Particle Filter Implementation

Guevara Moedano Jonathan Ernesto
Facultad de ciencias, UNAM
jonathagm@ciencias.unam.mx

December, 2023

Keywords Particle filter, Hidden Markov Model (HMM), Brownian motion, Wilson algorithm, random mazes, localization, simulation, trajectories, computational perception.

Abstract This project focused on the implementation and analysis of a particle filter. The effectiveness of the filter was evaluated in trajectory estimation, initially in a controlled environment and subsequently in randomly generated mazes using the Wilson algorithm. The impact of the number of particles and the level of noise on filter accuracy was explored. Results demonstrated the adaptability of the filter to dynamic and unknown environments, highlighting its applicability in robotics and autonomous systems.

Introduction

In the realm of robotics, computer vision, and signal recovery, the development of localization algorithms is essential to enable autonomous systems to understand and navigate changing environments. One of the most effective and versatile approaches in this field is the particle filter, a computational technique that has been successfully used in a variety of applications, ranging from autonomous vehicles to object tracking systems.

The particle filter finds numerous applications in various domains, primarily focusing on object tracking in video, speech recognition, and navigation of autonomous robots, showcasing its ability to address complex problems in dynamic environments.

This project aims to contribute to the advancement of research in the field of localization by providing a robust and accessible implementation of the particle filter, facilitating its application in a variety of practical scenarios.

Throughout this report, the theoretical and practical aspects of the particle filter will be detailed, along with the results of tests and experiments conducted.

The source code is available in the file pfilter.py, and can be found here.

Objectives

The objective of this project is to create a robust particle filter that can be applied in real-life situations and to comprehend its operation. To achieve this goal, we plan to undertake the following steps:

1. Develop a particle filter to estimate a real trajectory within a closed square.
2. Increase the difficulty of the environment and the noise with which we estimate the position.
3. Apply the particle filter in complex situations and compare its performance.
4. Analyze the effectiveness and efficiency of the filter in different scenarios and under various conditions.

Methodology

1. Hidden Markov Model (HMM)

The Hidden Markov Model (HMM) is a statistical model that describes a stochastic process with unobservable states evolving over time. It consists of two processes: the hidden state process and the observation process.

The HMM (Hidden Markov Model) is defined by:

- **Hidden States** ($\{X_n\}$)**:** A finite set of unobservable states evolving over time following a Markov chain.
- **Observations** ($\{Y_n\}$)**:** A finite set of observations dependent on the hidden states.

A requirement of the Hidden Markov Model is that the process $\{X_n\}$ is a Markov chain with a finite state space.

Formally, this can be defined as:

We say that the pair (X, Y) is a **Hidden Markov Model** if $X = \{X_n\}$ is a Markov chain and $Y = \{Y_n\}$ is a stochastic process that satisfies:

$$P(Y_n|X_n, X_{n-1}, \ldots, X_1) = P(Y_n|X_n)$$

The above equality indicates that the probability of observing Y_n at time n does not depend on the complete sequence of past hidden states $X_{n-1}, \ldots, X_1$, but only on the hidden state at that time X_n.

2. Particle filter

In the context of the Hidden Markov Model, the application of the particle filter allows estimating the hidden state (the process $\{X_n\}$) from the process $\{Y_n\}$. This process is carried out through the following stages:

1. **Particle Initialization**

   Each particle represents a potential state of our system. The process begins with the generation of a set of particles $X_0^{(i)}$, each representing a possible instance of the hidden state at the initial time.

2. **State Prediction**

   The state prediction is performed using the transition probabilities of the HMM, shifting each particle according to these probabilities. For each particle i:

$$X_t^{(i)} \sim P(X_t|X_{t-1}^{(i)})$$

3. **State Update**

   The update is based on real observations and the emission probabilities of the HMM. Each particle evaluates how well its predicted state matches the current observation. Particle weights are adjusted according to this match:

$$w_t^{(i)} \propto P(Y_t|X_t^{(i)})$$

4. **Weight Normalization**

   Particle weights are normalized to ensure they sum to 1 and define a probability measure:

$$\hat{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^{N} w_t^{(j)}}$$

5. **Resampling**

   To prevent filter degeneration, resampling is performed. New particles are selected with a probability proportional to their adjusted weights, ensuring the diversity of the set.

6. **State Estimation**

   The final state estimation can be obtained by averaging the states or combining the particles according to their normalized weights:

$$\hat{X}_t = \sum_{i=1}^{N} \hat{w}_t^{(i)} \cdot X_t^{(i)}$$

   These steps are repeated at each time step, allowing the particle filter to provide a robust and adaptive approximation of the hidden state of the system in the presence of uncertainty and nonlinearities.

