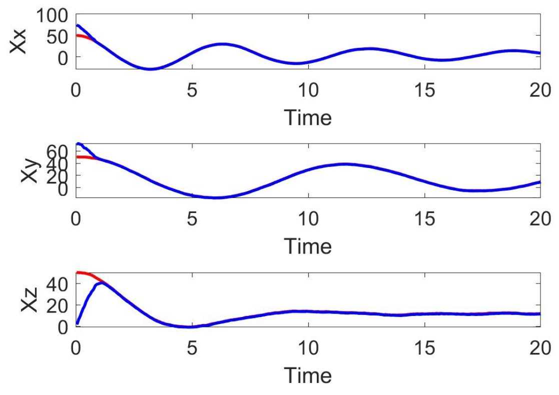
Figure 1: 3-D Robot System Simulation

We can see from the simulated trajectory diagram that the 3-D robot is gradually approaching its destination, which in our case is the origin. This implies a successful implementation of the feedback controller.

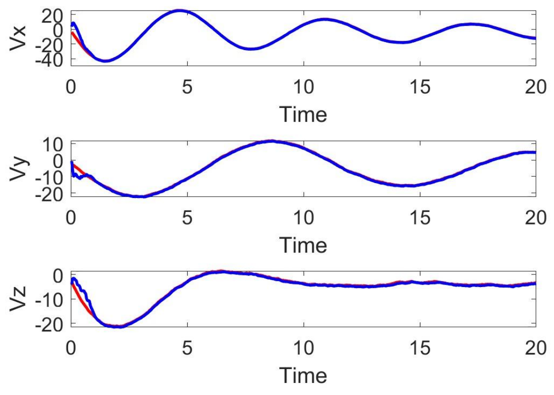
4 Results

Using the data generated from our 3-D robot simulation as observation, we tested the performance of the particle filter. We can see that both position and velocity estimations converge to the true states quickly and maintain convergence as time evolves. From the individual state estimation figures, we can see that the absolute errors between the true state values and the estimates gradually decrease and maintain a mean of zero, indicating convergence of the estimation.

From the individual state estimation results, we can see that our estimations of all six states converge to the real value in a short amount of time. The state estimation results are shown below.



(a) Position Estimation

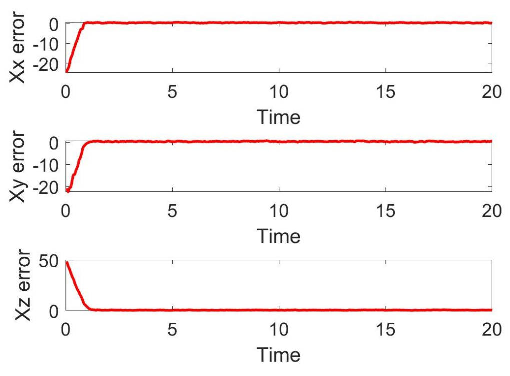


(b) Velocity Estimation

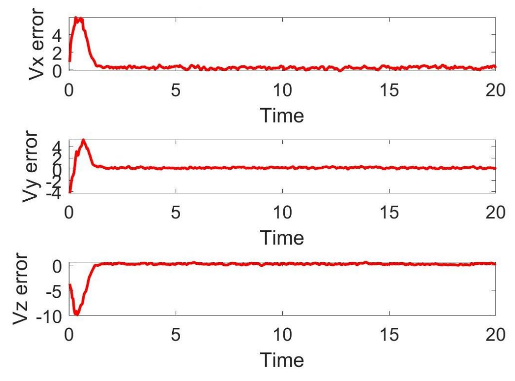
s

Figure 2: State Estimation Results

The estimation error is calculated by subtracting the estimate from the real value. From the error diagram, we can see that the magnitude of error reduces significantly, and the mean error goes to zero. This indicates a successful state estimation. The estimation error figures are shown below.



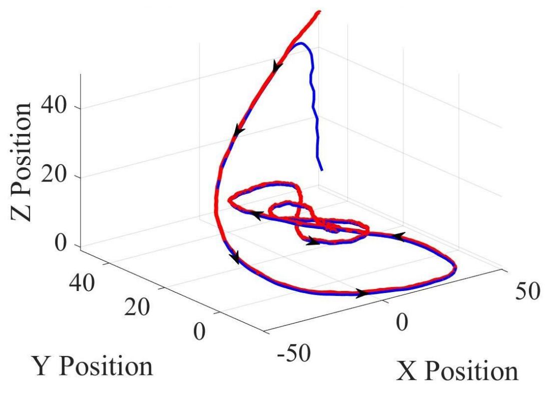
(a) Position Estimation Errors



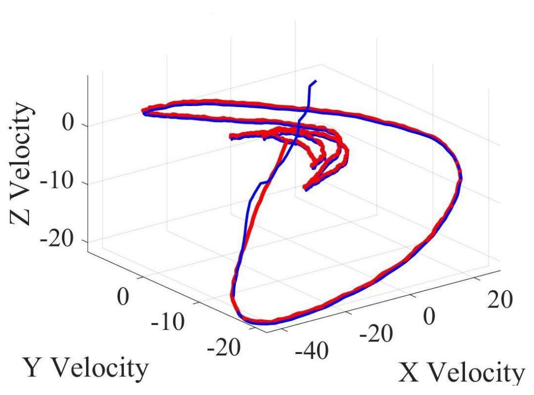
(b) Velocity Estimation Errors

Figure 3: Estimation Errors

By plotting our estimation results into 3-D figures, we can see that our estimations quickly converge to the real robot states. It is also clear on the 3-D graph that our feedback controller is successfully directing the robot to a fixed destination. The 3-D figures are shown below.



(a) 3-D Robot Position Estimation



(b) 3-D Robot Velocity Estimation

Figure 4: 3-D State Estimation and Control

5 Conclusions

In this project, we achieved two primary goals. First, we successfully simulated a feedback-controlled three-dimensional robot capable of navigating toward a predefined destination. Second, we implemented a generalized particle filter that accurately estimates the robot's internal states—namely, position and velocity—based on observation data generated by our custom observation model.

Our simulation results demonstrated that the feedback controller effectively guides the robot to its target location, and the particle filter provides state estimates that converge rapidly and accurately to the true values. This confirms the effectiveness of particle filtering for real-time state estimation.

6 References

Labbe, R. (2014). *Kalman and Bayesian Filters in Python* [Computer software]. GitHub. Retrieved from `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`