Capturing real-time dynamic environments for Tactile Internet

by

Kilian van Berlo

to obtain the degree of Master of Science in Embedded Systems
at the Delft University of Technology,
to be defended publicly on Monday, November 21, 2022, at 08:30 AM.

Student number:	5436737	
Project duration:	March 1, 2022 – November 21, 2022	
Thesis committee:	Dr. R. R. V. Prasad,	Associate Professor, TU Delft, supervisor
	Dr. M. Weinmann ,	Assistant Professor, TU Delft
	K. Kroep,	PhD Candidate, TU Delft
	Dr. V. Gokhale,	Postdoctoral Researcher, TU Delft

This thesis is confidential and cannot be made public until December 31, 2024.

TU Delft

Preface

Humans interact more and more at a distance these days. Where currently, this interaction is mainly visual and auditory, Tactile Internet (TI) also allows for remote kinesthetic interactions. With the development of TI, a paradigm shift is on the horizon where the current internet evolves from content delivery to a skill-set delivery network. Imagine the possibilities when specialists in whatever field can, in real-time, apply their skills everywhere in the world from within their living room. To make this a reality, TI applications have to be able to provide feedback to the user with ULL. The speed of light, however, constrains the extent to which TI can be used at large distances. This thesis hence takes a closer look at the problems that arise when long-distance TI is to be used and tries to tackle them. These problems are tackled by building a system based on Model-Mediated Teleoperation (MMT), where local simulations of the environments are used. A local feedback loop is instantiated through these local simulations, which lowers the system's dependence on the network performance. The former control loop, including the network, now becomes a corrective loop to prevent mismatches. This thesis focuses on the part where the local simulation is corrected by the real-life data received over the network. The question is how real-time dynamic environments can be captured and kept track of for use in MMT. We identify accurate tracking of objects in the environment as the critical building block enabling MMT with dynamic objects. Consequently, the first 6 Degrees of Freedom (DoF) tracking application designed for use in TI applications is created and evaluated. Through this research, the first steps are set towards a full-blown MMT system that can capture the remote environment successfully.

Kilian van Berlo
Delft, November 2022

Contents

1 Introduction

In the past couple of centuries, humans have revolutionized their approaches to remote interaction tremendously. Since the invention of the telegraph machine about two hundred years ago, we have managed to speak and see each other remotely in real-time. Do you need to educate people on the other side of the world about your newest invention? Send them a digital meeting invite, and within seconds you can tell and show them everything about it. However, sometimes more than hearing and seeing is needed. For example, when surgery has to take place, that can only be done by a few surgeons/experts worldwide. That expert being able to see the situation and tell what to do is, in all likelihood, not enough, as the expert must perform physical actions. Therefore, the expert has to travel to where the surgery is executed, losing valuable time that could have been spent more efficiently.

This valuable time does not have to get lost anymore as Tactile Internet (TI) revolutionizes interaction as we know it. Through building a solid foundation, TI can accelerate developments in Industry 4.0, making it possible to speak, see and feel things at a distance in real time. With TI, surgery could be executed remotely through teleoperation. In such a situation, the surgeon would not have to travel everywhere and could apply their skills from a single location. TI facilitates physical interaction with humans and machines as if they were near, enabling real-time remote control and physical, tactile experiences. Through TI, the exchange of simple touch-based gestures, such as handshakes, high-fives, and even hugs, can be facilitated. Besides, it enables us to seamlessly transfer skills over the internet (such as the surgeon's). Hence, TI shows the potential to provide a true paradigm shift from content-delivery to skill-set delivery networks and is, therefore, able to revolutionize almost every segment of society.

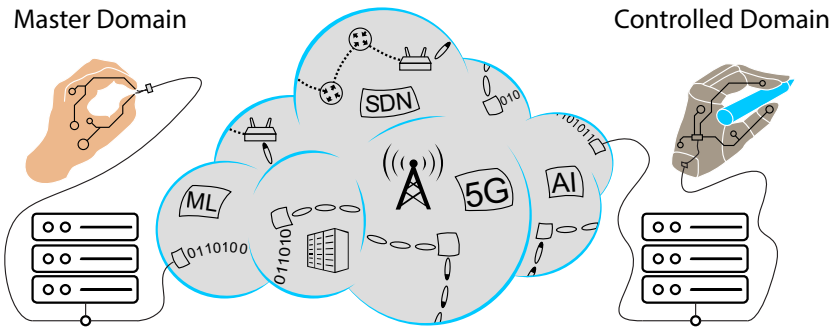


Figure 1.1: A schematic representation of Tactile Internet showing the master and controlled domains with the network domain in-between [1]

