Tracking Physics: A Virtual Platform for 3D Object Tracking in Tactile Internet Applications

Yue Chen[1]

Supervisor(s): Ranga Rao Venkatesha Prasad[1]**, Kees Kroep**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

Name of the student: Yue Chen
Final project course: CSE3000 Research Project
Thesis committee: Ranga Rao Venkatesha Prasad, Kees Kroep, Michael Weinmann

Abstract

Tactile Internet (TI) is a pioneering network paradigm that aims to communicate haptic feedback with ultra-low latency. It will enable new ways for us to interact with remote environments, such as transferring skills over the network and controlling remote objects. One crucial component of this framework is the ability to track the position and movement of remote objects. While there are many tracking algorithms, evaluating them in TI applications can be time-consuming, expensive, and sometimes impractical. Hence, we present a virtual platform for developing, evaluating, and comparing tracking solutions. We demonstrate how our platform can be used to tune tracking algorithms and determine trade-offs between different types of hardware. In turn, this provides insights that were previously difficult to obtain using real physical platforms, as real world coordinates may not align with virtual coordinates and experiments are not easily repeatable.

1 Introduction

Modern communication systems for remote interaction between human users or entities primarily rely on visual and auditory feedback. However, the full realization of haptic feedback, enabling direct manipulation and interaction with dynamic objects in remote environments, remains an ongoing challenge [5]. Haptic feedback would allow for *teleoperation*, where human operators can exert real-time control over remote objects. This would require ultra-low latency (ULL) communication with an end-to-end delay of approximately 1 ms [5]. Tactile Internet (TI) is a network paradigm aimed at realizing these requirements [14], ensuring ULL transmission capabilities and high availability, thereby unlocking many more ways to interact with remote environments.

Potential use cases include robot-assisted surgery, where skilled surgeons can guide robots to perform procedures on individuals in remote regions, or technicians conducting clean-ups and repairs in hazardous environments such as nuclear plants or satellites [5; 18]. TI systems also hold promise for education and training, where exoskeletons are used for physical movement training or rehabilitation. All these applications require a round trip delay of approximately 1 ms [5]. However, achieving round trip delays within 1 ms is simply impractical. Given the physical distance that may exist between the master and controlled domain, the delay would exceed 1 ms even if data signals traveled at the speed of light [23].

To address this challenge, model-mediated teleoperation (MMT) was introduced as a solution [15]. In MMT, a local model of the remote environment resides in the master domain. The operator will receive instant haptic feedback from the local model instead of the controlled domain which effectively satisfies the 1 ms delay [21] requirement.

Instead of employing MMT and a predefined local model directly, we build upon this framework by using a physics en-

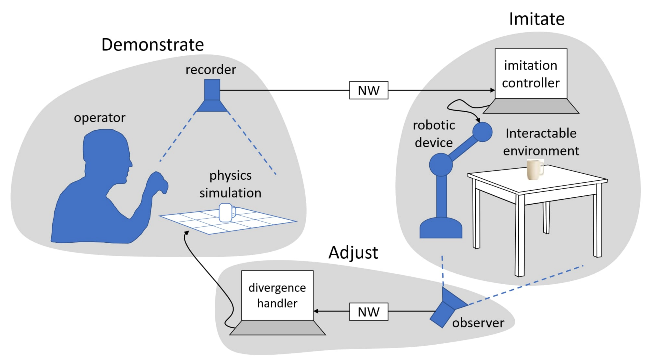


Figure 1: The master domain demonstrates the actions to be performed, which are then transmitted over to the controlled domain. The physics engine models the controlled domains as accurately as possible, with the aid of the convergence handler.

gine to simulate the controlled domain. In this way, the representation of the controlled domain is dynamic and flexible. We envision that tracking objects in the controlled domain will be one of the critical tasks of this new method. This not only calls for a tracking solution now but also a platform that accommodates testing and developing future solutions. As shown by Van Berlo [2], verifying the performance of real-life tracking solutions is a challenge. In addition to requiring a physical setup with a depth camera, there may not be a well-defined ground truth for verification. Repeatability is also not easily achievable with real-life tracking experiments, making it difficult to compare and evaluate solutions

In this work, we set the stage for developing tracking solutions that will drive the Tactile Internet applications of the future. Our proposed platform will allow one to develop, compare and evaluate tracking solutions, as well as accommodate both real and virtual sensor devices. Furthermore, using our platform, we take the first step to draw inferences on how particle filters can be tuned for tracking and how our solution can be used to choose appropriate sensor hardware. As of writing, two additional bachelor projects (by Koen Snijder and Sven Pegels) and two master students (Rodrigo Álvarez Lucendo and Pavlos Makridis) already used this platform for research and experimentation.

Contributions Our contributions are enumerated below:

1. We provide a virtual depth camera in Unity, capturing arbitrary dynamic scenes and producing corresponding point clouds. As it is possible to encode the movement of objects, the ground truth is always known.
2. We offer a tracking program that can track point clouds using the particle filter and produce predictive feedback.
3. Our virtual system allows for the modeling of arbitrary sensor devices, with varying precision or quality. This ensures that it is easy to evaluate how different devices may affect the performance of the MMT system.
4. We implemented a generic communication interface for the tracking program, allowing real physical sensor devices to easily interface with it.

