Analysing and Evaluating the Robustness of Particle Filter Tracking using Ms. Pacman

Sarvesh Rajkumar
sarvesh@kth.se

Sankeerth R. Prodduturi
srpr@kth.se

KTH Royal Institute of Technology

Abstract

This project focuses on utilizing the knowledge we obtained from the course, Applied Estimation(EL2320), to implement and analyze particle filter tracking in the game, Ms. Pacman. Tracking any complex system is vital to understanding its position in real-time and making appropriate decisions. The algorithm must also be robust enough to handle different scenarios like the classic case of a kidnapped robot. We have implemented the Sequential Monte Carlo Particle filter for tracking Ms. Pacman and analyzed its robustness across different scenarios. We have also extended the algorithm to track multiple agents that depend on each other's positions for their movement and found the algorithm to work well in this case.

1 Introduction

For any robot, tracking its state is important to make intelligent decisions further. Depending on the environment and what state we aim to obtain, we appropriately choose our sensors. However, we cannot completely rely on the measurements obtained from sensors as they are always subjected to errors and uncertainties that cannot be controlled or removed. These uncertainties differ between various environments and sensors and must be accounted for. On the other hand, a perfectly modeled motion model of the robot can also help track the state of the system. However, these motion models may also cause uncertainties where external factors can cause a building-up error, or in the case of different probable motions, we cannot choose which is the correct motion. Particle filter solves this issue by utilizing both the motion models and measurements to predict and correct the tracking of the system.

Ms. Pacman is an arcade video game by Atari. The user can control Pacman and their objective is to survive and obtain points. This game can be considered a classic 'cat-and-mouse' scenario where Pacman must strategically evade the ghosts while pursuing the food items to obtain points. We chose to implement and analyze the particle filter in this game as we use the respective colors for each entity for the observation model and the user's game objective for the motion model. The game also covers scenarios such as where entities move across from one side to the other and where there are overlapping entities for which we can evaluate the algorithm's robustness.

In the case of Ms.Pacman, the movement does not have any linearity. A user's command always varies and they can even move back and forth between points. We chose the particle filter as our method to track this randomized motion that the Kalman Filter fails to account for. Handling such non-linearities, non-Gaussian movements, irregularities, and multi-modal states is the inherent advantage that the particle filter has making it outperform the Kalman filter at the cost of computational complexity. [2].

1.1 Contribution

In this report, we discuss our implementation and analysis of the particle filter in the game to track Pacman. We have focused on the convergence of the particle filter in the case of different resampling methods and observation update methods. Certain scenarios such as occlusions between entities and moving across sides are the cases of kidnapping or teleportation in this game. We tested the algorithm's convergence in such scenarios. We further extended our algorithm to accurately track multiple agents whose motion models depend on the particle filter positions of other entities that we have estimated simultaneously.

1.2 Outline

The report starts with an evolution of the particle filter algorithm and some of the current widely used implementations. We then portray our implementation of the algorithm and the reasoning behind the algorithms we have chosen. We showcase our results for single-agent tracking and multiple-agent tracking using various resampling methods[3]. Different scenarios are used to evaluate the robustness of our algorithm and why it works well.

2 Background

2.1 Background

Tracking the states has always been an important part of any system. While the Kalman Filter tracks the states of linear systems and the Extended Kalman handles non-linear systems, they fail to account for randomized motion. The particle filter algorithm is based on the principles of the Bayesian filter that uses probabilistic models to estimate the state of the system. A particle represents the posterior distribution of the state of the system with a certain weight attesting to the observation that we obtain. Many such particles dispersed over the states then converge to the true state of the system based on the weights modified over each observation update. The resampling of the particles maintains the particles close to the true distribution of the state. Over each frame and time of the video of the Ms. Pacman game, we aim to track the agents' positions.

