Changing the Degrees of Freedom for a particle filter tracking algorithm based on prior knowledge

Koen Snijder

Supervisors: Dr. R. R. V. Prasad, K. Kroep

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Koen Snijder
Final project course: CSE3000 Research Project
Thesis committee: Dr. R. R. V. Prasad, K. Kroep, M. Weinmann

Abstract

The Tactile Internet (TI) aims to expand seamless interaction over the Internet by providing a new form of interaction through touch by providing haptic feedback. To realize this, the TI is limited by a round-trip latency of 1-10 ms, meaning that the TI is limited by a physical distance of $1500\,\mathrm{km}$. A workaround to this requirement is the introduction of local simulations. To keep track of moving objects in these simulations, a stable tracking algorithm is needed. This algorithm is provided in the form of a particle filter. The TI requires high tracking accuracy from this algorithm, but to achieve that the algorithm becomes computationally expensive. If the movement of the to-be-tracked object is known a priori, however, the particle filter can be adapted to focus only on that movement, neglecting the other directions. This increases tracking accuracy with an equal amount of samples, thus requiring a lower amount of samples to achieve the same accuracy, reducing computational power. This paper explores how to achieve this adaptation and analyses the increase in accuracy. By adapting the filter, the tracking accuracy is significantly increased, even with a lower number of samples. This results in gaining a speedup with a factor of about 36, while having similar tracking accuracy.

1 Introduction

Online interaction is becoming more prevalent nowadays. Calling a friend, video calling a loved one overseas, or even following a lecture online are only a few examples of what is possible. However, these interactions are limited to audio and/or video only. The Tactile Internet (TI) aims to revolutionize the online interaction scene by providing interaction in the form of touch through haptic feedback. [9]. The TI makes it possible for a user to physically interact with objects remotely. The TI has various applications, for example, a surgeon could operate on a patient from their office [8], while the patient is in a hospital kilometers away. This could save precious time that the surgeon otherwise would have to spend on traveling.

The TI consists of a user controlling some device (the master side) that captures the user's movements (the controlled side). These movements are sent over a network, where a controlled device mimics the received movements. Doing this over a network poses a challenge, as the latency must be 1-10 ms [9]. If the latency is more than 1-10 ms, the user might experience effects like motion sickness, or the operations might be disrupted [15]. Even considering an ideal scenario, where the only limiting factor is the speed of light, which is about $300\,\mathrm{km/ms}$, the distance between the endpoints of a TI application can be a maximum of $150\,\mathrm{km}$ to secure the latency constraint of $1\,\mathrm{ms}$ [12], which would increase to about $1500\,\mathrm{km}$ if the latency requirement is $10\,\mathrm{ms}$, but this still is too limiting for the TI.

To circumvent the 1-10 ms delay, a solution has been found in the form of Model-mediated teleoperation (MMT).

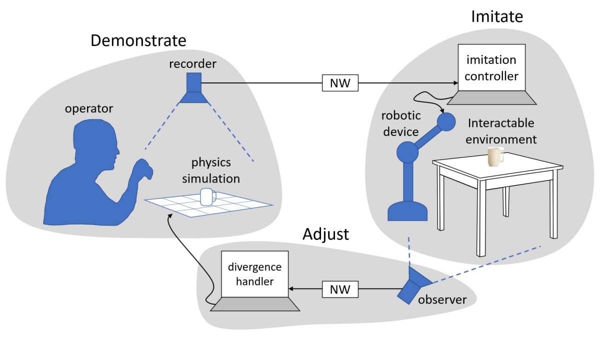


Figure 1: General depiction of an MMT system. A recorder records the actions done by an operator in a physics simulation on the master side (Demonstrate). These actions are sent over a network, where they are imitated by a robotic device on the controlled side (Imitate). The actions performed by the robotic device are observed and deviations from the master side are updated in the simulation (Adjust). Figure created by Kees Kroep.

It works by deploying simulations local simulations on the user and master side. Instead of directly interacting with each other, the user and master side now interact with these simulations. To keep the simulations from diverging, only periodic updates have to be sent over the network, eliminating the latency requirement (see Fig. 1).

Constructing a local simulation comes with several challenges. One of the challenges that arises is that in a dynamic environment, objects interact with each other and move around, making it vital to have a tracking system in place that can keep track of the movement. To solve this problem, an RGB-D camera is deployed, giving 3D representations of the environment in the form of point cloud data. From this point cloud data objects can be tracked.

The solution for tracking an object through a point cloud is found through the use of particle filtering (see Sec. 3). With particle filtering, the pose of the object is tracked in 6 Degrees of Freedom (DoF) by approximating the state of an object with randomly generated samples, also called particles [10].