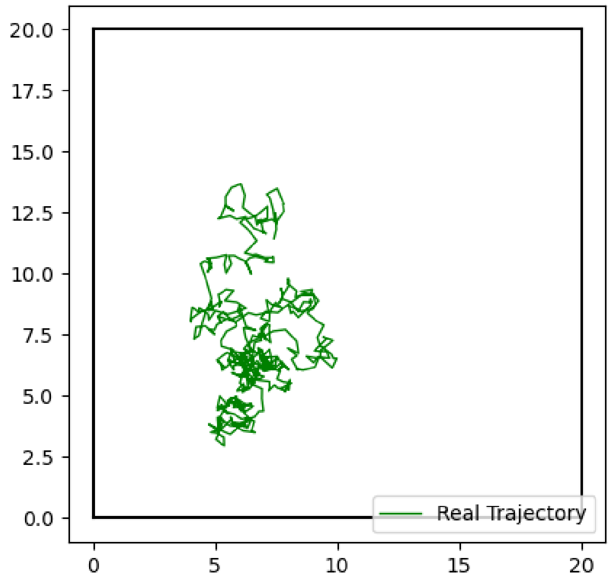
Implementation of the filter

First, we import the libraries:

[24]:
```
import numpy as np
import random
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from pfilter import *
```

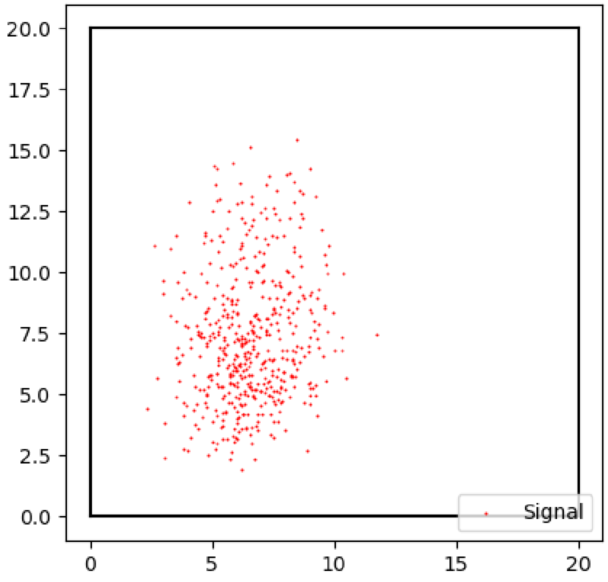
With the purpose of applying the particle filter, it is imperative to have the stochastic process X_n. In the context of this study, we have chosen the trajectory of a Brownian motion as our main process, because it is characterized by continuous stochastic behavior without a particular trend, making it a suitable choice for modeling the variation of a continuous trajectory in $\mathbb{R}^{\not{2}}$.

[25]:
```
maze = np.array([[(True, True) for _ in range(20)] for _ in range(20)])
position, dx, dy, media, obs, iteraciones = brownian_motion_filter(.1, 500,  maze, 500)
draw_maze(maze, create = False, trajectory = position)
```



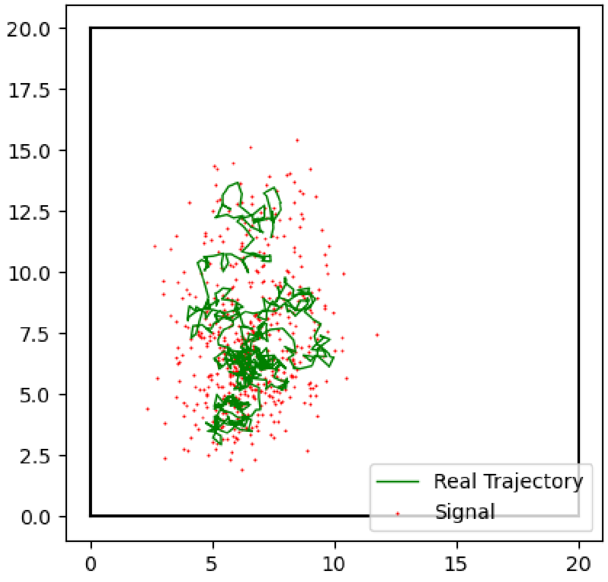
The subsequent phase involves obtaining a process Y_n that depends on the process X_n. To achieve this, we will introduce noise to the trajectory generated in the previous stage. In this way, the process Y_n can be visualized as follows:

[26]:
```
draw_maze(maze, create = False, observacion=obs)
```



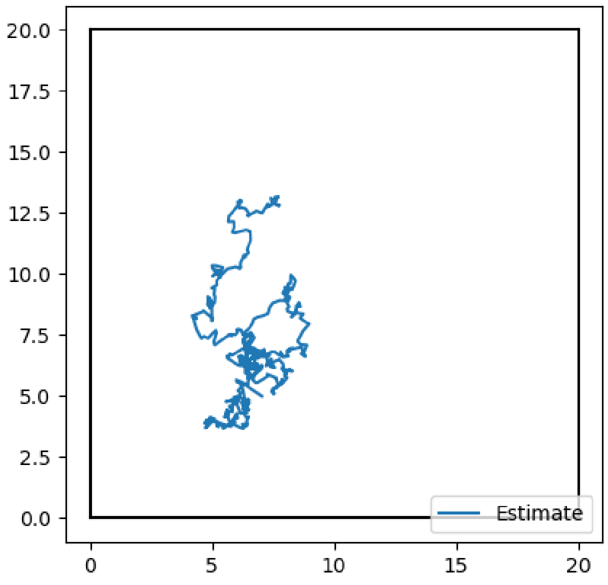
We perform a comparative analysis between the emitted signal Y_n and the trajectory X_n.