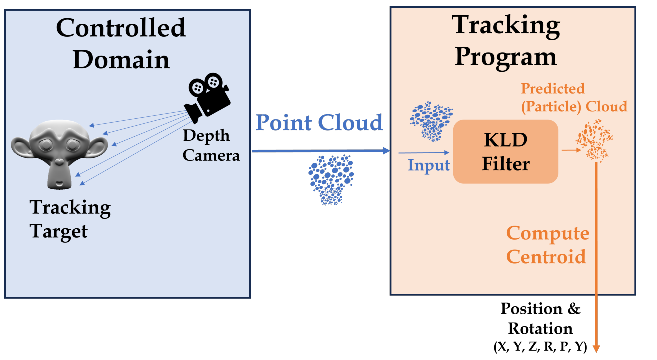


Figure 2: A schematic diagram of the communication framework which features two UDP communication channels. The first channel is used by Unity's virtual camera to transmit point cloud data. The PCL tracking program offers predicted positions (X, Y, Z, R, P, Y) used in the second communication channel.

5. We also conduct an extensive investigation on how parameters in the particle filter can affect tracking accuracy and identify relevant trade-offs that may exist.

The rest of the paper is organized as follows. Section 2 analyzes similar works in the field of TI and MMT. Section 3 explains the implementation of each component in the virtual testbed. Section 4 describes the experiment set-up and is followed by an evaluation of relevant results. The ethical implications and reproducibility of our work are discussed in Section 5. Finally, Section 6 suggests possible extensions to our work and concludes the paper.

2 Related Work

In this section, we analyze some existing work in the field of point cloud tracking and TI applications. Section 2.1 provides an overview of common point cloud tracking solutions, and Section 2.2 describes recent works in the field of TI applications and corresponding testbeds.

2.1 Point Cloud Tracking

Point cloud tracking is a common challenge tackled in the field of robotics, computer vision, and autonomous driving [1; 3; 13]. Current approaches can be categorized either as adapting traditional control theory algorithms or applying state-of-the-art machine learning techniques.

Works that employ traditional tracking algorithms generally exploit additional information sources available in the target environment. Hossein *et al.* [9] introduces an inertial measurement unit (IMU) and global positioning system (GPS) to optimize point cloud tracking. The authors show that by incorporating these additional sources of information into the Kalman filter, tracking accuracy is improved albeit with lower quality or occluded point clouds. Similarly, Held *et al.* [8] proposes a technique to combine RGB information and object velocities to improve the predictions of the Kalman filter. Their method improves the efficiency of real-time state exploration during tracking.

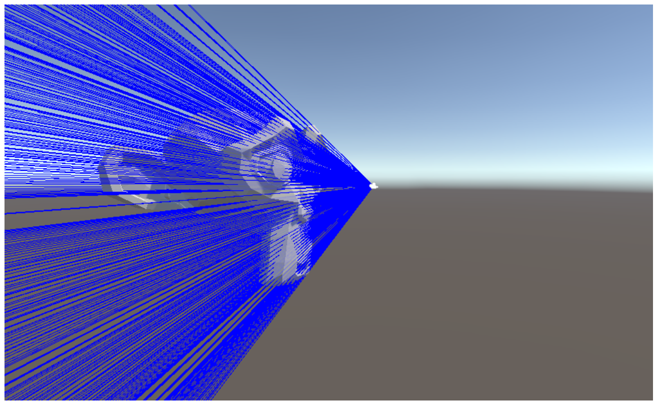


Figure 3: The rays cast by the virtual depth camera are visualized. Points of intersection with the actual mesh are used in the final point cloud.

Deep learning is also a prevalent solution for point cloud tracking. In Zhou *et al.* [25], a transformer-based network is used to predict the location and orientation of the target object. Their work shows improved tracking accuracy for sparse point clouds, as a result of subsampling the input cloud so that critical features of the target environment are well preserved. Qi *et al.* [19] presents a neural network that jointly executes 3D target proposal and verification during tracking, which in turn optimizes target search. Furthermore, Vaquero *et al.* [24] applies a deconvolutional network to point clouds to map learned features into 2D, which are then used as input to a Kalman filter. This approach reduces the data requirement for training the network that performs pose estimation and tracking.

2.2 Tactile Internet Applications and Testbeds

Recent works in MMT and TI applications typically focus on improving specific components within applications, while testbeds often assess only specific aspects of the overall system. There is a limited number of works that address generic testbeds or applications.

For example, Song *et al.* [21] proposed a parameter update method to mitigate abrupt or unstable force feedback from the local model, allowing for more stable interactions. In Liu *et al.* [12], the parameters of the local model are estimated using the recursive least squares technique, which aims to improve system transparency between the master and controlled domain. Alternative architectures for MMT have also been proposed to target system instability and enhance transparency by Leonam *et al.* [16]. However, their experiments were limited to 1 Degree of Freedom (DoF) interactions. Engelhardt *et al.* [4] presented a novel communication protocol aimed at haptic communications and a corresponding testbed to evaluate the protocol using small wireless devices over long distances. Examples of more generic testbeds for TI applications can be found in [17] and [7]. Polachan *et al.* [17] presents a testbed for TI-based cyber-physical systems that facilitates experimentation with alternative communication protocols and hardware devices. Gokhale *et al.* [7] offers a generic categorization of TI applications and implementation details for a versatile testbed.

3 Methodology

This section outlines how each component of the virtual testbed was designed and its corresponding functionality.

3.1 Virtual Depth Camera

Experimental set-up for model-mediated teleoperation or TI applications generally requires a Kinect sensor, which gathers depth or motion information about objects in an environment, and an external program to track the dynamic objects. Such experiments can be costly to set up, as hardware devices may not be readily available, and physical test beds can be time-consuming to build. Moreover, investigating the effects of higher quality Kinects on the overall system performance is expensive, as it necessitates purchasing additional hardware. To address these issues, we have implemented a virtual depth camera that mirrors the capabilities of a real Kinect, along with multiple configurations that can be modified with ease.