2.2 Literature review and related methods

The first implementation of Particle Filter was the Bootstrap filter [2]. It utilized modifying weights of particles based on the likelihood of the particles given the observations. The beauty of the particle filter was the resampling of particles which focused on replicating and giving importance to the highly weighted particles while under-representing the low weighted particles[1]. There are many variations of resampling today [3] which aim to solve problems like particle depletion, particle diversity, preventing system divergence, etc. Based on the algorithm and the purpose of our implementation, some resampling methods work better than others. Some of the newer methods integrate other filters such as the Kalman filter to improve the efficiency and accuracy of the filter. Another type of Particle Filter uses the Mean Shift Algorithm [4] which reduces the computational cost and increases efficiency by shifting particles towards the maximas. The type of tracking filter that we require also depends on the particular use case.

3 Algorithms and Implementation

3.1 Motion Model

In our project, our motion model was to estimate the movements of the various agents across each continuous frame. The motion models are based on the user's objective for Pacman, and the AI modeling for the ghosts. We represent each possible motion with probabilities based on which direction they are likely to move towards next and also on the allowable motions from the constraints of the map which will be explained in the next section.

For the single-agent tracking, we used the position of the ghosts for the motion model from the observation model. Whereas for multi-agent tracking, we used the best estimate of each entity for the motion model.

3.1.1 Pacman

As mentioned before, Ms. Pacman is a classic game of cat and mouse where the ghosts try to eat Pacman and Pacman aims to run away from them. We hence denoted an attraction field from each ghost by their Manhattan weights denoting how close the ghost is to Pacman.

Ignoring the map constraints for now, Pacman can move in 4 possible directions: up, down, right, and left. For each ghost, we multiply their Manhattan weights with the inverse of the differences in the respective coordinates of the ghost and Pacman. If the ghost is top left compared to Pacman, we multiply their weight by the inverses of the up direction based on the y coordinate difference and the left direction based on the x coordinate difference. Then we inverse this weighted sum for each direction indicating that a higher value showcases more possibility of motion direction.

Since between frames, a user may tend to continue along the same direction, based on the previous frame's observed motion, we increase the possibility of motion in that direction by 25%. Then based on the constraints of the wall, we split this probability of motion in the direction (that is not possible) equally, and add them to the other directions.

3.1.2 Ghosts

For the Ghosts, their objective is to eat Pacman. The motion model for ghosts is similar to Pacman's except that we don't take the inverse of the finally computed sum as those initially weighted sums indicate the likeliness to go in that direction. Then we perform the same steps of incrementing the weight in the direction of the previous action and redistribution based on the walls present.

Algorithm 1 Pacman Motion Model

1: Calculate the Manhattan Weights of each ghost with respect to Pacman
2: Calculate a Weighted sum for each ghost in all directions
3: Compute the inverse to obtain the most possible direction
4: Increment weight by 25% based on the previous action of Pacman
5: Normalize probabilities
6: Redistribute not possible direction probabilities (40% to orthogonal, 20% opposite)

3.2 Constraints from the wall

Each entity cannot move in all possible directions in every frame. The constraints of the environment, i.e. the walls, only allow certain possible directions of motion. Since the environment is static, we had to only evaluate the map once to identify the walls. The walls were identified using a color mask of saturated blue color. The lower limit in the HSV(Hue, Saturation, and Value) range is $(120, 100, 30)$ and the upper limit is $(130, 255, 255)$. Given the Identified environment, based on the best estimate of the position of each entity from the previous frame, we identify a small window of the environment surrounding the entity to decide along which directions the movement is blocked. This is passed as a dictionary when to the motion model.

3.3 Measurement Model

Each entity and the environment, have a different color palette. This makes it relatively easy to identify through a color mask. For the observation model, we use differing upper and lower ranges of HSV values for each entity. The HSV lower and upper ranges behave such that any color that has a hue, saturation, and value which are all between their respective lower and upper ranges will become the mask for identifying the entity. Red has two ranges as it goes over and around the Hue scale and both these ranges are considered for the red ghost.