Not only a surgeon can reap the benefits that TI provides. Search and Rescue operators and astronauts do not have to get into dangerous situations anymore and the entertainment industry will be able to, now more immersively, mix the virtual world with reality. While the possible applications of TI that one can think of are endless, the primary system behind each use case shows to be similar. As shown in Figure 1.1, a TI system consists of three distinct domains: a master domain, a network domain

(the cloud in the image), and a controlled domain. In the telesurgery example mentioned earlier, the master domain would be the expert controlling the robotic hand with which the surgery is performed. This robotic system is part of the remotely controlled domain communicated over the network. Thus, when the expert uses his hand during surgery, the robotic hand will receive data indicating what the hand is doing and replicate those actions. Once the controlled domain mimics the action, the data acquired on that side is used to provide feedback. This feedback is sent over the network to the operator on the side of the master domain.

The promising benefits of TI accompany several enormous system design, development, and realization challenges. The most important of these challenges is the Ultra-Low Latency (ULL) requirement of a round-trip latency of $1\,\mathrm{ms}$ [2]. That means that the data transmission latency from one domain to the other domain and back should be within the $1\,\mathrm{ms}$ range to allow an optimal quality of teleoperation. This requirement might feel exaggerated; however, as studies have shown, humans can distinguish this order of magnitude for tactile sensing [3]. When the surgeon, for instance, has to make a small incision without damaging the lower tissue layers, data about force feedback must be fed back in time [1]. The surgeon applies a force sent to the controlled domain, and the controlled domain sends back the force feedback. If, for example, the surgeon cannot receive the haptic feedback in time, it will not notice when the incision becomes too big, leading to possibly disastrous consequences.

Next to ULL, also Ultra-Reliability (UR) and throughput are considered important aspects in this field [4]. However, since the throughput in TI applications is relatively small compared to already existing applications, this should not be of any issue. For UR, the TI community recommends a requirement of up to $99.9999\,\%$ [5, 6]. However, unlike the ULL requirement, which has been quantified through experiments, the requirement of UR is mainly speculation based. At the time of writing, only one recent work has applied rigorous experiments on this topic. In this research, the results do not support the requirement claim and even reveal that a much lower level of reliability, under $20\,\%$ of packet losses, is also sufficient [1].

UR and throughput do not seem to be the main culprits for TI. Thus, the main focus of this work is on meeting the ULL requirement of the round-trip time. To achieve this, one might expect it is solely a matter of diminishing latency times in the network domain. Despite the outstanding advancements in 5G and beyond, the ultimate limit for latency will still be set by the finite speed of light. Speed of light is limited at $300\,\mathrm{km/ms}$, and hence even when the latency is optimized, TI applications will still be unable to function correctly over large distances. Take Figure 1.2, where a finger presses a button. This finger is part of the controller at the side of the master domain. In this domain, it attempts to push a button on the side of the controlled domain. When the latency is within the defined limits, a situation as shown in Figure 1.2 a occurs. For example, when a setup exists between Amsterdam and Rotterdam, the movement of the finger would be mimicked by the controlled domain. The controlled domain would, in turn, send force feedback back to the finger in the master domain, all within $1\,\mathrm{ms}$. However, in case a setup is created between Amsterdam and London, or even further, network latency becomes too big. A situation like this is envisioned in Figure 1.2 b, where the finger will substantially push past the point of contact since the force feedback is received too late.

Because of the delayed force feedback, the controlled domain will perceive the finger to be applying more force than initially planned. Therefore, the controlled domain will push the button further than intended or supply a more significant feedback force. In the first case, a situation with the surgeon cutting too deep would occur. In the latter case, the controller could become unmanageable as it cannot handle the feedback force properly and attempts to correct it. The situation sketched here leaves us with two options: deploy TI only over small distances or think of another approach that circumvents the natural speed limit. As one might expect, researchers do not take no for an answer; thus, several radical approaches for TI have been invoked already. With these approaches, researchers attempt to accelerate further the required paradigm shift toward realizing seamless teleoperation with haptic feedback.

One of these possible approaches being researched to achieve seamless teleoperation is called Model-Mediated Teleoperation (MMT). This approach builds on the idea that if remote interaction with the controlled domain takes too long, the controlled domain can be interacted with locally. The difference between the classical control loop, as shown in Figure 1.1, and MMT (Figure 1.3) is in the fact that MMT makes use of local simulations. These local simulations are models of the domain on the opposite side of the network (see the human and environment models) and are deployed to approximate that other side. By creating local models on either side, the system opens the control loop between