With particle filtering, more samples results in higher tracking accuracy [6]. However, more particles also result in a higher computational cost and a higher execution time [14]. Therefore, it is necessary to tune the number of particles used to find a high enough accuracy with an acceptable execution time. But, if with prior knowledge, the general movement of the object is known, the filter can be tuned to only focus on the direction(s) the object will move in. This results in being able to obtain a higher tracking accuracy for the same amount of particles, therefore needing fewer particles for a similar accuracy, resulting in a massive speed-up in execution time.

This paper aims to answer the following question: "How to tune a particle filter based on prior knowledge of an object's movement?".

This paper contributes the following:

- Show how to tune the importance density of a particle filter to allow tracking for specific movement.
- Show that for a similar accuracy, the tuned particle filter gains a decrease in execution time by a factor of at least

36.

This paper is structured as follows. First, it relates previously done work to the research done in this paper in Sec. 2. This is followed by some theory explaining what a particle filter is in Sec. 3 After that, the methodology is presented in Sec. 4, where the method of research is explained and the gained insights and results are explored. Third, the experimental setup and results are in-depth explored in Sec. 5. A discussion about the results is held in the following section (Sec. 6). After the discussion, a reflection on ethical aspects is done in Sec. 7. Following the responsible research is Sec. 8, which discusses open issues and possible improvements. Finally, the paper concludes in Sec. 9

2 Related Works

2.1 Tactile Internet

The Tactile Internet comes with several challenges, but the main challenge limiting the advancements of TI is the requirement of 1-10 ms latency, or the Ultra Low Latency (ULL) requirement [15]. Another critical challenge to solve is a packet delivery reliability of $99.999\,\%$, or the Ultra Reliability (UR) requirement. [9]. To meet these requirements, ongoing research is focused on the possibilities of 5G [1]. As 5G has a broad range of capabilities, it will make for an important enabler for TI [13].

5G is currently on its own not enough to satisfy the ULL requirement [2]. Therefore, this paper uses the MMT approach to circumvent the 1-10 ms delay as mentioned in Sec. 1. The breakthroughs of 5G could also be combined with the MMT approach to further increase the responsiveness of MMT.

2.2 Object Tracking Methods

Dynamic object tracking with 3D-point clouds is an intensively explored research area, from which various options emerge. Options such as Monte Carlo approximations, like particle filters, variations of a Kalman filter [11, 16] and the use of Neural Networks [11] all appear promising.

In a linear system where the noise is Gaussian-distributed, the Kalman filter is proven to be optimal. However, in practice, the Kalman filter is limited by the non-linearity and non-Guassianity of the real world, even with its variations [3]. As the simulation created in MTT is constructed to be as close to the physical world as possible, a particle filter is a better option, as a particle filter can track objects in a non-linear setting [6].

As particle filters perform better in a physical world scenario and as MSc student Kilian van Berlo has done extensive research on particle filtering and created a base tracking system (see [14]), this paper builds upon particle filtering. The possibilities of object tracking using Neural Networks are not explored in this paper.

3 Theoretical Background

This section gives a definition of particle filters and the Kullback-Leibler Distance (KLD) particle filter variation. Readers already familiar with these definitions can skip this section.

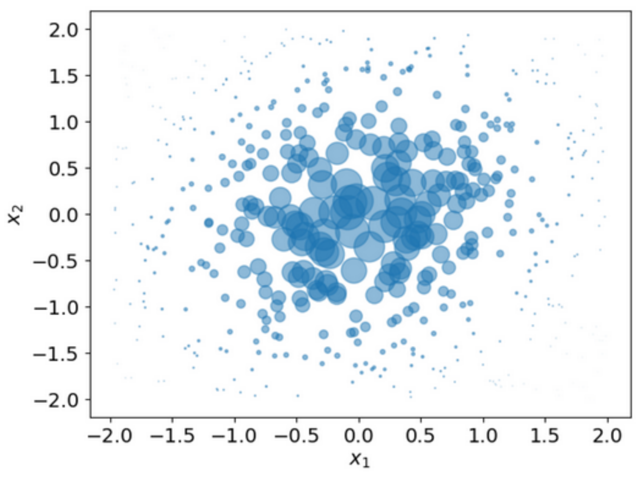


Figure 2: Set of 500 weighted samples (blue circles) estimating a 2D-zero mean Gaussian distribution. The radius of the samples is proportional to their weights. Samples are drawn from a 2D uniform distribution, where the weights compensate for the difference in the importance density and posterior. With the proper weights, the weighted samples provide a reasonable estimate [6]. Figure created by Jos Elfring et al.