[27]:
```
draw_maze(maze, create = False, trajectory=position, observacion=obs)
```



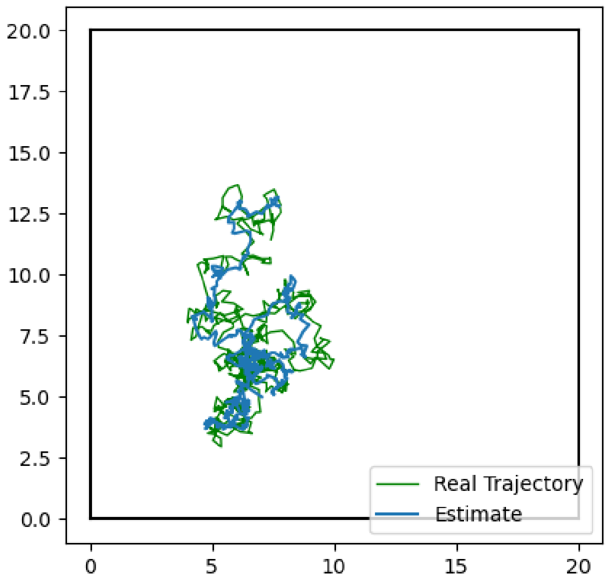
In the given context, the green process symbolizes the trajectory targeted for estimation through the application of the filter, while the red signals denote pertinent elements. Subsequent to the implementation of the particle filter, we derive the ensuing estimation of the process X_n, visually depicted in blue. Consistently, 500 particles will be employed for all estimations.

[28]:
```
draw_maze(maze, create = False, estimate= media)
```



Presently, we superimpose the two trajectories: the genuine trajectory in green, targeted for estimation, and the estimation derived through the particle filter in blue. This juxtaposition furnishes a lucid visualization of the precision and efficacy of the filter in approximating the desired trajectory.

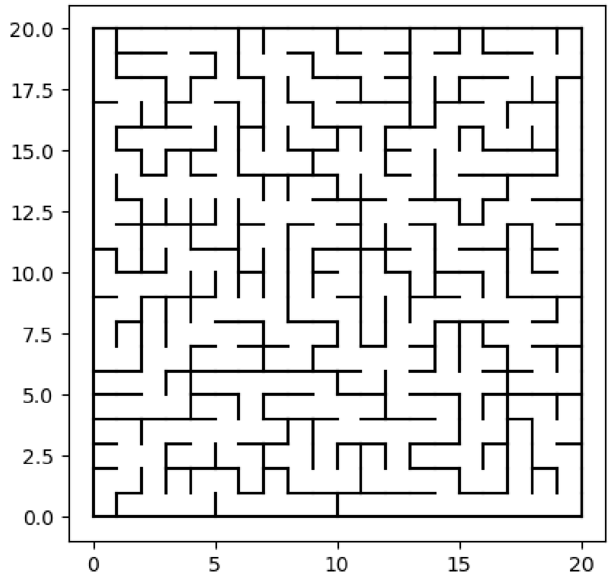
[29]:
```
draw_maze(maze, create = False, trajectory= position, estimate= media)
```



The estimated trajectory is articulated as a substantiated approximation of the genuine trajectory, attesting to the efficacy of the filter within this specific context. Subsequently, the implementation of the Wilson algorithm (refer to the appendix) will be undertaken to generate random mazes. This strategic undertaking is poised to facilitate experimentation within more authentic environments, thus augmenting the depth of our system analysis. The incorporation of randomness via the Wilson algorithm is anticipated to foster the exploration of intricate scenarios, thereby amplifying the resilience of our approach.

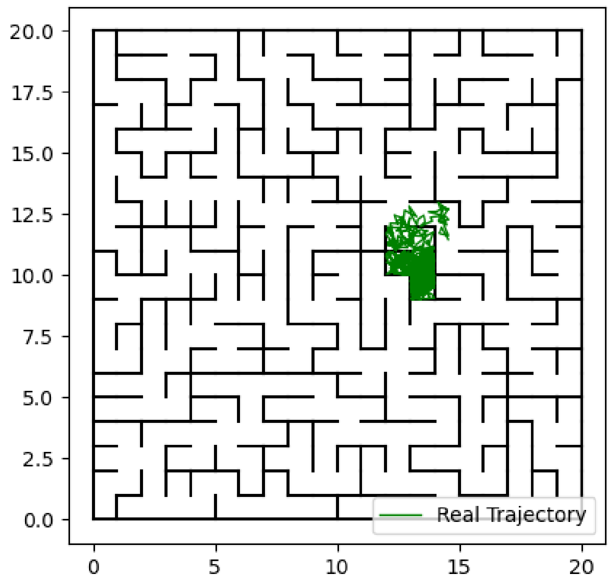
Subsequently, we offer a visual representation of the outcomes derived from the implementation of the Wilson algorithm.