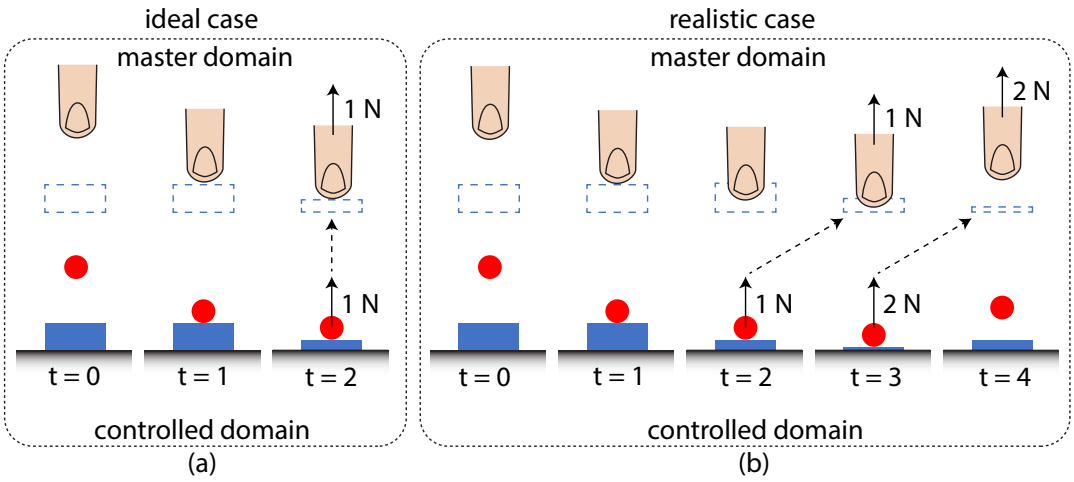


Figure 1.2: Illustration of the detrimental effects of the network on TI interaction. The operator intends to press a blue button in the controlled domain with his finger in the master domain. The red circle is the finger representation in the controlled domain, while the dotted blue button is the representation in the master domain. (a) The ideal case where the feedback loop happens within $1\,\mathrm{ms}$ (b) The realistic case where the feedback loop takes longer and significant performance degradation occurs. [7]

the master and controlled domain, leading to two decoupled local control loops supported by the original, overarching control loop. Through the local models, uninterrupted data is presented, of which the perceived delay is unaffected by the network. The models on either side would still use the network's data; however, now, it only serves for corrections on the simulations.

Given that the goal is complete immersion and functionality for the master in the controlled domain, the local simulations will not be shown visually to the user. These local simulations will instead serve as an underlying invisible layer. This approach with invisible layers will allow the system to eventually fix divergences in the background without making the user aware of these fixes. In the case of the surgery, when making an incision, this would result in the expert initially receiving force feedback based on its interaction with the local model. Because it interacts with the local model, the interaction does not have to go over the network before feedback can be received, leading to an interaction without noticeable latency. In the meantime, corrective data is still provided over the network. Based on this data, the mismatch between the simulation and the real-life environment is handled in the background. By guiding this collaboration between simulation and real data, a situation is maintained in which the surgeon can successfully implement their skills.

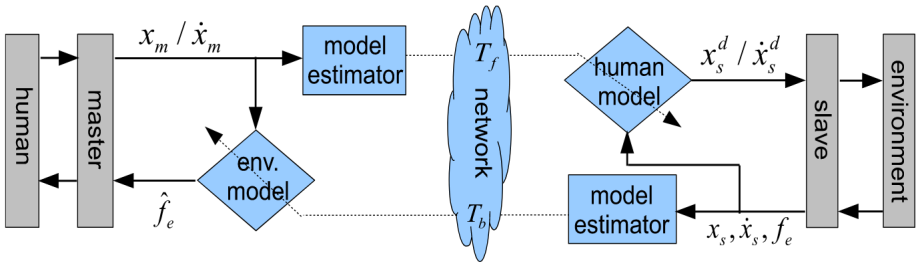


Figure 1.3: State-of-the-art MMT control loop where there is a local control loop on either side. One loop is between the human (controller) and the environment model, and the other is between the environment and the human model. The model estimator, in turn, is part of the overarching control loop and hence gathers information about the controlled environment on the one side and information about the controller on the other. This information is sent over the network to support the local models on both sides. [8]

When making use of a local control loop, latency can become unnoticeable. Unnoticeable latency indicates that if the estimated models are accurate enough approximations of the actual domains, functional long-distance teleoperation can be achieved. To achieve this, however, it is of utmost importance that the local model is a good enough representation of the respective environment. In principle, the better the match between the local model and the controlled environment, the more stable the teleop-

eration will be in arbitrary communication latency. However, the problem is that a perfect match cannot be obtained; hence, the master and the slave environment model sometimes mismatch. This mismatch can be caused due to several situations, such as changes in the environment, operator commands and movements, improper model approximations, or inefficient parameter estimation methods.