3.1 Particle Filter

Particle filtering is a sequential Monte Carlo methodology, where the idea is to approximate the probability distribution of a state with discrete random measures, called particles. The posterior probability distribution is approximated by these particles and a weight assigned to them [5]. The posterior is modeled by a Markov Chain, stating that the posterior only depends on the previous time step and is independent of all other steps [6]. Given the posterior probability distribution $p(x)$, the approximation of $p(x)$ is given by:

$$X = \{w_t^i, x_{0:t}^i\}_{i=1}^N,$$

where X is a set containing N samples (particles) and weights. Each sample $x_{0:t}^i$ is a representation of the possible state sequence. A weight w_t^i is the relative importance of all N samples $x_{0:t}^i$ at time step t, and $\sum_{i=1}^N w_t^i = 1$. Samples that are given a high weight are a closer representation of the real sequence than samples that are given a lower weight [6].

This approach comes with a challenge, as the posterior is unknown, hence sampling from it is not possible. Therefore, samples must be taken from another distribution instead, called the importance density. The weights compensate for the fact that the samples are drawn from this importance density instead of the posterior distribution (see Fig. 2) [6].

The next step in the particle filter is determining the weight of the samples given the importance density. This is done by obtaining a posterior guess for each particle following the Markov Chain, so the guess is based only on the prior. Then, after receiving measurements, the precision of the guess is computed using a likelihood function. From this likelihood function, a weight is obtained and the current weight of a particle is updated to be the new weight. This is called Sequence Importance Sampling (SIS).

If the particle filter keeps executing with only updating the weights, the variance of the weights will increase, and a few particles will hold most of the weight, while the other particles will hold a negligible weight. This problem is called degeneracy [10]. To solve this problem, another step is introduced in a particle filter, called resampling. Resampling aims to prevent degeneracy by taking samples from the original distribution X_t to create a new distribution $X_t^{'}$ and replacing X_t with $X_t^{'}$. The sampling is done by taking samples from X_t based on the weights of the particles. This means that particles with a high weight have a high probability to be sampled, while particles with a low weight have a low probability to be sampled. Since the particles are not removed after they are sampled once, $X_t^{'}$ will most likely hold particles with high weights, meaning that particles with negligible weight are very likely to not be included in $X_t^{'}$ [10].

3.2 KLD-Sampling

The algorithm used for this paper is a variation of the particle filter algorithm using Kullback-Leibler Distance (KLD) sampling. This variation adapts the number of samples used over time, which increases the efficiency of particle filters. This is done by determining the number of samples so that the distance between the Maximum Likelihood Estimate (MLE) and the true posterior does not exceed a threshold ϵ. The distance between the posterior and the MLE is determined by the KLD, hence the name of the algorithm [7].

4 Methodology

This section explores the methodology of the paper. It explains what has been done to answer the research question and the rationale behind the choices made. It first presents the insight gained into what a particle filter is, followed by the possibilities of GPU acceleration for particle filtering. After that, the transition in the research direction from GPU acceleration to the current research question is explained. Following this is an overview of the contributions given to the development of the tracking test bed used for the experiments. Lastly, the experiments done are laid out.

4.1 Particle Filter

The first step in the research cycle was to consult the literature to find out what a particle filter exactly is. The first paper read was an MSc thesis giving background about TI and the use of a particle filter algorithm, as well as explaining the results of creating a first tracking test bed (see [14]). After that other papers were consulted to find a more mathematical and in-depth explanation of particle filters (see [3, 5–7, 10]). A definition of particle filters can be found in Sec. 3 for readers unfamiliar with this concept.

4.2 GPU Acceleration

Originally, the aim of this project was to find a way to speed up the particle filter algorithm by GPU acceleration. As particle filters have high computational costs, adopting them for robotic uses for example has been limited. However, particle filter algorithms are inherently parallelizable, as their main bottleneck is the likelihood evaluation. If N particles are used, N identical and independent likelihood evaluations are performed at each iteration. Since these evaluations are independent of each other, they can be parallelized. Since a Graphics Processing Unit (GPU) provides massive parallel power, it is ideal for a particle filter to enable fast object tracking [4].

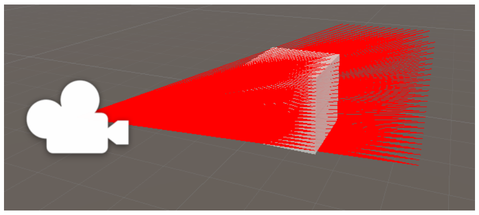


Figure 3: The Unity scene used to create point clouds of an object. Rays are shot from the camera towards the far-clapping plan of the camera. The rays that hit an object are colored red for debugging purposes.