The implementation of our virtual depth camera uses Unity, more specifically using the Physics Raycasting library. Given a virtual scene, a ray is cast through each pixel center of the image plane. For each collision of an object with a ray, the corresponding coordinates of the intersection and the RGB information of the object are stored in PCD format. Fig. 3 shows how the rays are cast to collect depth information. Note that only the rays that collide with an object are drawn. The number of rays used for collecting the final point cloud can easily be adjusted. For dynamic scenes, the frame rate can also be adjusted accordingly to model different devices.

3.2 Point Cloud Tracking

In this section, we present the particle filter algorithm and how it was used to develop our point cloud tracking program. We used a variation of the particle filter implemented in the Point Cloud Library [20]. A comparison of the filters available in PCL [20] can be found in Appendix A.

Background

Particle filters are a family of sequential Monte Carlo methods which are employed in statistical estimation and inference problems. It offers a flexible approach for estimating the state of a dynamic system given a set of potentially noisy or incomplete observations. Particle filtering leverages a collection of weighted particles to represent the potential states of the system. These particles undergo iterative updates through a two-step process comprising prediction and update. In the prediction step, particles are propagated forward in time using a dynamic model. In the update step, particles are reweighted based on the likelihood of the observations and resampled to concentrate on regions of heightened probability [10]. By employing particles to represent the state space, particle filtering can handle nonlinear and non-Gaussian scenarios [10]. This was used as opposed to the Kalman filter, which is based on unimodal Gaussian densities and therefore cannot be employed to predict the movement of a point cloud. Moreover, the particle filter offers the ability to model the true point cloud using the particle cloud, whereas the Kalman filter only provides a single estimation of the position and orientation of the tracking target.

The KLD-adaptive filter builds upon the original particle filter by varying the number of particles used per iteration. This is achieved by bounding the approximation error ϵ, calculated using the Kullback-Leibler distance, with a certain probability δ. The approximation error ϵ is defined as the distance between the maximum likelihood estimate and the true posterior [6]. The motivation for this approach is that if the approximations are spread over a small part of the state space, fewer particles are used. However, if there is more uncertainty and particles are spread over a larger area of the state space, more particles are introduced.

As we aimed to investigate how the KLD adaptive filter can be tuned to track different types of motion, we also performed experiments (see Section 4.3) which varied the following parameters:

- **Sampling Covariance**: The sampling covariance refers to the variance of the Gaussian distributions used to model the true posterior distribution. Each degree of freedom has an associated Gaussian, hence the sampling covariance is represented by a 6-dimensional vector. Variance indicates how much a particle or point may change between two consecutive frames in each of the 6 degrees of freedom. A large variance means that the particle is believed to move more in a specific direction, and vice-versa for a smaller variance.
- **Downsampling Level**: Downsampling refers to taking a subset of representative points of the actual point cloud. A smaller point cloud can increase the efficiency of the particle filter and maintain accuracy if representative points are preserved. However, a high level of downsampling can also result in a loss of accuracy as the structure of the object may be lost.
- **Number of Particles**: The number of particles is the number of samples used for state estimation. The greater the number of samples, the more accurately the true posterior distribution can be modeled, but at the loss of computational efficiency. Since the number of particles used per iteration are not deterministic, we verify the impact of the particle count by manually setting a particle count when initializing the particle filter.

Implementation

During implementation, there were several obstacles in applying the PCL particle filter to the generated point clouds. Due to the lack of documentation, debugging was very time-consuming. Previous examples and projects used a real Kinect device to provide input to the particle filter, while our implementation provided a sequence of virtually generated point clouds. This posed challenges in referencing previous solutions as we could not easily derive appropriate filter parameters from them. Thus, alternative libraries to PCL were briefly considered. However, after comparing available functionalities in other libraries, it was decided that PCL remains the most suitable and that further troubleshooting the reason for poor tracking accuracy was necessary. Eventually, the solution was found by manually adjusting each of the parameters of the particle filter.

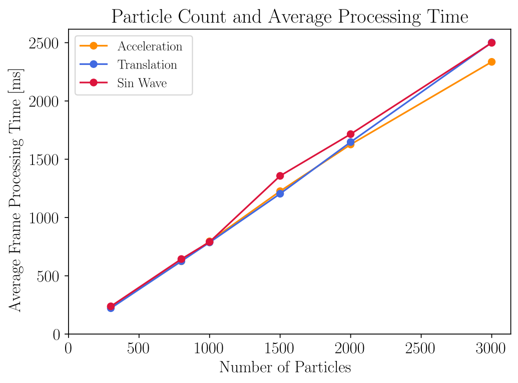


Figure 4: The average processing time of each frame increases as the number of particles increases. A similar relation holds for all three types of motion.

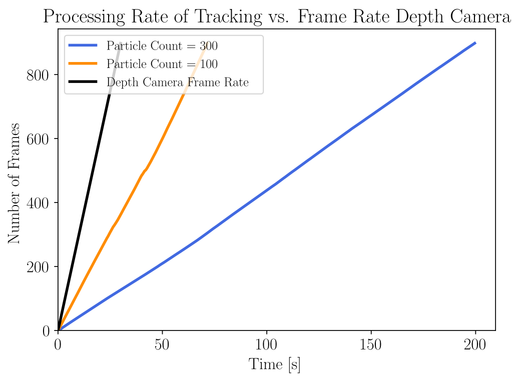


Figure 5: The frame rate of the virtual depth camera is approximately 30 FPS, similar to that of a real Kinect. We compare the frame rate of the depth camera to the processing rate of the tracking program, when different numbers of particles are used.