Entity	HSV lower and upper Ranges
Pacman	(25,100,100) - (35,255,255)
Red ghost Range 1	(160,100,100) - (180,255,255)
Red ghost Range 2	(0,100,100) - (5,255,255)
Pink ghost	(110,20,100) - (160,110,255)
Cyan ghost	(80,100,100) - (100,255,255)
Orange Ghost	(5,20,100) - (20,255,255)

Table 1: Hue Saturation and Value lower and upper ranges

For each entity, the mask turns into a collection of blobs where each blob is a group of pixels. By taking the contours of each blob and their moments, we obtain the centers of each blob. Since some other background elements may correspond to the same colors, we only appended those centers for which the blob has an area over a particular threshold.

When for the respective entity, no blob has been detected, then we get the case of kidnapping. In this case, we reduce the weights of each particle. We then add some measurement noise to the centers to ensure robustness in our algorithm.

We used an associative algorithm where for each particle, we utilize a matching threshold based on its distance from the observed center of the entity. If the particle falls within this range, then its weight is incremented based on how close it is to the observed center. If the particle is out of the range, then the weight is reduced. The weights are then normalized and particles are resampled based on the updated weights.

3.4 Particle Filter Implementation

We implemented the particle filter on a dataset collected by ourselves online. We utilized Python for the implementation and used packages such as OpenCV for processing the image and Numpy for array manipulation. There are two main classes, *Extract* which corresponds to obtaining the relevant observatory information from the frame, and *ParticleFilter* which contains the implementation of the algorithm.

Algorithm 2 Particle Filter

1: **Input:** Particles $\{x_{t-1}^{(i)}, w_{t-1}^{(i)}\}_{i=1}^{N}$ from the previous timestep, observation y_t.
2: **Output:** Updated particles $\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^{N}$.
3: **for** $i = 1$ to N **do**
4: **Motion Model:** Predict the state through sampling
5: **Observation Model:** Compute the likelihood of the observation
6: **Weight Computation:** Compute the unnormalized weight
7: **end for**
8: **Normalize Weights:**
9: **Return:** Updated particles $\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^{N}$.

4 Experimental Results

4.1 Single Agent Tracking - Ms.Pacman

4.1.1 Testing Various Resampling Methods

We first tested the resampling methods in the case of tracking a single agent. We implemented four of the generally used resampling methods and analyzed their performances. The methods were multinomial,

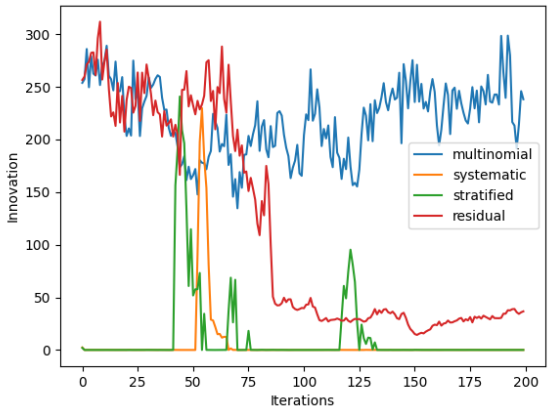

Figure 1: Innovation over Time for various resampling methods

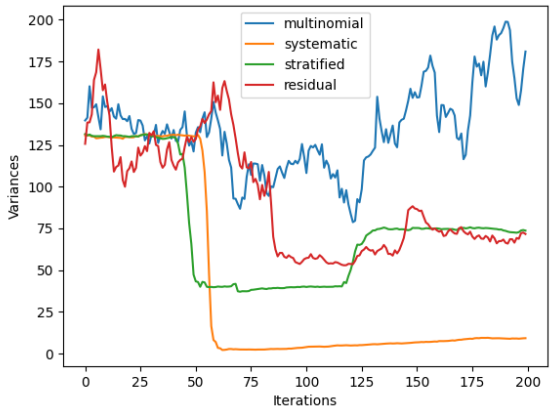

Figure 2: Particle Variance over Time for various resampling methods

systematic, stratified, and residual resampling. Our initial testing was with 100 particles to evaluate the performances of each method.