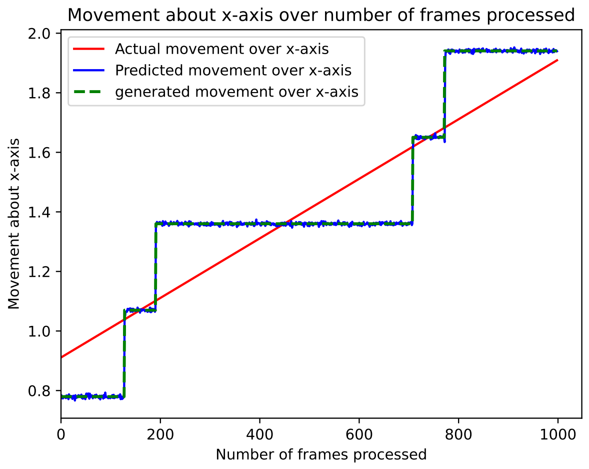


Figure 4: Graph of a cube translating along the x-axis. The red line is the movement of the cube as generated by unity, the actual movement. The dashed green line is the movement as generated by the point cloud, the mismatched movement. The blue line is the movement predicted by the particle filter algorithm. The movement is in meters.

To show the speed up a GPU-accelerated particle filter can provide, the average execution time per frame of a particle filter algorithm without GPU acceleration was measured with a varying number of particles. However, before the implementation of GPU acceleration was attempted and experiments were done with the upgraded algorithm, it was decided to switch from GPU acceleration to the current research topic, as this was expected to yield more interesting results and be more useful for the Embedded System and Network System group this research was conducted with.

4.3 Improving the Test Bed

Group mate Yue Chen created a particle filter tracking test bed as part of their research. In order to perform experiments for this paper, the tracking test bed had to be fully functioning first. Therefore, help with debugging/improving the code was given to proceed with further testing and experiments.

Movement Mismatch In a real-world TI scene, an RGB depth camera will record the scene, from which a point cloud is created. To mimic this depth camera, Yue Chen created a virtual depth camera in Unity, which can record objects and create a point cloud out of partial object view. This is done by shooting a ray from the near-clipping plane of the camera to the far-clipping plane of the camera for each height and width pixel of resolution of the camera. This resolution can be manually set. If a ray hits an object, the hit-point is calculated and saved as a point in the point cloud, as can be seen in Fig. 3. From the Unity scene, the objects can be made to move in any desired direction. This movement is then recorded by the camera and can be run for a set amount of frames, where a new point cloud of the object is created for each frame. These point clouds can then be fed into the tracking algorithm to track the wanted object.

However, the movement generation and point cloud generation were mismatched, as the point cloud generation was controlled from the camera object, but the movement was controlled from the desired object that is to be tracked, causing the mismatch. This resulted in the particle filter algorithm tracking the object based on the mismatched movement, causing tracking error, as shown in Fig. 4. Even though the particle filter closely matches the mismatched movement, it is far from the actual movement. To fix this, the movement of the object is controlled by the camera, instead of directly from the object itself. This means that the object will always move after a point cloud of a frame has been created, thus causing the movement of the point cloud to be exactly the movement of the object.

Other improvements made to the test bed had to do with being able to obtain results. This includes being able to measure execution time, writing movement to a file directly from the Unity scene, instead of only using the movement from the generated point cloud, and being able to plot various results. Help was also provided with doing various tests to tune the particle filter algorithm.

4.4 Performing Experiments

The final step was to perform the experiments to measure the accuracy of a standard particle filter algorithm and a particle filter algorithm tuned to be optimized for specific movement based on prior knowledge.

To test this, three situations were defined. The first situation is a cube translating over only the x-axis following a sinusoidal movement while keeping the y-axis, the z-axis, and the rotation constant (1 DoF). The second situation is translating the same cube again over the x-axis following a sinusoidal movement, but also translating the cube over the z-axis following a cosine with a slightly different frequency, causing the cube to move in a spiral. The rotation and y-axis are kept constant (2 DoF). The last situation is the same as the second one, but letting the cube rotate about the y-axis. Translation over the y-axis and the other two rotational axes are kept constant (3 DoF) (see Fig. 5). These three situations were chosen as similar situations will be performed during a real-life demo presenting the possibilities of the TI.