[30]:
```
maze = generate_maze(20, True)
```



Subsequently, we shall undertake the simulation of the identical Brownian motion as previously considered, albeit now constrained within the generated maze. The introduction of the maze introduces an additional element to the simulation, whereby the movement adheres to the constraints imposed by the labyrinthine structure. This methodological approach affords an exploration of the impact of obstacles on the trajectory of Brownian motion and its behavioral characteristics within a maze-like environment. Through this simulation, our objective is to attain a more thorough and intricate comprehension of the interplay between the stochastic process and the topological features of the maze.

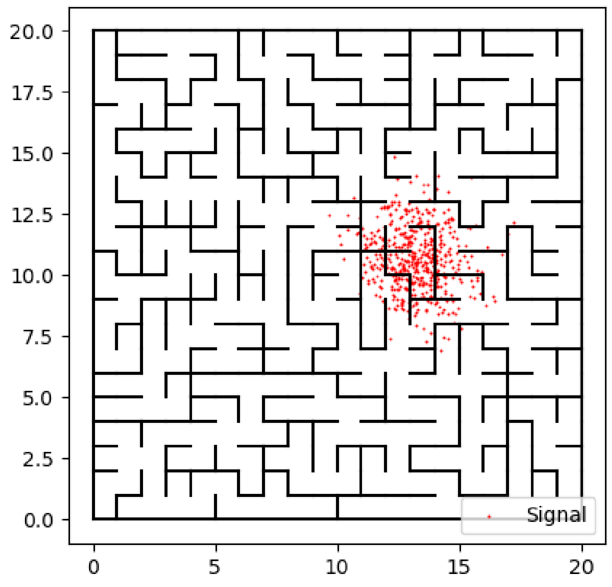
[31]:
```
position, dx, dy, media, obs, i = brownian_motion_filter(.1, 500,  maze, 500)
draw_maze(maze, create = False, trajectory = position)
```



It is pertinent to emphasize that, in this instance, the simulated Brownian motion can be interpreted as the trajectory that a person would follow while navigating the streets of a city. The analogy between the stochastic behavior of Brownian motion and urban movements effectively reflects the unpredictable and directionless nature of movements in an urban environment.

Now, let's plot the corresponding signal:

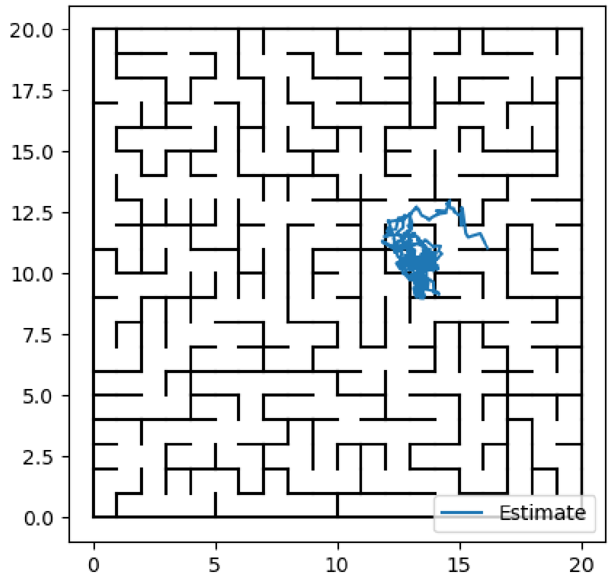
```
[32]: draw_maze(maze, create = False, observacion = obs)
```



At this scale, it is evident that the amount of noise in the simulation is significant. This phenomenon can be likened to real-life situations, such as the common experience on Google Maps when we are in proximity to many buildings, where the GPS signal tends to be imprecise.

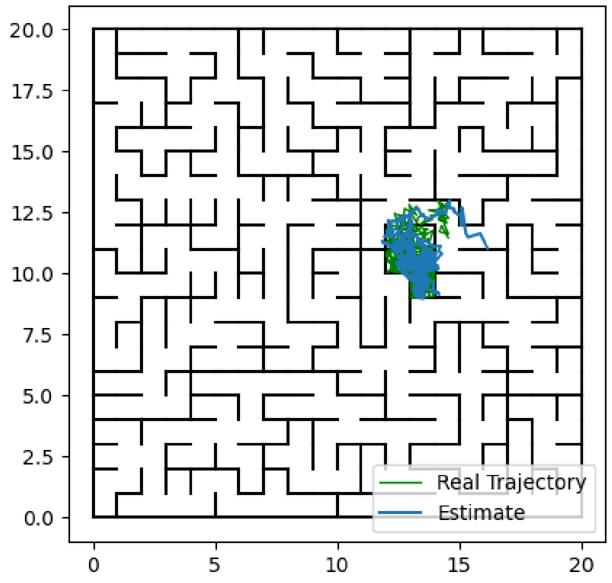
The results provided by the particle filter are presented as follows.

```
[33]: draw_maze(maze, create = False, estimate=media)
```