In Fig. 1 we observe the innovation between the particles after the motion update and after resampling. Systematic and stratified sampling seem to converge much faster than the other methods. Residual sampling converged a bit slower but multinomial doesn't converge at all in this case.

In Fig. 2 we observe the particle variance over time. Systematic resampling converges to a very low variance compared to the other resampling methods which have a decent variance. By analyzing both figures simultaneously, when the innovation is low, it indicates that there aren't many grave changes to the particles from the resampling indicating that the particles are in line with the observation update. When the variance is low in this case, it indicates that the particles have converged to an estimate. Even drops in the innovation values indicate that the particles are converging to an estimate.

Overall we found stratified to work best as even though it has a higher variance than systematic, it maintains diversity around the center. Hence for random movements, more particles will be closer to the observation we get and it can track it better. While systematic converges to the correct estimate, it is more compact around the center and doesn't handle sudden movement changes as well.

4.1.2 Testing Different Number of Particles

We tested the innovation and compared them against different numbers of particles too. In Fig. 3, we see that for 10 particles, the innovation is 0 up until the 175th iteration. The random shifting of particles finally catches up to Pacman. The particles can't converge to the state of Pacman as it doesn't have enough particles to cover the image. For 100 and 1000 particles, the particles pretty quickly. We see the spike in innovation for 1000 particles right in the beginning as it would have already covered Pacman. Furthermore, we see more spikes in the middle of the graph for 1000 particles which attest to the randomness of the Pacman motion and that it is captured well by 1000 particles.

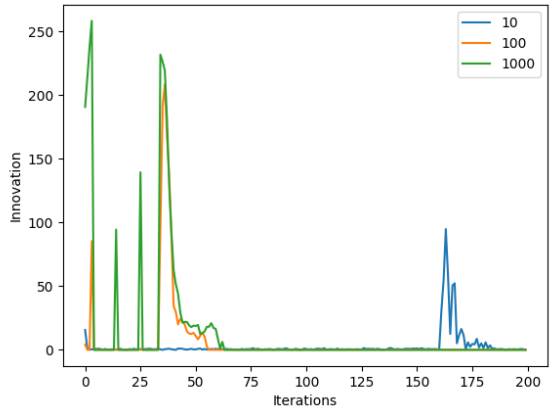

Figure 3: Innovation over time for Different Number of Particles

4.1.3 No Detection Case

During certain scenarios, Pacman can become undetected. This can either happen when Pacman is occluded by another entity, teleports from one end to another, or when it eats a ghost and disappears for a