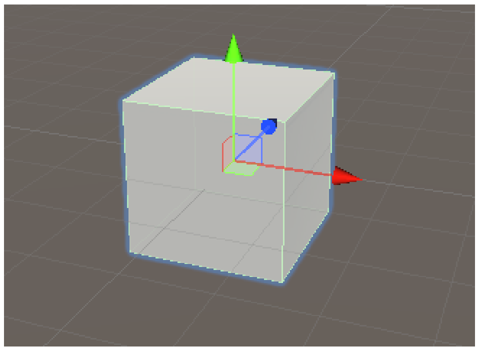


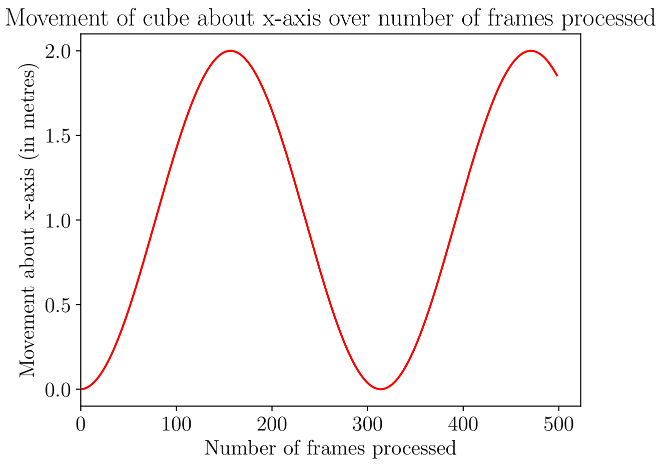
Figure 5: The axis system in Unity. The red arrow is the x-axis (rotation about this axis is called roll), the green arrow is the y-axis (rotation about this axis is called pitch) and the blue arrow is the z-axis (rotation about this axis is called yaw).

To tune the particle filter, the covariance matrix can be changed based on the experiment. The importance density used for the experiments is a Gaussian distribution with its mean at the center of the cube, which was centered at the origin for all the experiments. The covariance σ of the distribution, which can be manually set in the algorithm, indicates the spread of the area in which the particles are sampled. The covariance matrix is then given by:

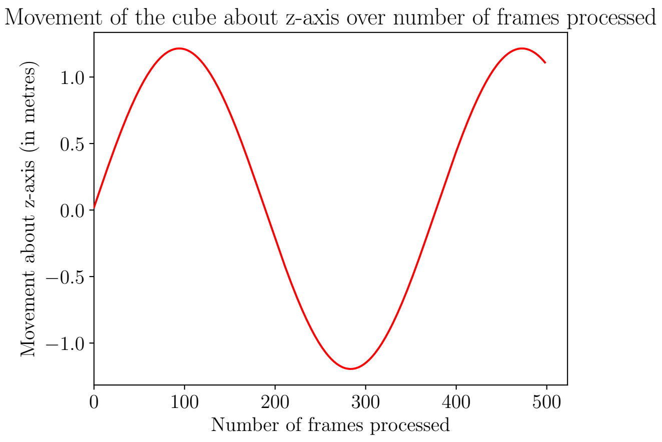
$$\begin{bmatrix} \sigma_x & & & & & 0 \\ & \sigma_y & & & & \\ & & \sigma_z & & & \\ & & & \sigma_{ro} & & \\ & & & & \sigma_{pi} & \\ 0 & & & & & \sigma_{ya} \end{bmatrix},$$

where the covariance on the diagonal is for the x-axis, y-axis, z-axis, roll, pitch and yaw respectively. For each diagonal, a factor can be set to multiply the covariance to indicate the spread needed on the specific axis. If it is known that an object is not going to move or rotate about certain axes, the factor can be set to 0. This results in the same amount of particles being sampled in a smaller area, thus increasing the accuracy for the axes that the object is known to move or rotate about while neglecting the axis that the object isn't going to move or rotate about. This effectively means that the particle filter can be changed from a tracking algorithm that tracks in 6 DoF to a tracking algorithm that only tracks in the relevant DoF. This method was hypothesized based on the definition of a particle filter explored by reading literature.

After the method of tuning was found, the experiments were each run once to indicate if the hypothesized method was giving results. After the first two experiments, the results for translation only looked promising, when doing the third experiment however, it was noted that the rotational data was not saved anywhere, meaning that the accuracy of rotation could not be evaluated. After fixing this, the rotational accuracy of the algorithm while tracking in 6 DoF turned out to be very poor. Due to time constraints, this issue could not be fixed, therefore the experiments will only include experiments where the cube is not rotated.



(a) Movement of the cube along the x-axis



(b) Movement of the cube along the z-axis

Figure 6: Movement of the cube along the x and z-axis

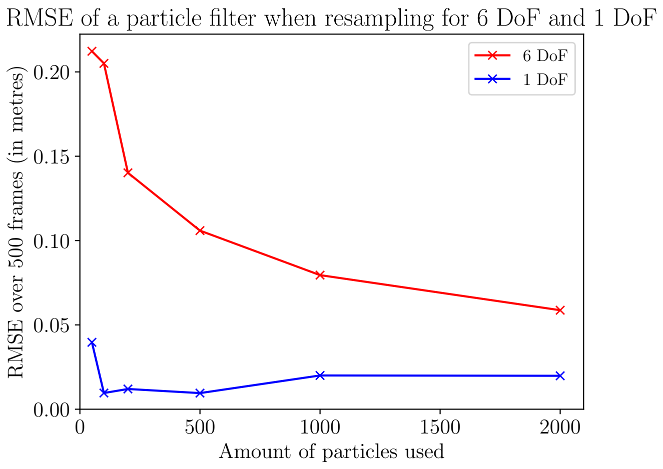


Figure 7: Root Mean Square Error (RMSE) when tuning the particle filter algorithm for 6 DoF and 1 DoF for a various amount of particles