Due to the probable mismatch in the model and environment, adequately correcting these mistakes is crucial. In MMT, for mismatch correction, the overarching control loop comes into play, through which corrective data is provided. On the one hand, this is data about the controller physics from the master to the controller model in the controlled domain. On the other hand, data about the environment in the controlled domain is provided to the local environment model in the master domain. Both data streams are crucial elements for the proper functioning of the system. Acquiring data for the latter, the local environment model, can be done through online parameter estimation. According to the state of the art, online parameter estimation for environment modeling is the most critical and challenging task that MMT is currently facing [2, 9].

When modeling an environment, there is a distinction between static and dynamic environments. A static environment contains non-movable objects, while a dynamic environment contains movable objects. Static objects make modeling a static environment relatively easy compared to a dynamic environment, given that with non-movable objects, the critical point is to estimate the dynamics of the object at the area of contact. This contact area can only change to a limited extent over time; thus, constructing the environment model is only needed once. For movable objects in a scene, constructing a model is more challenging as the static object dynamics and the dynamics regarding interaction with the surroundings have to be modeled. Because movable objects require more parameters than non-movable objects, previous studies on MMT have, so far, mainly focused on static environments [9]. Motivated by this knowledge, the main research question of this thesis is:

> *How to capture real-time dynamic environments for use in Model-Mediated Teleoperation?*

Our approach looks at what is needed to quantify an environment reliably. Since an environment can be dynamic, it is vital to keep track of the changes at all times. Hence, most of the thesis is focused on how one can track moving objects in space. With the help of point clouds, multiple possibilities are explored for tracking in 6DoF. For the tracking algorithms, special attention goes to the opportunities that particle filters present; hence, several different implementations are objectively evaluated in terms of timing and accuracy performance. Eventually, once the tracking functionality is in place, it is combined with a TI application to make the first steps towards a complete MMT system. Once tracking and TI are combined, a small user study is performed to give a first indication of the requirements of the tracking performance for deployment in MMT systems.

In essence, this thesis intends to contribute to the relaxation of the timing requirements of TI. For this purpose, a 6DoF tracking module that keeps track of objects in a dynamic environment is theorized, implemented, and analyzed. This tracking functionality is eventually used to interact with a remote environment through MMT. The contributions from this work can be listed as follows:

1. We identify and demonstrate accurate tracking of objects in the environment as the key building block to enable MMT with dynamic objects

2. We develop the first 6DoF tracking application that is specifically designed for use in TI applications

3. We propose the use of a particle filter-based tracking algorithm to gather data about the remote environment. This data can then be used to correct for mismatches of the local model in the master domain of the MMT system.

4. We identify the performance trade-offs required when tracking objects, both in the static and dynamic situations with which MMT deals.

5. We evaluate the Quality of Experience (QoE) of an MMT system that uses object tracking and determine its dependence on variables such as delay and sensitivity.

6. We highlight the importance of fast-paced tracking algorithms for real-time tracking capabilities

7. We demonstrate that our particle filter algorithm outperforms the standard implementation in terms of noise elimination in static scenarios.

The rest of the report is organized as follows: Chapter 2 gives a first insight into the works related to this research. Chapter 3 presents the theory behind all the concepts utilized. Chapter 4 outlines how the project was performed. Chapter 5 highlights and discusses the most important results from the experiments conducted. Chapter 6 evaluates the project and briefly looks ahead to future opportunities.

2

Related Works

2.1. Tactile Internet

TI is a relatively new research area that has shown tremendous growth over the past decade. The idea of a TI was already being explored around the late 2000s and early 2010s. Around this time, researchers were looking into sending haptic feedback over a network [10, 11]. However, it was G. P. Fettweis who published the first article specifically focused on the applications and challenges of TI as a whole [5]. A few months after Fettweis' paper, he aided the International Telecommunication Union - Telecommunications (ITU-T) in publishing an extensive Technology Watch Report on the TI [12]. There on, it lasted only a short time before other researchers started to join in contributing to the field.

With latency turning out to be the biggest challenge facing the TI, much work is being put into restricting it. In the field of mobile networks, a significant part of current research in 5G is centralized around meeting the low-latency real-time requirements that the TI brings around [13, 14, 15, 16]. These latency requirements, however, are not realizable with just improvements to the network. As stated earlier, the speed of light is not fast enough for application over long distances. Hence, people have started to look at different approaches to tackle this challenge.

2.2. Model Mediation

One of the solutions proposed for satisfying the latency constraints in TI applications is to use Model Mediation. The idea goes back to the '90s when multiple papers were published with roughly the same idea [17, 18]. They proposed creating a local model to manipulate a system that experiences considerable latency when operated remotely. Then a leap in time occurred, and from 2004 onward, interest has been gradually growing in using model-mediation in teleoperation systems [19, 20, 21, 22, 23].