Parameter	Default Value
Downsampling Level	0.02
Number of Particles	2000
Variance	0.08
Delta δ	0.99
Epsilon ϵ	0.02
Sample Bin Size	0.1

Table 1: Default parameters for the KLD filter during experiments for tracking accuracy and computational time

3.3 Network Interface Module

To ensure that virtual and real devices can easily interface with our tracking program, we implemented a corresponding communication interface for the PCL tracking program. In the current implementation, the interface is tailored toward virtual devices in the physics engine, but it can easily be modified to accommodate real devices.

The interface features two communication channels based on UDP, as shown in Fig. 2. The first channel is used by the physics engine to send point clouds to the tracking program. Note that this can easily be exchanged with a real device which may produce other data formats. The physical device only needs the capability to engage in UDP message transfer. The second channel enables the tracking program to transmit predicted positional data, such as $(x, y, z, roll, pitch, yaw)$, or the full particle cloud. In the current implementation, this information is sent back to the physics engine. The physics engine does not use this predictive feedback, as the ground truth is known. However, in real applications, the predictive feedback from the tracking program can be transmitted to the master domain to guide the operator.

Furthermore, the choice of UDP as the communication protocol ensures independent speed and efficiency between the programs. TCP would guarantee no loss of data and synchronization between the programs, but it would create interdependency in terms of processing rate, which does not reflect real physical TI applications. Real TI applications involve the Kinect sensor tracking moving objects while the tracker program may provide delayed information. TCP cannot accurately simulate this behavior.

Additionally, packet loss was observed on the PCL side due to asynchrony between tracking and Unity's frame rate. The current implementation does not mitigate this. Nonetheless, a possible solution would be for the tracking program incorporates a queue to store incoming cloud data, preventing data loss.

4 Results

4.1 Experiment Set-Up

Computing Platform

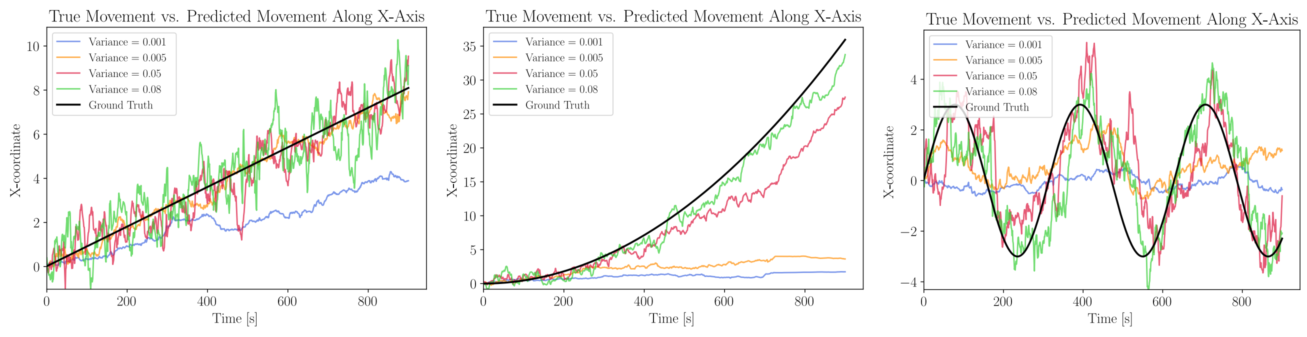
For the following experiments, we used a computer with Intel(R) Core (TML) i7-10510U CPU, 16GB RAM, and NVIDIA Quadro P520 GPU. The operating system used is Windows 11 and we implemented the virtual camera in Unity version 2021.3.23. To build the particle filter for tracking, we used the Point Cloud Library version 1.10.

Experiment Design

To evaluate our system, we initially defined three distinct types of motion that our target model, the Suzanne Blender monkey, should perform. These motions are as follows:

1. **Translation**: The model translates at a fixed speed along a predefined axis.
2. **Acceleration**: The model moves along a predefined axis at an increasing speed.
3. **Periodic Movement**: The model moves along a fixed path defined by the sin function.

The sequences of point clouds generated by the model when performing these motions were recorded. Thereafter, each frame or point cloud was sequentially processed by the PCL tracking program. For evaluation, we used the ground truth $(x, y, z, roll, pitch, yaw)$ coordinates of the object and compared it directly to the predicted $(x, y, z, roll, pitch, yaw)$ values. We focused our analysis on the axes exhibiting movement, as the object's motion was constrained to specific directions. Subsequent plots compare the predicted centroid coordinate x and the true centroid coordinate x, further analyzed in Section 4.3.



(a) Translation **(b)** Acceleration **(c)** Periodic Movement

Figure 6: Each track represent the predicted x-position of the object over $900\,\mathrm{ms}$**. Multiple tracks are plotted, each with a higher level of variance for the KLD particle filter.**

Furthermore, we examined how the parameters outlined in Section 3.2 influenced tracking accuracy. The set of default parameters used during these experiments is detailed in Tab. 1. For the experiment involving translation, the default value of the *variance* parameter was adjusted from 0.08 to 0.005. This modification was necessary as the initial variance value introduced unnecessary noise.

We also analyze the trade-offs which exist between using varying resolutions of depth cameras. The difference in tracking accuracy and processing time is compared across five cameras with different resolutions. This gives insight into the type of depth camera required or available options to consider when designing the real physical TI application.