Amount of Particles	Average execution time per frame for 6 DoF (ms)	Average execution time per frame for 1 DoF (ms)
50	182.94	134.23
100	351.08	258.64
200	695.46	352.03
500	1705.79	352.87
1000	3199.71	388.32
2000	4885.41	413.99

Table 1: Average execution time per frame of translating a cube over the x-axis for 6 DoF and 1 DoF for a different amount of particles

5 Results

5.1 Experimental Setup

For all experiments, a cube of $1 \times 1 \times 1$ m is used as the object that is tracked by the particle filter algorithm. All variables of the particle filter are kept constant throughout all experiments, except for the covariance matrix, which will be either tuned for 6 DoF, 1 DoF, or 2 DoF. The number of particles used will also be changed to get an indication of the accuracy improvement with different amounts of samples. The number of particles used is 50, 100, 200, 500, 1000, and 2000 to show the trade-off between accuracy and execution time of a particle filter algorithm. The algorithm is for each amount of particles run for 500 frames, where the algorithm has to finish tracking one frame before moving to the next.

As mentioned in Sec. 4, the cube will translate only along the x-axis in the first experiment, which can be seen in Fig. 6a. During the second test, the cube will translate along the x-axis in the same manner as during experiment one, but now the cube will also translate along the z-axis, which can be seen in Fig. 6b. The difference in frequency between the x and z movement causes the cube to move in a spiral, instead of a circle.

5.2 Performance Analysis

In this section, the results are presented. First, the results of tuning the particle filter algorithm to 1 DoF are presented, then the results of tuning a particle filter to 2 DoF are presented.

1 DoF For each amount of particles, the algorithm was run in 6 DoF and 1 DoF, and the Root Mean Square Error (RMSE) was calculated. The results can be seen in Fig. 7. The average execution time per frame can be seen in Table. 1.

From Fig. 7 it can be seen that the RMSE is lower for tracking in 1 DoF than it is for tracking in 6 DoF for all particle amounts. From Table. 1, it can be seen that the average execution time per frame is also lower for tracking in 1 DoF than tracking in 6 DoF for all particle amounts.

Inference 1: Tracking in 1 DoF gains over tracking in 6 DoF for each amount of particles a significant increase in accuracy while simultaneously gaining a significant decrease in execution time.

From Fig. 7 it can be seen that the RMSE when tracking in 6 DoF with 2000 particles comes closest to tracking in 1 DoF with 50 particles, where the RMSE of tracking in 1 DoF

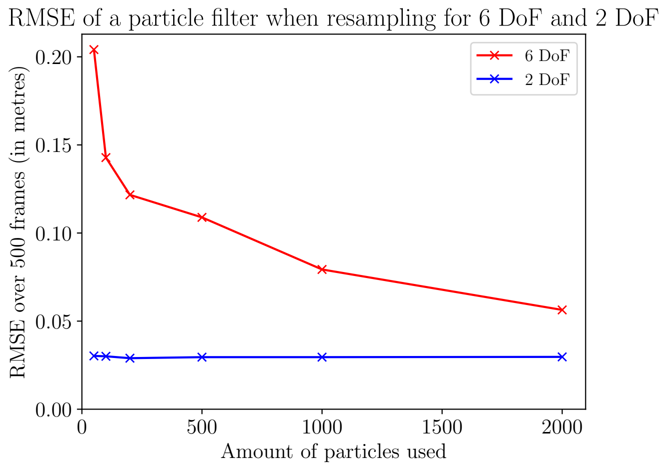


Figure 8: Root Mean Square Error (RMSE) when tuning the particle filter algorithm for 6 DoF and 2 DoF for a various amount of particles

Amount of Particles	Average execution time per frame for 6 DoF (ms)	Average execution time per frame for 2 DoF (ms)
50	132.58	133.93
100	272.50	259.46
200	487.33	503.87
500	1165.57	1191.07
1000	2613.64	1963.42
2000	4963.51	1820.59

Table 2: Average execution time per frame of translating a cube over the x-axis for 6 DoF and 2 DoF for a different amount of particles

with 50 particles is lower by a factor of approximately 1.5. However, tracking in 1 DoF is about 36.4 times faster than tracking in 6 DoF with 2000 particles.

Inference 2: Tracking in 1 DoF with a lower amount of samples has a similar or higher accuracy than tracking with a high amount of samples in 6 DoF while gaining a significant decrease in execution time.

2 DoF For each amount of particles, the algorithm was run in 6 DoF and 2 DoF, and the RMSE was calculated. The results of be seen in Fig. 8. The average execution time per frame can be seen in Table. 2.

From Fig.8 it can be seen that for all particle amounts, the RMSE is lower for tracking in 2 DoF than tracking in 6 DoF. From Table. 2, it can be seen that the average execution time per frame is only significantly lower in 1 DoF for 1000 and 2000 particles, but similar for the other amounts.