In recent years a group of researchers led by X. Xu has made numerous contributions to MMT. Not only did they provide clarity on the current state of the research domain, but they also performed studies that provide a stable foundation for this work to expand on further [8]. Some papers of Xu et al. specifically focus on using Point Clouds to aid model mediation [24, 25]. Another work of Xu et al. from 2015 [9] focuses on MMT for movable objects. Besides this work, to the best of our knowledge, there is no other comprehensive study available on MMT with freely movable objects. Hence, this work is regarded as the current state of the art concerning dynamic real-time environment modeling.

2.3. Environment modelling

An environment can be modeled using online parameter estimation, which is one of the most important, and at the same time, challenging tasks in MMT [9]. Online parameter estimation is the estimation of the model parameters (e.g., geometry, force, position, velocity) in real-time [8]. When modeling an environment, it is essential to know whether it is dynamic or static. A dynamic environment with movable objects logically requires dealing with more parameters than a static environment. Because of this, previous studies on MMT mainly focus on modeling the simpler, static environments without movable objects [25, 26, 27]. Little research has been conducted on a dynamic environment with movable

objects, as mentioned in Section 2.2. In 2012, Passenberg presented an algorithm that estimates object inertia in real-time [28] while Xu et al. described a more generic approach to modeling movable objects back in 2015 [9].

For creating a detailed environment model, it is vital to discover the environment, detect the individual pieces and their associated features, and eventually track these pieces. While for static environments, mainly the first two aspects hold, the third one, tracking, is mainly essential for a dynamic situation. Point clouds obtained through RGB-D cameras are a promising solution in both cases. For both discovery and detection, point clouds already acquired lots of attention in the field of teleoperation [29, 30, 31, 32], and also a bit in the field of MMT [24, 25, 33] specifically. Whenever objects are desired to be tracked in a dynamic environment, point clouds have not yet been considered much for this specific use case.

2.4. Object tracking

When dealing with dynamic environments, obtaining a good level of immersion and situational awareness becomes challenging. Physics models, combined with local models of the remote environment, might be excellent starting points [28, 9]. However, it is also essential to keep track of all the parts that make up that environment individually. While, as stated in Section 2.3, the domain of model mediation is barely invested in this challenge, another domain is to a certain extent. This domain is the domain of robotic sensing, where much attention goes to providing robots with capabilities to track objects, either in 2D or 3D. Since for MMT, it is crucial to interact with an object from all directions, 3D (or 6 DoF) object tracking is a must.

While many studies have looked for the best approach to object tracking in dynamic environments, a clear best solution has yet to emerge. Point clouds are currently the preferred technique for tracking an object in 3D space. However, there are two different approaches one can use with them, deep learning and classical methods with filters such as Particle and Kalman. While the classical methods started promising [34, 35, 36, 37], more and more attention has shifted to applying deep learning for object tracking. While some studies combine both approaches [38, 39], others have indulged in the space of deep learning completely [40, 41, 42].

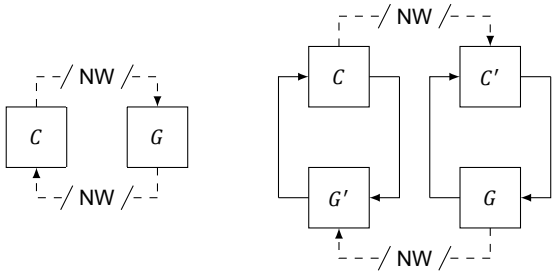
3

Theory

3.1. Model-Mediated Teleoperation

TI applications enabling haptic communications require correct and near-instant force feedback to have an acceptable Quality of Experience (QoE). As mentioned in Section 1, several networking conditions need to be met to make this a reality. While challenges such as throughput and reliability of the network are relatively easy to address, the opposite holds for latency. For TI applications such as teleoperation, this remains the fundamental challenge since, for long distances, the latency is restricted by the limitations of the speed of light [2]. This challenge can be tackled in two ways: sticking to scenarios where the distances are limited or finding an alternative approach that relaxes the latency requirements. One of these alternative approaches to circumvent the speed limitations is a concept called MMT.

As shown in Figure 1.3, MMT makes use of local models on either side of the system to bypass the effects of the adverse conditions of the network, such as significant latency. The local model on the master side simulates the environment in the controlled domain and vice versa. With these local models, critical interactions could occur in real-time on one network side. Information from the opposite domain could be used as corrective information for possible mismatches. Because this networked information is less critical for instant execution, it does not have the exact strict latency requirements as before. By applying this approach, the most critical communication takes place locally. Local communication makes a latency within the prescribed limitations possible, thus allowing for good force feedback. However, the advantage of this system, which is better equipped for handling significant latency, comes with the downside that it needs significant contributions to outperform the traditional methods. These contributions are needed because of the lack of research that has, so far, been performed on this method.