Finally, we set the frame rate of the virtual depth camera in Unity to be approximately 30 - 60 FPS, similar to that of a real Kinect [22]. The tracking program cannot process frames at a similar rate, as described in Section 4.2. Thus, to evaluate the tracking program in isolation, it was necessary to provide the program with a recorded sequence of point clouds and not execute both programs simultaneously. For future work, it is worthwhile to adjust the tracking program to buffer a large number of incoming frames from the virtual depth camera, so that point clouds do not need to be saved before running the program.

4.2 Computational Efficiency

In this subsection, we evaluate the runtime of the tracking program and verify the impact of several parameters on the total efficiency. Thereafter, we compare the processing rate of the physics engine and tracking program.

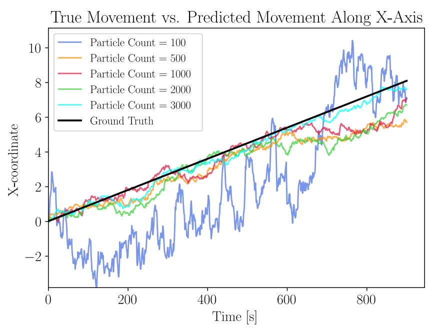
Fig. 4 shows an approximately linear relationship between the number of particles and the runtime of the tracking program. Note that the average processing time is the average time in *ms* to process each incoming frame and produce a predicted position. The same relation holds for all three types of motion and the average processing time is also similar. This means that for the particle filter, tracking objects with variable velocity is not more computationally expensive than tracking objects with constant velocity. Moreover, this linearly increasing trend is expected as with a greater number of particles, more calculations need to be performed per iteration. Fig. 5 further corroborates this result. The processing rate increases dramatically when the number of particles is decreased from $N = 300$ to $N = 100$. In addition, Fig. 5 also shows that the frame rate of the depth camera greatly exceeds the processing rate of the particle filter. In fact, the frame rate of the depth camera is at least twice the processing rate of the tracking program. This suggests that when running the full system, the tracking program cannot produce predictive feedback in time. Ideally, the processing rate of the tracking program must be greater than the frame rate.

To mitigate the low processing rate, reducing the number of particles is a potential solution. However, this compromises tracking accuracy (see Section 4.3), which may be unacceptable when high precision is required. Another option is to decrease the number of points in the initial point cloud model by downsampling, simplifying the tracked object's overall representation. In this way, fewer particles are needed to match the target object. Nonetheless, this may introduce errors in subsequent position or centroid calculations if the representative initial points are not selected carefully.

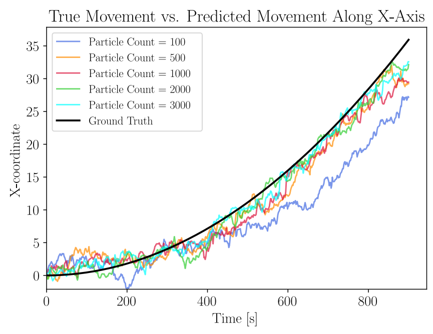
4.3 Tracking Accuracy

In this subsection, we discuss how each of the aforementioned parameters in Section 3.2 can affect how accurately motions can be tracked. The influence of the parameters variance, particle count, and downsampling level are analyzed in respective order.

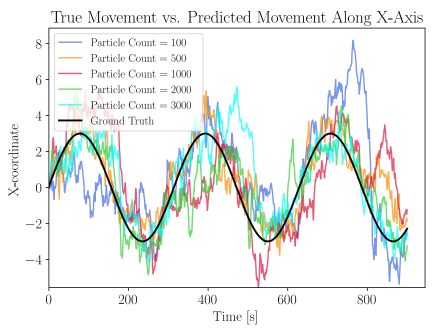
Variance: To examine the impact of variance on tracking accuracy, we conducted experiments using four distinct magnitudes of variance, as illustrated in Fig. 6. In Fig. 6a, the target object moves along the x-axis with constant velocity. A relatively low variance of 0.005 is acceptable, as demonstrated by how the track with variance 0.005 follows the true movement closely. While a variance of 0.001 significantly underestimates the magnitude by which the object moves per frame, higher levels of variance (> 0.005) introduce an unnecessary amount of noise. Hence, for objects with constant velocity, it is crucial that the magnitude of the variance suits the distance by which the object moves per frame. The exact numerical relation between variance and velocity is not known, as this is not documented in PCL [20] and requires numerous empirical experiments. The impact of variance is



(a) Translation



(b) Acceleration



(c) Periodic Movement

Figure 7: The same motions are tracked but with varying number of particles. Each track demonstrates how the predictions of the KLD filter evolve with a different number of particles used.

most notable when the velocity of the target object is not constant. When the object accelerates as in Fig. 6b, the track with the highest variance follows the true movement closely, while tracks with lower levels of variance deviate greatly from true movement. The same trend is observed in Fig. 6c. Lower levels of variance simply fail to capture the true movement of the target, whereas the track with variance 0.08 detects directional changes with minimal delay.

The trade-off, therefore, is that tracks with higher variance may result in higher errors at particular frames, so-called local errors, but are less likely to completely deviate from the true movement, while smaller variance results in smaller local errors but may fail to track the target. This can be attributed to how the particle filter is unable to generate samples that match closely with the true point cloud when the variance is not sufficiently large. Smaller variance corresponds to a narrower Gaussian distribution for each DoF, meaning that the particle filter may incorrectly predict that the next position deviates very little from the current position. A larger variance corresponds to a wider Gaussian, but it can also create samples that overestimate the amount of movement. This is a limitation inherent to the particle filter algorithm itself. If the particle filter does not generate samples that match the true point cloud, tracking will simply be unsuccessful. It is also impractical to determine the level of variance beforehand as the movement of the target object is unknown. Hence, the trade-off between local errors and the ability to track the general trend needs to be considered.