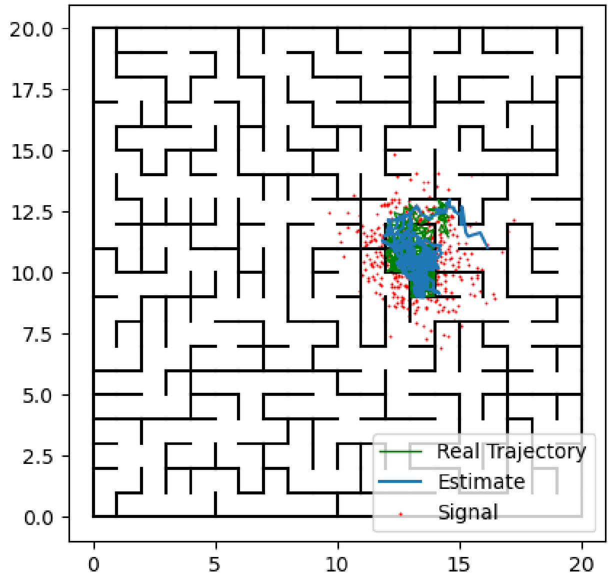
Upon superimposing the real trajectory with the estimated one, we obtain:

```
[34]: draw_maze(maze, create = False, estimate= media, trajectory= position)
```



The three elements together appear as follows:

[35]:
```
draw_maze(maze, create = False, estimate= media, trajectory= position, observacion=obs)
```



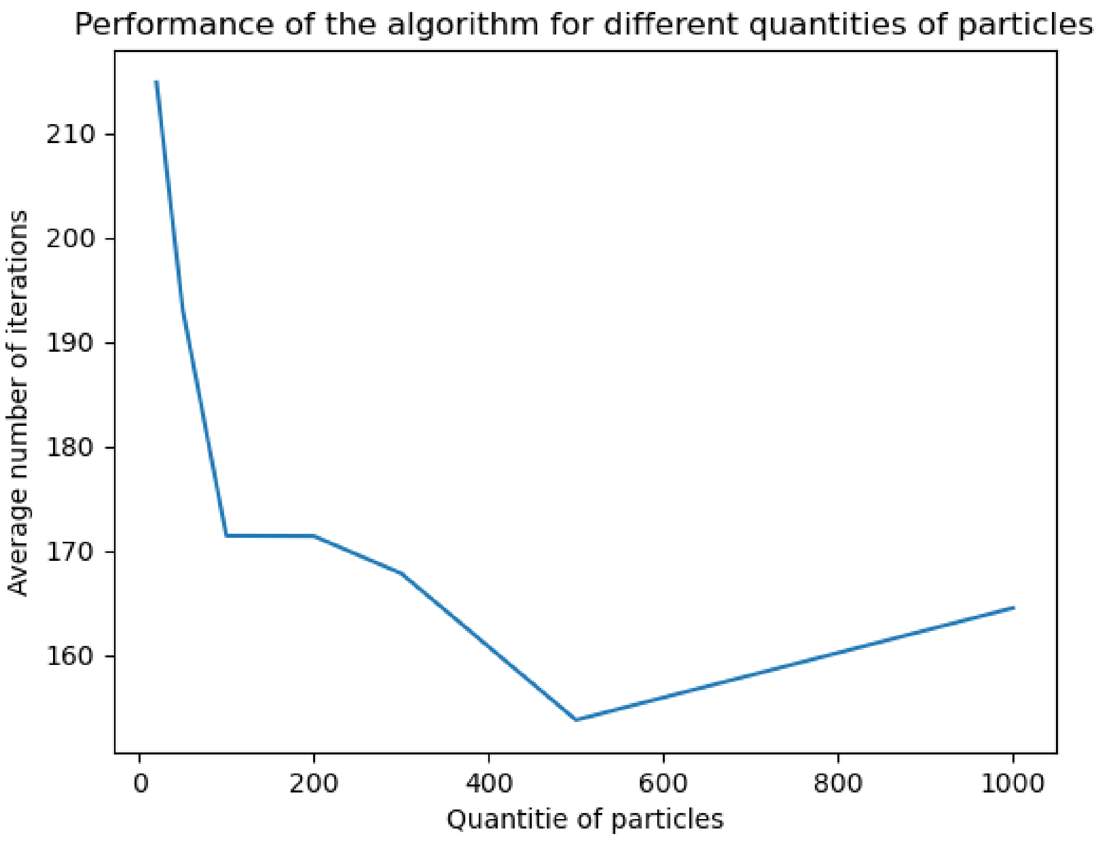
Results

We will conduct a comparison to determine how many iterations are necessary until the difference between the estimated position and the real position is less than 0.05 for different numbers of particles.