(a) Traditional tactile system (b) Model-mediated tactile system

Figure 3.1: Simplified Control Systems Flowchart

Taking a closer look, one can see in Figure 3.1a that the control loop of a traditional TI application has its critical connections running over the network (NW). A model-mediated application, however, as shown in Figure 3.1b, enables both sides to function as independent systems. By executing it this way, the local models can still utilize the data coming over the network; however, this data becomes

less crucial for execution. Figure 3.1 displays simplifications of both control systems, with the controller being represented by C and the remote environment being portrayed as G. In the improved, model-mediated version, C' and G' are also shown and serve as local representations of C and G respectively. C' is thus a local model of the controller on the controlled side. At the same time, G' is a local model of the remote environment on the master side. By adding these two local models, the controls can still get corrected with data from the domain on the opposite side of the network. However, they will also be able to keep functioning with more considerable data latency on the network than in the original setup.

A perfect local model of the environment (G') would, irrespective of communication delays, enable stable teleoperation [8]. The possibility for stable teleoperation means that, in theory, MMT successfully translates the latency requirement of the network to a model-fit requirement. With a better model, the model becomes more accurate, and the update frequency and efficiency can be relaxed for more extended periods of time. Once a perfect local model is created through data from the opposite side, it should be able to maintain this perfect fit through the seamlessly fitting physics engine. In practice, however, having a perfect real-time model proves impossible sometimes, and model mismatches occur inevitably. Because of these mismatches, it is also critical to consider how one wants to deal with them over time. This process raises three crucial questions, how to: create the best physics engine, create the best environment model, and deal with mismatches optimally. These questions are imperative for the design of MMT applications since the better they are solved, the more delay is allowed while maintaining adequate system performance. The research discussed here, as already mentioned briefly in Section 1, is mainly about the second question of obtaining a digital model of a real dynamic environment in real-time. This aspect is vital for initializing an environment and preparing for mismatch correction.

3.2. Tactile physics engine

Over the past decade, commercially available physics engines, such as the ones from Unity and Unreal, have improved tremendously. Despite the improvements, however, they still lack their tactile physics abilities. Tactile physics deals with fundamentally different challenges. Where standard physics engines require quick and efficient calculations for collision detection, tactile physics engines demand more precision. Next to precision, there are many more responsibilities that tactile physics engines have to bear. Tracking the position of the haptic controller and measuring and applying the forces registered are also important.

Traditionally, collisions are detected by registering intersections between two or more volumes. This intersection problem is then resolved by pushing the objects out in the most efficient direction. The problem with this approach for TI is that this method looks at the deepest point of penetration between objects instead of the first point of contact. The forces applied to these objects will look good. However, the haptics will be perceived as incorrect or noisy.

This project uses Unity as a base to create environments and extends it with functionality from the Chai3D tactile framework [43]. Chai3D is added to create well-functioning tactile physics. Unity, however, is generally not fast enough to comply with the $1\,\mathrm{ms}$ bottleneck that TI introduces. Therefore, Bullet Physics is also added to Unity, a physics engine fast enough for tactile applications [44]. This physics engine is extended with tactile functionality from the Chai3D framework to construct a complete tactile physics engine.

One of the core components that our physics engine reverse-engineered from Chai3D is the proxy algorithm. This algorithm is critical for creating realistic haptic experiences of objects in a virtual environment. The proxy algorithm determines how much force has to be applied when only the position of the haptic device is known. It represents the haptic device in the virtual environment through a proxy object. This object perfectly follows the haptic device when there are no collisions; however, it behaves differently when a collision is detected. In case a collision is faced, the haptics device penetrates the colliding object in the virtual environment while the proxy stays put at the first point of contact. The distance between the actual device, x_{device}, and the proxy, x_{proxy}, can be interpreted as a spring drawn between them. The bigger the spring constant, k, the more rigid the surfaces and, thus, the more sensitive the haptic device is to touch. The force, F, to be applied is determined based on this distance and the spring constant, calculated as in Equation 3.1.