Particle Count: The number of particles refers to the number of samples taken per iteration of the algorithm. From Fig. 7, the tracks generated with $N = 500$ to $N = 3000$ have comparable accuracy. The track with $N = 3000$ particles follows the true movement most closely, but the gain in accuracy is negligible compared to $N = 2000$ or $N = 1500$. Furthermore, the number of particles also does not affect the noise of the predictions, meaning that local errors are generally of similar magnitude. Significant differences only occur when $N = 100$ particles are used. This likely results from a lack of indicative samples being generated, hence causing the predicted position of the object to be erroneous. It is also reasonable that $N = 500$ particles prove to be sufficient for capturing the point cloud, as the input point cloud is approximately the same size.

Downsampling Level: Downsampling is the process where some original points in the point cloud are removed prior to using the particle filter and re-estimating the weights for the particles. This ensures that there are fewer computations per iteration and improves the efficiency of the algorithm. Higher downsampling levels indicate that more points are removed.

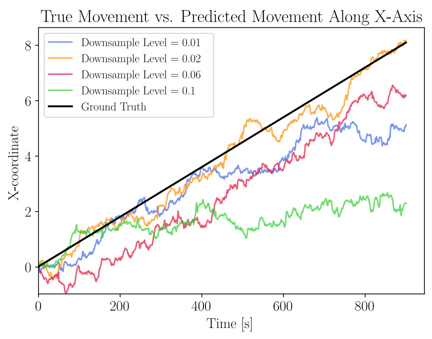
From Fig. 8a and Fig. 8b, a downsampling level of 0.10 results in poorer tracking accuracy, while other levels of downsampling show comparable tracking accuracy. It is expected that a higher level of downsampling results in lower tracking accuracy. If a significant number of points are removed, the particle cloud cannot capture the full shape of the target object. In this way, the shape of the particle cloud might be incorrectly biased, and the predicted centroid to have an offset. The effect of downsampling is less pronounced with periodic movement, as shown in Fig. 8c. This can be attributed to the particle cloud's ability to track the overall periodic motion and detect directional changes of the target object. The effect of downsampling is only noticeable at the peaks of the sinusoidal waves, where the track with the least downsampling follows the true movement most closely.

Furthermore, the effects of downsampling may not be as significant since the generated point clouds from the virtual scenes are not sufficiently large. Real Kinects typically generate point clouds of approximately 300K points [22], as opposed to the 500 to 800 points used in these experiments. If the number of point clouds increases, the effect on accuracy and runtime is similar, but of a larger magnitude [2].

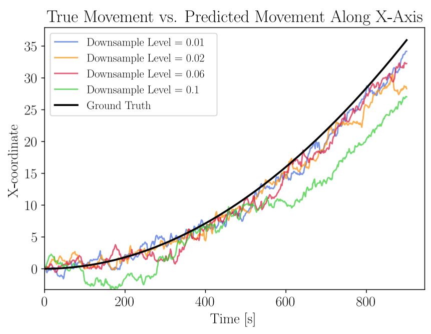
4.4 Configurable Depth Camera

To analyze how camera or sensor quality impacts tracking accuracy and execution time, we recorded the same sequence of translation motions using cameras with resolutions (width $\times$ height) : 30×40, 40×50, 50×60, 60×70, 70×80. Thereafter, we tracked the generated point clouds and compared the resulting accuracy and execution time.

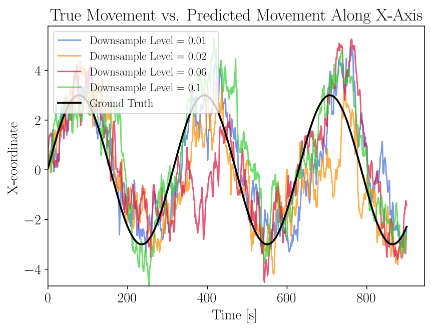
In Fig 9, the average L2-norm error between the predicted x-position and true x-position is shown to decrease as the resolution of the camera increases. Variance in the L2 error is also shown to decrease with higher-resolution cameras,



(a) Translation



(b) Acceleration



(c) Periodic Movement

Figure 8: For each track and type of motion, the input point cloud is downsampled at various levels. This investigates how downsampling can compromise tracking accuracy.

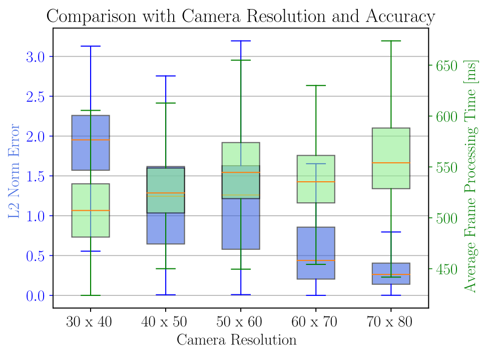


Figure 9: Average tracking L2 error is compared to average processing time for each frame, for cameras with different resolution levels.