Inference 3: Tracking in 2 DoF gains over tracking in 6 DoF for each amount of particles a significant increase in accuracy while not compromising on execution time.

From Fig. 7 it can be seen that the RMSE when tracking in 6 DoF with 2000 particles comes closest to tracking in 2 DoF with 50 particles. Tracking in 2 DoF has a lower RMSE by a factor of roughly 1.9. However, tracking in 2 DoF is faster

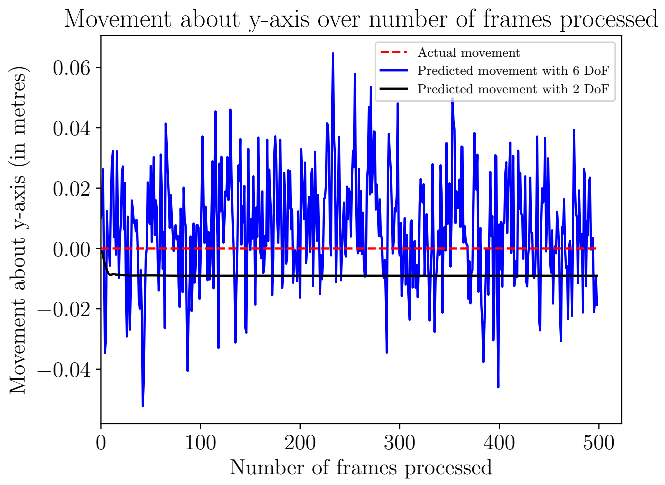


Figure 9: Example of the particle filter algorithm predicting the movement about the y-axis with an offset. As can be seen, tracking in both 2 DoF (the black line) and 6 DoF(the blue line) predict the true movement about the y-axis (dashed red line) with an offset.

with a factor of approximately 37 than tracking in 2 DoF with 2000 particles.

Inference 4: Tracking in 2 DoF with a lower amount of samples has a similar or higher accuracy than tracking with a high amount of samples in 6 DoF while gaining a significant decrease in execution time.

Limitations The results presented have some limitations. The first and major is that the experiments have only been performed once. This means that the results are not an average taken from multiple runs, thus the results are likely to have outliers. This would very likely explain why in Fig. 7 the tracking is most accurate for 100 particles, instead of for a higher particle amount.

Another limitation is that the particle filter randomly predicts the movement about an axis with an offset on that axis (see Fig. 9). This then causes the error to be higher than expected. This seems to happen for each axis, and the cause of this is currently unknown.

6 Discussion

This section outlines the main takeaways resulting from this research project. It explains the things that were learned and the things that would be done differently.

As mentioned in Sec. 4, the first step taken was reading literature to understand what a particle filter is and how it works mathematically. In order to understand the principle of operation of particle filters and to be sure to be able to correctly interpret results, significant time has been spent on reading literature and figuring out the math behind it. It might have been more efficient to spend less time reading and more on studying the actual code, for a better intuition of how the code functions and to start earlier with aiding Yue Chen in setting up the test bed.

As also mentioned in Sec. 4, the research question was changed during the research phase. As this can be a natural event during research, this was an opportunity to learn how to handle changes in the research topic during research and how to continue with the new topic. This went smoothly and thus switching research question did not cause a significant loss of time.

7 Responsible Research

For this paper, the experiments were conducted without human participants nor was any personal data used. Therefore, all responsibility related to the results and integrity of the paper falls on the author. To preserve the integrity of the paper, all sources used are listed and the results weren't altered.

However, the experiments were performed with a test bed created and largely set up by group mate Yue Chen. Even though help was given with debugging the test bed, all credit for creating this and setting this up is given to Yue Chen by the author.

A critical aspect to maintain scientific integrity and uphold a researcher's ethos is conducting experiments and obtaining reproducible results. Therefore, the methodology, experiments, and results are carefully explained, and all parameters used are given. The code for tracking and point cloud creation will be made available as well. It is important to note that a particle filter is a tracking method involving probability. Therefore, when recreating the experiments, slight deviations in results are to be expected.

8 Future Work

With further development, errors could be fixed in the tracking test bed for more representative results. The first major error that currently exists is that the generated point cloud randomly has offset to the origin of the axes. When this happens, the particle filter performs worse on the specific axis than expected, thus increasing the tracking error. The cause of the error is unknown, but fixing this error would stabilize the performance of the algorithm. The second major error is that the tracking algorithm performs poorly when an object is rotated. This likely is because the covariance for the roll, pitch, and yaw aren't properly tuned. Fixing this would mean that experiments could be performed to indicate how well a tuned particle filter handles rotations.

Another future improvement is performing the presented experiments multiple times to obtain an average tracking error. This would be more representative than the current results from only one iteration of experiments, as the particle filter predicts the next state of an object with a certain probability. This means that the results will be slightly different each time, and results can have outliers. This uncertainty in results would be eliminated by taking an average of a certain amount of experiment iterations.