$$F = (x_{\text{device}} - x_{\text{proxy}})k. \tag{3.1}$$

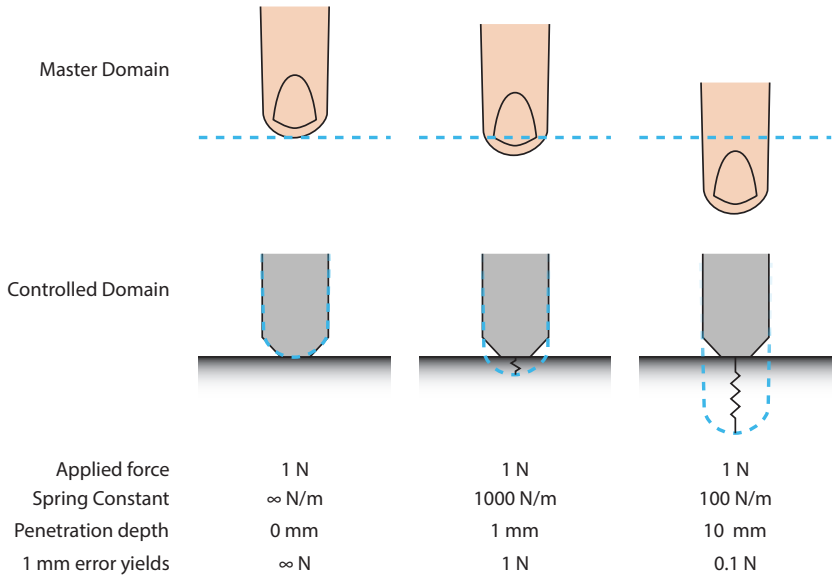


Figure 3.2: TI interaction where an operator pushes on a virtual surface (blue dashed line) in the master domain. The haptic device presses down on a real surface in the controlled domain. The proxy algorithm spans a spring to calculate the force applied on the surface. [7]

In figure 3.2, an illustration of the proxy algorithm shows how the applied forces are determined and how the spring constant affects these forces. A more significant spring constant feels more realistic; however, it comes at the cost of higher force errors for relatively minor mistakes. This part of the algorithm makes up a small part of the C' functionality envisioned for the controlled domain in the MMT system shown in Figure 3.1b. In a more complex control system, a more straightforward C' model dealing with virtual physics would also be visible on the master side.

3.3. Environment modelling

When a remote environment G has to be modeled, this requires sensors to extract information from the real-life environment and send this information to G' in the other domain. Sensors are essential to creating an initial idea of what the environment is like without having yet interacted with it. If humans did not infer details of an environment before interaction, life would be much more complicated. Imagine seeing a glass filled with water but having no idea what the weight will be; before you know it, the water is on the ceiling, and you are left thirsty. If there were a way to estimate the weight of the glass beforehand, you would not have exercised excessive force when lifting the glass. Luckily, our brains can make these inferences; computers do not so quickly. Of course, interacting with the environment will support the creation of an even more detailed model. However, by adding non-interaction-based data to the mix, most of the information about the environment can already be extracted and will serve as both a great starting and reference point for model generation.

The goal of this work is to generate the best possible approximation of an environment before having any interaction with that environment. When the environment is observable, before manipulating anything in the surroundings, it should be possible to get a solid idea of everything that shapes the scene. When creating a detailed model of an environment and keeping track of it, three steps are essential in the process: discovery, detection, and tracking. With the discovery, a detailed overview of the scene is obtained; detection is about identifying individual pieces within the scene, while tracking is about keeping track of these individual objects in the current surroundings. All three steps should be approached differently when modeling the environment.

Discovery is currently the step that academia focuses on in the area of MMT. Even the state-of-the-art, as discussed in the work of Xu et al. [9], does not go far beyond this step. The idea of discovery is to perform online parameter estimation of the surroundings through which a good overview of the scene is created. Within this scene, objects are identified, and their characteristics, such as e.g. weight, material, or size, are determined. It is crucial to be accurate in the estimation efforts as, this way, the

discovery phase can be executed well. The more detailed the information is, the better an environment can be modeled. Since discovery is not necessarily connected with direct interaction, the timing constraints are less critical. Discovery, in most cases, takes place the moment a new element of the surrounding shows up, which is most likely to happen when the object is not yet close enough for interaction. Hence, in that instance, it makes more sense to take longer for the estimation to achieve a higher level of detail. In this project, the discovery phase of environment modeling does not receive much attention as it is not one of the main goals. Since many aspects are contained in the eventual PoC, parts that can be circumvented will be circumvented. Given that the discovery phase is one such part, it is decided to mimic the functionality by having a known environment and populating a library with the objects in that environment and their corresponding parameters. Doing this makes discovery redundant, as the situational details are already known. As a result, they provide more time to focus on other aspects of the process.