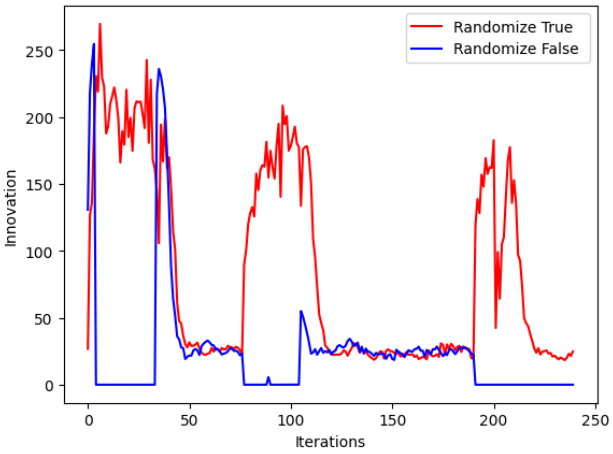

Figure 4: Innovation of particles with and without randomize when no Detection

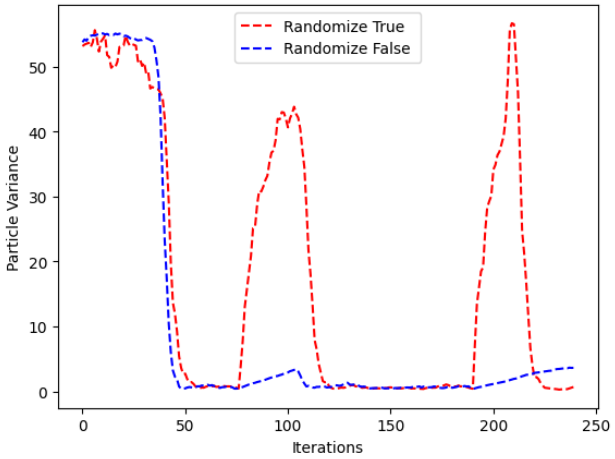

Figure 5: Variance of particles with and without randomize when no Detection

bit. We consider this as the kidnapping case of the agent. In Fig .4 and Fig. 5 we observe the innovation and variance in such scenarios. We experimented with two algorithms, one in which the weights are only decreased when the entity is not detected and another where the particles are resampled randomly across the map when not detected.

The peak in the middle for randomization is true corresponds to when Pacman eats a ghost and disappears for a bit. In this scenario, both the algorithms cause the particles to converge to Pacman once it reappears as Pacman does not displace from its position. The variance increases for randomization as particles are distributed across the map and we see the corresponding peak in innovation which then settles around a value. The peak in the end for randomization is true corresponds to when Pacman teleports from one end to the other. However, in this scenario, randomization causes the particles to converge to Pacman once Pacman reappears at the other end. The innovation and variance increase and then settle to a value. With no randomization, the innovation is at 0 and the variance starts to increase.

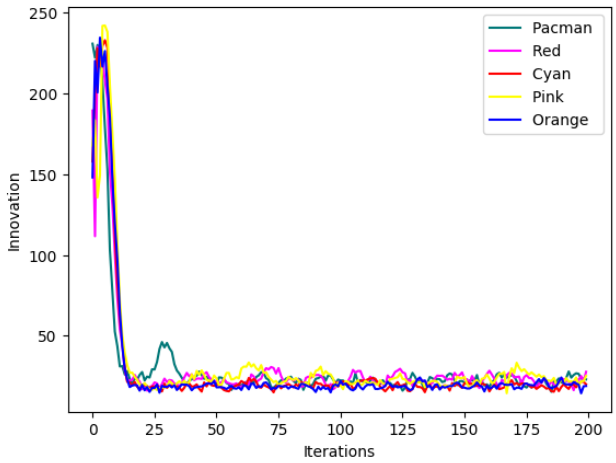

Figure 6: Innovation of particles for all Entities over time

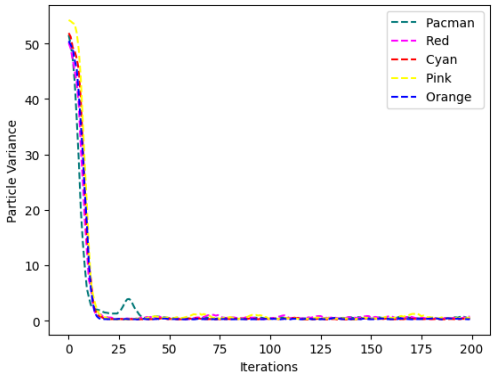

Figure 7: Variance of particles for all Entities over time

4.2 Multiple Agent Tracking

4.2.1 Tracking of all Agents

In Fig. 6 and Fig. 7 we can see that the algorithm perfectly tracks all the entities over time. The small peaks indicate the random motion of the entity to which the particles converge immediately after. A screenshot of the tracking is shown in Fig. 8 with the best estimate being the colored circle on the entity.