[14]:
```
maze = generate_maze(20)
promedio = []
cantidad_de_particulas = [20, 50, 100, 200, 300, 500, 1000]
promedios = []

for k in cantidad_de_particulas:
    p = 0
    for i in range(500):
        position, dx, dy, media, obs, it= brownian_motion_filter(.1, 500,  maze, k,␣
 ↪stop = True)
        p += it
    promedios.append(p/500)
```

[36]:
```
plt.plot(cantidad_de_particulas, promedios)
plt.xlabel('Quantitie of particles')
plt.ylabel('Average number of iterations')
plt.title('Performance of the algorithm for different quantities of particles')
plt.show()
```



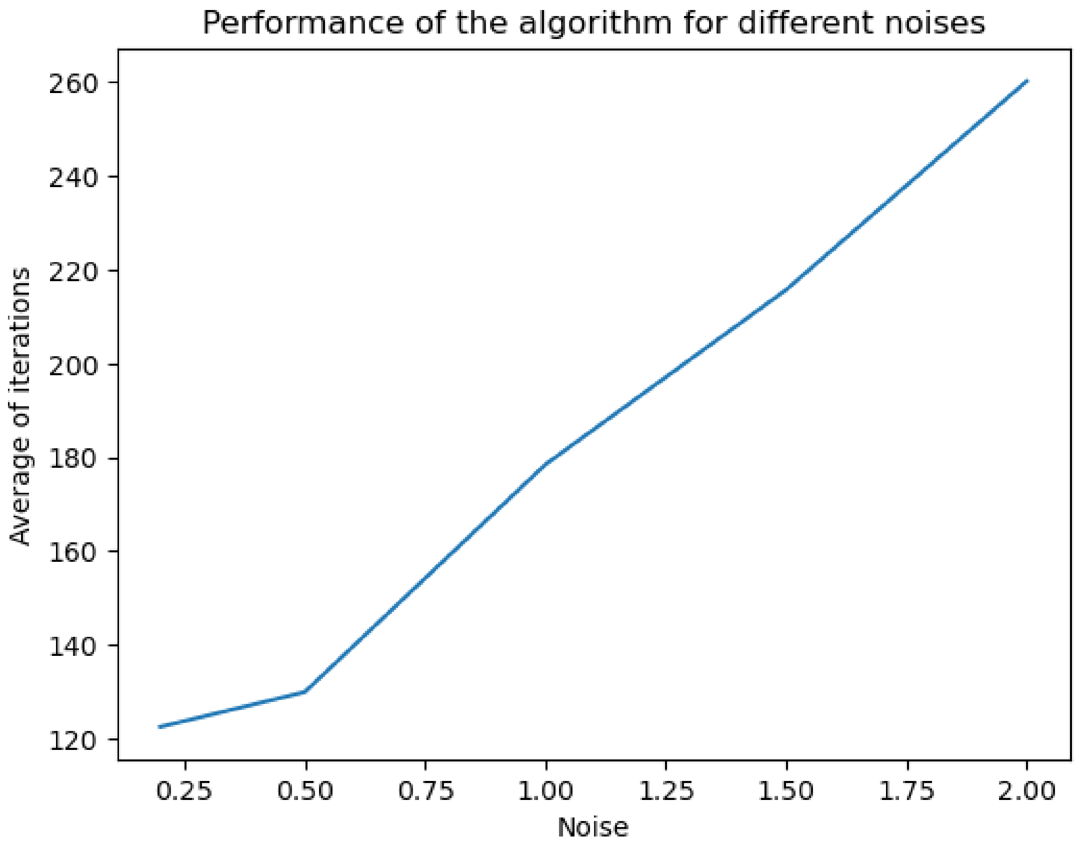
It is noteworthy that, in this case, there is an improvement with an increase in the number of particles. However, it is important to note that this benefit comes with an increase in the computational load of the algorithm. The enhanced performance suggests that a higher number of particles positively contributes to the accuracy of the estimations, but at the same time, it is crucial to consider the associated computational cost.

Now, we are going to use different quantities of noise:

[20]:
```
maze = generate_maze(20)
promedio = []
cantidad_de_ruido = [.2, .5, 1, 1.5, 2]

for k in cantidad_de_ruido:
    p = 0
    for i in range(300):
        position, dx, dy, media, obs, i = brownian_motion_filter(.1, 500,  maze, 500,␣
 ↪ruido = k, stop = True,)
        p += i
    promedio.append(p/300)
```

[37]:
```
plt.plot(cantidad_de_ruido, promedio)
plt.xlabel('Noise')
plt.ylabel('Average of iterations')
plt.title('Performance of the algorithm for different noises')
plt.show()
```



We observe that the amount of noise appears to follow a linear behavior; as the noise level increases, the number of iterations needed to reach the specific position also increases. This phenomenon suggests a direct relationship between the magnitude of the noise and the complexity of the estimation process, emphasizing the sensitivity of the filter to disturbances introduced by noise in the simulation of movement.