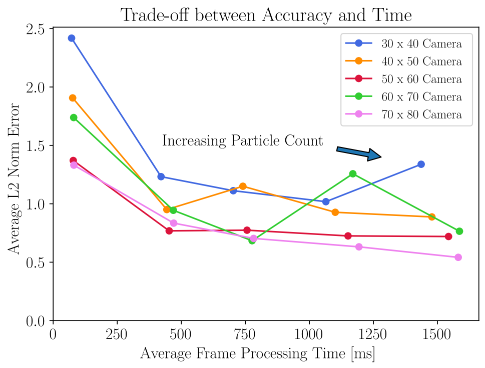


Figure 10: For each of the five possible cameras, we increased the particle count to observe how the L2-norm error and the average frame processing time change. All cameras show the same general trend: as the number of particles increases, the error drops but the processing time increases. The number of particles used (from left to right) is $\{100, 600, 1000, 1500, 2000\}$.

meaning that the filter can produce more stable and accurate predictions when the particle clouds are denser. With a denser cloud, the particle filter captures the geometry of the tracking target more accurately. Hence, the evaluation of the particles during re-sampling will be biased towards particles that truly match the true target. If the input point cloud does not fully capture the true target, evaluations, and re-sampling may incorrectly bias particles that do not closely match the target, resulting in poorer predictions.

Furthermore, execution times are fairly similar for all five cameras. This can be attributed to how the same number of particles are used for all five cameras. Fig. 10 further corroborates this claim. In Fig. 10, the L2-norm error and the frame processing time are plotted for each camera resolution as the number of particles increases. From left to right, the number of particles used is $N = \{100, 600, 1000, 1500, 2000\}$. The average frame processing time for a given particle count is approximately the same, irrespective of the camera resolution. This indicates that the runtime of the KLD filter is more dependent on the number of particles used as opposed to the dimensions of the input point cloud.

Furthermore, while the L2-norm error differs between the cameras, the overall change in accuracy becomes less significant from $N = 600$ onward. This experiment, therefore, suggests that an optimal trade-off between speed and accuracy is achieved at $N = 600$ for this specific motion and model. Using more particles than that will greatly increase execution time while offering negligible improvements in accuracy.

The accuracy of higher-resolution cameras dominates that of lower-resolution cameras. This is demonstrated by how the L2-norm error of the 70×80 camera is consistently lower than that of the 30×40 and 40×50 cameras. However, the distinction between the 50×60 camera and the 70×80 camera is much smaller. Thus, it can be argued that the 50×60 camera is already sufficient for tracking this particular type of motion. There is no clear trend for the camera with resolution 60×70 as it is neither fully dominating the lower resolution cameras nor comparable to the higher resolution cameras. We hypothesize that this results from the randomness in the KLD filter, as each data point is the average of only 6 independent runs.

Finally, this experiment serves as a proof of concept of how our platform can be used to choose appropriate hardware sensors. Real Kinect devices generate point clouds orders of magnitudes larger than what has been used in our experiment.

Nonetheless, our experiment shows how one can easily evaluate the performance and cost trade-offs between available devices, by tuning the quality of the virtual ones and without purchasing real hardware.

5 Responsible Research

Our platform is one of the many tools used in the development of general TI applications, which will enable human control of robotic devices through teleoperation. Hence, it follows that the ethical implications of the general TI systems are relevant to our project. Moreover, we believe that it is important to uphold the principles documented in [11] throughout the entire project, thus clearly documenting our results, data, and experimental process.

5.1 Ethical Implications of Tactile Internet Applications

TI applications will enable human control of robotic devices through teleoperation. Given that teleoperation encompasses tasks directly related to human lives and safety, such as telesurgery, healthcare, education, and exploration in possibly hazardous environments, the system must exhibit *reliability*. To this end, we performed various experiments and tests to ensure that our program works as intended, as demonstrated in Section 4.3. Furthermore, in instances of errors, human operators should be able to *understand* the causes behind malfunctions or deviations from the intended operations. Thus, we have designed our system to have a comprehensible interface, thereby helping future engineers to understand or find potential errors. Extensive documentation of our design and implementation choices is also publicly available in our online repository.

5.2 Reproducibility and Repeatability

To ensure the reproducibility of our results in Section 4, we have made all our code and data publicly accessible. The physics engine used to generate and transmit the sequence of point clouds also includes the code necessary to recreate our experimental data. The settings used for the experiments are documented and published. Additionally, we also published the code we used to generate the result plots shown in Section 4. However, it is important to note, that while our experimental data and code is available, future researchers may not achieve precisely identical results due to the randomness inherent in the KLD filter algorithm.

6 Conclusions and Future Work

In this work, we proposed a virtual platform for developing and evaluating 3D object tracking solutions for Tactile Internet (TI) applications. We also developed our own particle filter-based tracking solution and demonstrated how relevant parameters can be tuned to improve tracking efficiency and accuracy. Additionally, this platform proves to be a valuable tool for determining appropriate sensor hardware, as it allows one to analyze the performance trade-offs introduced by various configurations of virtual sensor devices. Therefore, our work has enabled experiments and evaluation strategies that were previously much more difficult with real-world set-ups and devices.

While our platform allows for experimentation with virtual devices and 3D object tracking, we acknowledge that the processing rate of our tracking program is insufficient for effective real-time tracking. The frame rate of a Kinect is at least double the processing rate of our tracking program, indicating that it will be a bottleneck for the full system if deployed as is. Hence, for future work, we recommend optimizing the particle filter implementation in the PCL library [20], so that the processing time of a point cloud is significantly reduced. Finally, we strongly encourage future researchers to incorporate other types of virtual sensors, such as an inertial measurement unit (IMU), and experiment with relevant sensor fusion algorithms. We hypothesize that this could lead to more accurate tracking, as additional information is used to guide the estimates of the particle filter algorithm.