Figure 8: All entities Tracked

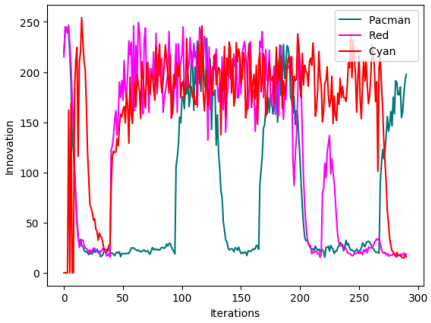

Figure 9: Innovation of Entities' particles in Ghost Respawn Case

4.2.2 Ghost Respawn Case

At one point in the game, all the ghosts turn blue. The detection of ghosts fails until Pacman eats the ghosts and they spawn back in the center, or the timer runs out. This is also a case of kidnapping of the ghosts. In Fig. 9 and Fig. 10 we see the innovation and variance of particles belonging to Pacman, Red Ghost, and Cyan Ghost as they are the interest points. We again utilized the randomization of particles during no detection of an entity. The beginning of the peaks in innovation and variance for the ghosts' particles correspond to when the ghosts turn blue. The peaks in innovation and variance for Pacman's particles correspond to when Pacman eats the ghosts and Pacman disappears for a bit. Once the ghosts spawn again, the variance and innovation drop. The small peak for the Red ghost corresponds to an occlusion.

5 Conclusions

We have implemented a particle filter for the video game Ms.Pacman. Considering the game's objective, a motion model was built to satisfy the game standards. The evaluation portrayed the particle filter's reliability in such $2D$ state tracking. Several resampling methods and their respective convergences were assessed for this particular game. We also tested our model against common problems in systems such as kidnapping and its robustness against such scenarios in the game of

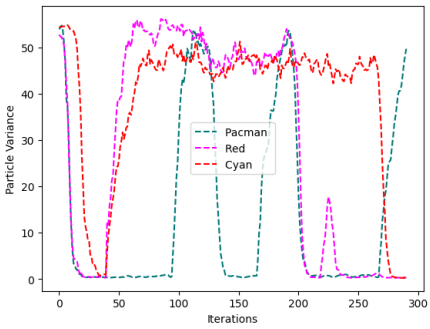

Figure 10: Variance of Entities' particles in Ghost Respawn Case

Ms. Pacman. We used innovation between particles and the particle variance to evaluate the convergence of the filter to the true distribution of the entity. By extending our implementation to multiple agents, we found the algorithm to be robust in this case as well.

This algorithm can further be extended to a joint particle filter instead of multiple filters. The motion model of Pacman can also be modified to seek food particles or chase the ghosts when they turn blue. Overall, the particle filter has proven to be a very robust algorithm in tracking.

References

[1] Simon Godsill. Particle filtering: the first 25 years and beyond. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7760–7764, 2019.

[2] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET, 1993.

[3] Jeroen D. Hol, Thomas B. Schon, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 79–82, 2006.

[4] E. Maggio and A. Cavallaro. Hybrid particle filter and mean shift tracker with adaptive transition model. In *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 2, pages ii/221–ii/224 Vol. 2, 2005.

[5] Sankeerth and Sarvesh. Simulation videos for various scenarios. `https://drive.google.com/file/d/12L1sPP9xrwwHJEV97sHhNK6vn4bVrgyY/view?usp=sharing`, 2025.

6 Appendix

Figure 11: Screenshot of Ghost Respawn with Randomization OFF

Figure 12: Screenshot of Ghost Respawn with Randomization ON

Figure 13: Pacman

Figure 14: Red Ghost

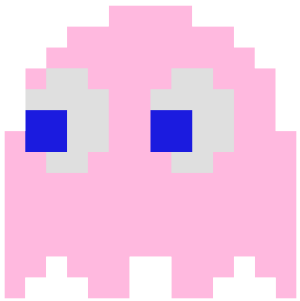

Figure 15: Pink Ghost

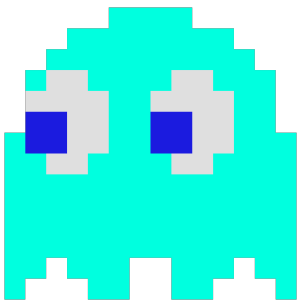

Figure 16: Cyan ghost

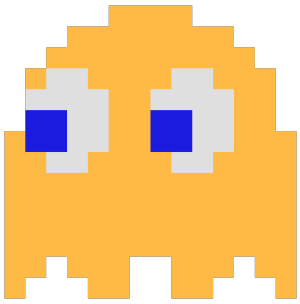

Figure 17: Orange ghost