Finally, the execution time of the algorithm could be further reduced by deploying an upgrade using GPU acceleration, or the tracking accuracy could be improved using Neural Networks.

9 Conclusions

The Tactile Internet aims to revolutionize online interaction by enabling interaction through touch. It is however limited by the Ultra Low Latency requirement. To circumvent this requirement, a solution has been proposed in the form of interacting with simulations. One challenge that comes with these simulations is the need for object tracking. To track objects, a particle filter algorithm is used, however, this algorithm is computationally expensive.

This paper aimed to find a method to tune a particle filter based on prior knowledge of an object's movement to obtain a similar accuracy while gaining a massive decrease in execution time. To achieve this, the tracking error of a standard particle filter was measured with different amounts of samples, and compared to the tracking error of a particle filter tuned for specific object movement. To tune the particle filter the covariance matrix of the importance density Gaussian distribution was changed to focus only on the direction of movement. This leads to that while having a similar accuracy for the original and tuned particle filter, the tuned filter has a decrease in accuracy by a factor of approximately $36,4$ for 1 DoF and 37 for 2 DoF. Therefore, it is recommended to tune a particle filter if the movement of an object is a priori known, to gain a massive decrease in execution time.

References

[1] Konstantinos Antonakoglou, Xiao Xu, Eckehard Steinbach, Toktam Mahmoodi, and Mischa Dohler. Toward haptic communications over the 5g tactile internet. *IEEE Communications Surveys & Tutorials*, 20(4):3034–3059, 2018.

[2] Abdelhamied A Ateya, Ammar Muthanna, Anastasia Vybornova, Irina Gudkova, Yuliya Gaidamaka, Abdelrahman Abuarqoub, Abeer D Algarni, and Andrey Koucheryavy. Model mediation to overcome light limitations—toward a secure tactile internet system. *Journal of Sensor and Actuator Networks*, 8(1):6, 2019.

[3] Zhe Chen. Bayesian filtering: From kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

[4] Changhyun Choi and Henrik I Christensen. Rgb-d object tracking: A particle filter approach on gpu. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1084–1091. IEEE, 2013.

[5] Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. Particle filtering. *IEEE signal processing magazine*, 20(5):19–38, 2003.

[6] Jos Elfring, Elena Torta, and René van de Molengraft. Particle filters: A hands-on tutorial. *Sensors*, 21(2):438, 2021.

[7] Dieter Fox. Kld-sampling: Adaptive particle filters. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[8] Vineet Gokhale, Mohamad Eid, Kees Kroep, R Venkatesha Prasad, and Vijay S Rao. Toward enabling high-five over wifi: A tactile internet paradigm. *IEEE Communications Magazine*, 59(12):90–96, 2021.

[9] Kees Kroep, Vineet Gokhale, Ashutosh Simha, R Venkatesha Prasad, and Vijay S Rao. Tim: A novel quality of service metric for tactile internet. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, pages 199–208, 2023.

[10] Tiancheng Li, Miodrag Bolic, and Petar M Djuric. Resampling methods for particle filtering: classification, implementation, and strategies. *IEEE Signal processing magazine*, 32(3):70–86, 2015.

[11] Peter Morton, Bertrand Douillard, and James Underwood. An evaluation of dynamic object tracking with 3d lidar. In *Proc. of the Australasian Conference on Robotics & Automation (ACRA)*, page 38, 2011.

[12] Nattakorn Promwongsa, Amin Ebrahimzadeh, Diala Naboulsi, Somayeh Kianpisheh, Fatna Belqasmi, Roch Glitho, Noel Crespi, and Omar Alfandi. A comprehensive survey of the tactile internet: State-of-the-art and research directions. *IEEE Communications Surveys & Tutorials*, 23(1):472–523, 2021.

[13] Meryem Simsek, Adnan Aijaz, Mischa Dohler, Joachim Sachs, and Gerhard Fettweis. 5g-enabled tactile internet. *IEEE Journal on selected areas in communications*, 34(3):460–473, 2016.

[14] Kilian van Berlo. Capturing real-time dynamic environments for tactile internet, Nov 2022.

[15] Daniël Van Den Berg, Rebecca Glans, Dorian De Koning, Fernando A. Kuipers, Jochem Lugtenburg, Kurian Polachan, Prabhakar T. Venkata, Chandramani Singh, Belma Turkovic, and Bryan Van Wijk. Challenges in haptic communications over the tactile internet. *IEEE Access*, 5:23502–23518, 2017.

[16] Jonathan Vincent, Mathieu Labbé, Jean-Samuel Lauzon, François Grondin, Pier-Marc Comtois-Rivet, and François Michaud. Dynamic object tracking and masking for visual slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4974–4979. IEEE, 2020.