References

[1] Alireza Asvadi, Pedro Girão, Paulo Peixoto, and Urbano Nunes. 3d object tracking using rgb and lidar data. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1255–1260, 2016.

[2] Kilian Van Berlo. Capturing real-time dynamic environments for tactile internet, 2022.

[3] A.I. Comport, E. Marchand, and F. Chaumette. Robust model-based tracking for robot vision. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 1, pages 692–697 vol.1, 2004.

[4] Frank Engelhardt, Johannes Behrens, and Mesut Güneş. The ovgu haptic communication testbed (ovgu-hc). In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6, 2020.

[5] Gerhard P. Fettweis. The tactile internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9:64–70, 2014.

[6] Dieter Fox. Kld-sampling: Adaptive particle filters.

[7] Vineet Gokhale, Kees Kroep, Vijay S. Rao, Joseph Verburg, and Ramesh Yechangunja. Tixt: An extensible testbed for tactile internet communication. *IEEE Internet of Things Magazine*, 3(1):32–37, 2020.

[8] David Held, Jesse Levinson, Sebastian Thrun, and Silvio Savarese. Combining 3d shape, color, and motion for robust anytime tracking.

[9] Siavash Hosseinyalamdary, Yashar Balazadegan, and Charles Toth. Tracking 3d moving objects based on gps/imu navigation solution, laser scanner point cloud and gis data. *ISPRS International Journal of Geo-Information*, 4:1301–1316, 9 2015.

[10] Michael Isard and Andrew Blake. Condensation-conditional density propagation for visual tracking, 1998.

[11] KNAW; NFU; NWO; TO2-federatie; Vereniging Hogescholen; VSNU. Netherlands code of conduct for research integrity, 2018.

[12] Chao Liu, Jing Guo, and Philippe Poignet. Nonlinear model-mediated teleoperation for surgical applications under time variant communication delay. volume 51, pages 493–499. Elsevier B.V., 1 2018.

[13] Eiji Machida, Meifen Cao, Toshiyuki Murao, and Hiroshi Hashimoto. Human motion tracking of mobile robot with kinect 3d sensor. In *2012 Proceedings of SICE Annual Conference (SICE)*, pages 2207–2211, 2012.

[14] Martin Maier, Mahfuzulhoq Chowdhury, Bhaskar Prasad Rimal, and Dung Pham Van. The tactile internet: vision, recent progress, and open challenges. *IEEE Communications Magazine*, 54(5):138–145, 2016.

[15] Probal Mitra and Günter Niemeyer. Model-mediated telemanipulation. *The International Journal of Robotics Research*, 27(2):253–262, 2008.

[16] Leonam S. D. Pecly, Marcelo L. O. Souza, and Keyvan Hashtrudi-Zaad. Model-reference model-mediated control for time-delayed teleoperation systems. In *2018 IEEE Haptics Symposium (HAPTICS)*, pages 72–77, 2018.

[17] Kurian Polachan, Joydeep Pal, Chandramani Singh, T. V. Prabhakar, and Fernando A. Kuipers. Tcps-bed: A modular testbed for tactile internet-based cyber-physical systems. *IEEE/ACM Transactions on Networking*, 30(2):796–811, 2022.

[18] 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 and Tutorials*, 23:472–523, 1 2021.

[19] Haozhe Qi, Chen Feng, Zhiguo Cao, Feng Zhao, and Yang Xiao. P2b: Point-to-box network for 3d object tracking in point clouds, 2020.

[20] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[21] Jingzhou Song, Yukun Ding, Zhihao Shang, and Ji Liang. Model-mediated teleoperation with improved stability. *International Journal of Advanced Robotic Systems*, 15, 3 2018.

[22] Zainab Namh Sultani and Rana Fareed Ghani. Kinect 3d point cloud live video streaming. volume 65, pages 125–132. Elsevier, 2015.

[23] 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.

[24] Victor Vaquero, Ivan Del Pino, Francese Moreno-Noguer, Joan Sola, Alberto Sanfeliu, and Juan Andrade-Cetto. Deconvolutional networks for point-cloud vehicle detection and tracking in driving scenarios. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7. IEEE, 2017.

[25] Changqing Zhou, Zhipeng Luo, Yueru Luo, Tianrui Liu, Liang Pan, Zhongang Cai, Haiyu Zhao, and Shijian Lu. Pttr: Relational 3d point cloud object tracking with transformer, 2022.

A Appendix A

There are two filters implemented in the PCL library. The other filter is the regular particle filter which offers worse performance. This is demonstrated by Fig. 11.

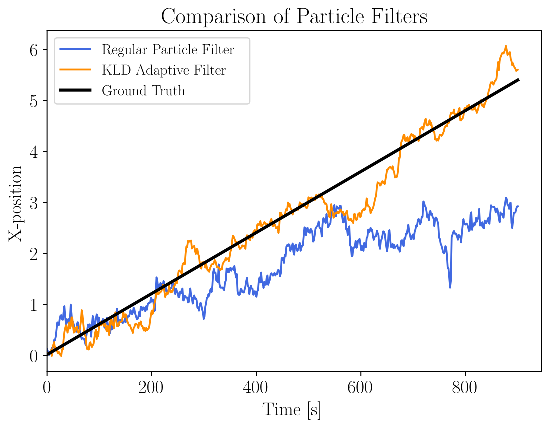


Figure 11: Comparison between available filters.

Full code can be found here. Note that the names of repositories may change, but the projects will be on the same account.

- Tracking Program: https://github.com/victoriayuechen/tracking-physics
- Virtual Depth Camera: https://github.com/victoriayuechen/My-project