Conclusion

This project focused on the implementation and analysis of a particle filter, a key technique in the fields of robotics, computer vision, and signal recovery. Through the combination of the Hidden Markov Model (HMM) and the particle filter, we efficiently estimated trajectories in complex and noisy environments.

The obtained results demonstrated the effectiveness of the particle filter in estimating trajectories in an initially controlled environment. By introducing the Wilson algorithm to generate random mazes, we were able to evaluate the filter's performance in more realistic scenarios. The filter's ability to adapt to dynamic and unknown environments was evident when simulating Brownian motion within these mazes, where uncertainty and noise were more pronounced.

However, we found a drawback of the filter is that, when using relatively large numbers of particles, processing times tend to be somewhat lengthy, which in more complex environments may prove inefficient.

Furthermore, we analyzed the impact of the number of particles and the level of noise on the filter's accuracy. We found that an increase in the number of particles improved accuracy, but with diminishing returns as the quantity increased. Additionally, we observed that a higher level of noise affected the filter's ability to accurately estimate the trajectory.

Appendix

Wilson's Algorithm for Random Mazes:

Wilson's algorithm is a highly efficient method for creating random mazes based on the concept of random walks. Unlike other algorithms such as Prim or Kruskal, which construct the maze through incremental addition of passages, Wilson adopts a more dynamic and probabilistic approach.

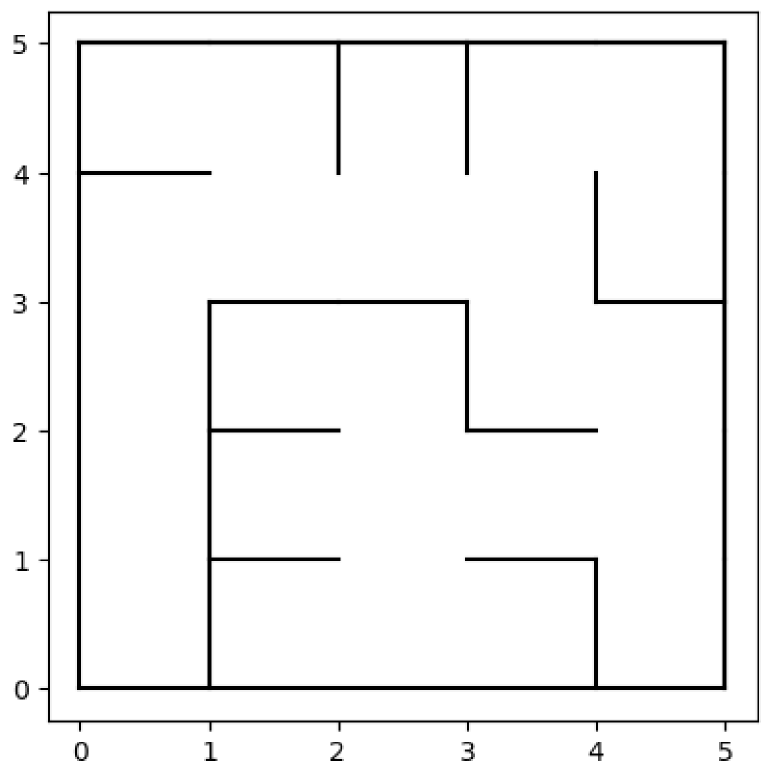
The following are the steps to carry out the algorithm:

1. **Initialization:**
   - A random cell in the maze is selected and marked as the starting point of the path.
2. **Random Walks:**
   - Another unmarked cell is randomly chosen as the starting point for a random walk.
   - The random walk continues until reaching a marked cell. During the walk, each visited cell is marked. This random walk acts as an "explorer" seeking to connect with the existing path.
3. **Loop Removal:**
   - Once the random walk has found and connected with the main path, any generated loops are eliminated. This is achieved by retaining only the main path, thereby avoiding redundancies and unnecessary cycles.
4. **Process Repetition:**
   - The process of randomly selecting unmarked cells and random walks is repeated until all cells in the maze are marked. Each iteration adds a new path to the maze, and repetition ensures complete exploration of the maze space.

This algorithm, although seemingly simple, results in mazes with intricate structures and complex paths. It ensures that all paths in the maze are interconnected. The inherent randomness in random walks contributes to the generation of unique mazes in each execution. Additionally, loop removal ensures the efficiency and structural coherence of the final maze.

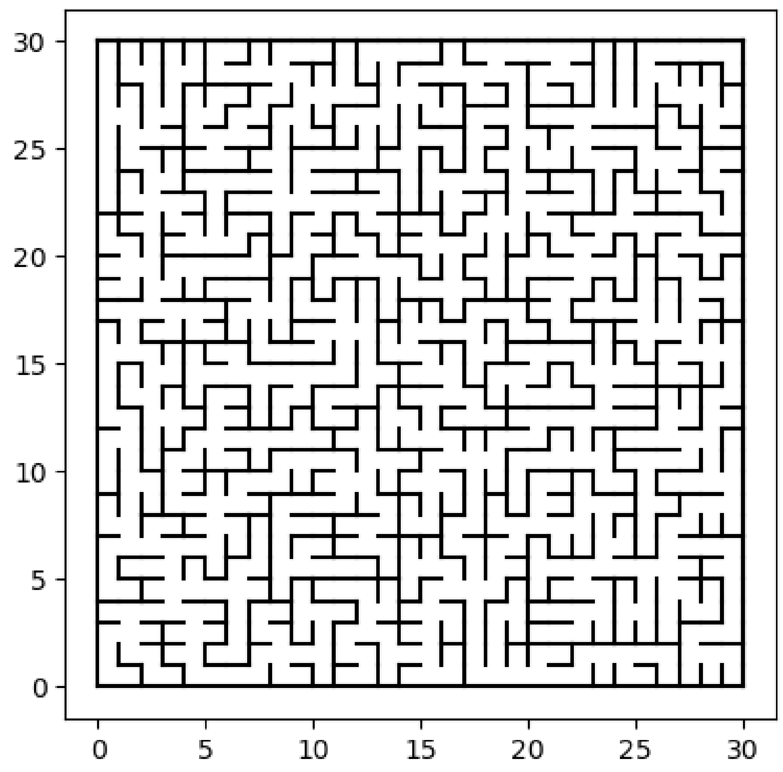
To assess the performance of the Wilson algorithm in maze generation, we conducted tests with a maze of size 5x5. Below, we present the visually obtained results:

```
[ ]: maze1 = generate_maze(5, True)
```



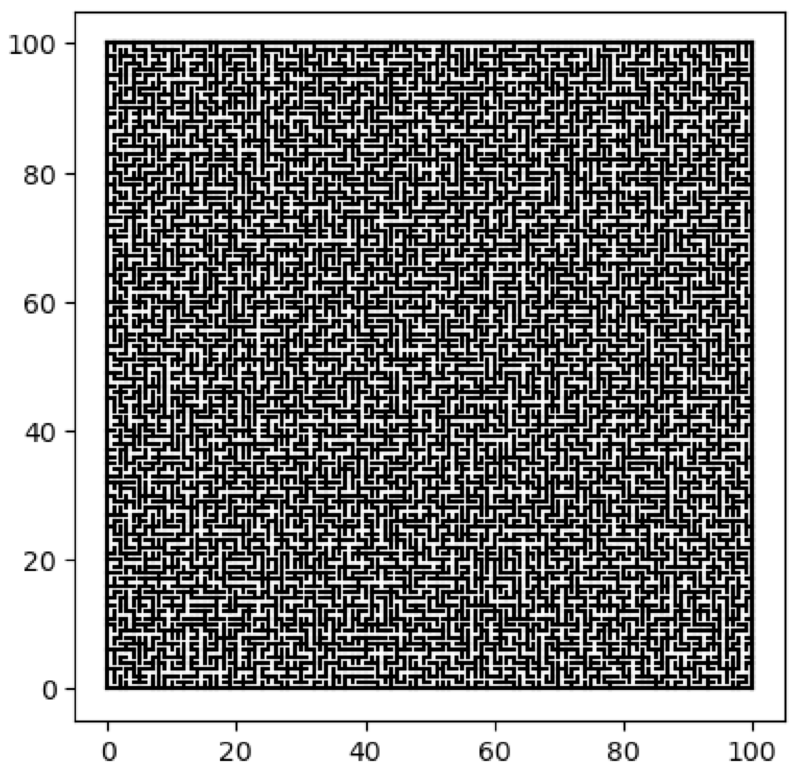
We tested the algorithm for mazes of size 30x30:

```
[ ]: maze2 = generate_maze(30, True)
```



Size 100x100:

```
[ ]: maze3 = generate_maze(100, True)
```



References

- Del Moral, Pierre (1996). *Non Linear Filtering: Interacting Particle Solution.* Markov Processes and Related
- Doucet, A.,et al. (sf.) *An Introduction To Sequential Monte Carlo Methods*, Oxford University.
- Doucet, A.; Johansen, A.M. (2008). *A tutorial on particle filtering and smoothing: fifteen years later* . Technical Report
- Haug, A.J. (2005). "A Tutorial on Bayesian Estimation and Tracking Techniques Applicable to Nonlinear and Non-Gaussian Processes" The MITRE Corporation, USA, Tech. Rep.
- Wills, Adrian G.; Schön, Thomas B. (2023). *Sequential Monte Carlo: A Unified Review.* Annual Review of Control, Robotics, and Autonomous Systems. 6: 159–182. ISSN 2573-5144.
- Wilson, D. (sf.) *Generating random spannin trees more quickly than the cover time* MIT, department of Mathematics