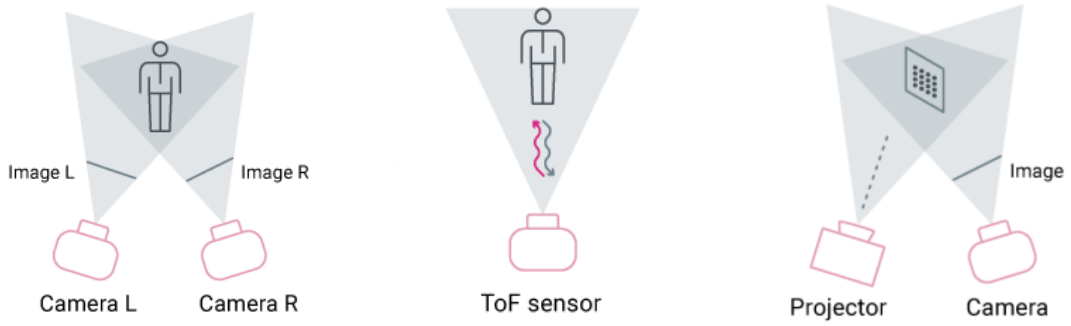
Once the surroundings are discovered, or in the case of this project, predetermined, the detection phase starts. The state of detecting objects in the environment is not desired to be as accurate as when discovery takes place since less time is available. Less time is available because detection is mainly needed when an object gets in the range of possible interaction, unlike discovery. As mentioned, discovery starts when new elements show up. When an object is in the range of possible interaction, a more time-constrained approach has to be followed, and thus less accuracy is expected. Detection is an aspect of the process that can be imitated relatively quickly. Therefore, a workaround for this is also devised. The approach for detection circumvention is to use the predetermined library with objects and go through it to select the desired object from the list. A 3D model of that object can be placed at a starting point in the environment. Based on this starting point, the next step in the process, tracking, knows where to be. The way these 3D models are created and prepared for deployment in the local model is discussed in the next Section (Section 3.4).

While for unknown environments, the tricks used for discovery and detection will not cut it, the third crucial step, tracking, creates a hurdle that, even for familiar environments, takes work to overcome. This work focuses primarily on tracking objects within a scene, which has only been done to a limited extent in the MMT domain. As tracking is primarily needed when objects can move within a scene, it makes sense that this aspect is missing in research as the most focus has been on static scenes. In more detail, how object tracking is approached will be discussed in Section 3.5.

Suitable sensors will be required to successfully model a real-life environment and keep track of everything happening. Hence, some light must be shed on the sensors used for the process. The type of sensor to use depends on the parameters desired for the environment model. For this project, the main desired parameters are location and rotation; the rest, as already mentioned before, is predetermined to be able to focus entirely on object tracking. Location and rotation are crucial parameters because it would be impossible to track an object within a 6 Degrees of Freedom (DoF) field without them. To extract this information from the environment, popular sensing equipment that can be used is either a conventional camera, an RGB-D camera, or a LiDAR sensor. With a conventional camera, assuming only one is used, 2D scene information can be collected. 2D compared to 3D information strongly restricts the data collection abilities during scene capture and makes the time constraint for real-time scenarios less easily attainable. More processing is required for 2D data to attain the same level of scene knowledge.

When it comes to using 3D sensing technologies, there are three prominent techniques to consider: using Stereo Vision, Time of Flight (ToF), or Structured Light. As shown in Figure 3.3a, Stereo Vision uses two slightly apart sensors, like our eyes. Using these two sensors ensures that two slightly different images are created, which are then compared, and since the distance between the two sensors is known, depth can be inferred. ToF works differently where the technique depends on how fast light returns after being emitted by the sensor, as shown in Figure 3.3b. The third option, structured light, is seen in Figure 3.3c and works in a way that a structured image (e.g., a grid) is projected onto an object, and based on the distortion of this image, the shape and distance of the object are calculated.

All three options can capture depth information in a scene, which can then be used to create point clouds. These point clouds are data points in 3D space, and because they provide depth measures, further understanding of the scene can be achieved. Hence, as one has already seen in the sources referenced in Section 2, 3D sensing technology is currently the go-to approach in almost all recent studies related to 3D environment modeling. It is also clear that for 3D modeling, point clouds show great promise for MMT due to their ability to provide depth information about a scene and its arbitrary



(a) Stereo vision (b) Time-of-Flight (c) Structured light

Figure 3.3: Different types of 3D sensing technologies [45]

objects. However, how valuable point clouds can eventually be is still not fully determined. A MMT system making use of point clouds has already been touched upon briefly by existing research and is called Point Cloud Based Model-Mediated Teleoperation (pcbMMT) [25]. Nonetheless, state-of-the-art has not yet managed to get far past the step of environment modeling; hence, opportunities lie in the space beyond, the space of tracking.

3.4. Object modelling

Before the details about tracking are discussed, it is essential to know how the models of the objects that make up the environment are constructed. A photogrammetry technique is applied to create a 3D model of an object [46]. Photogrammetry is a technique that uses 3D geometry derived from 2D images. If correctly done, these 2D images are a series of photos of an object from various angles with considerable overlap. Common points between multiple images are then identified using point matching. Next to the information about these common points, information about the pixel size, position and angles, lens distortion, and focal length of the camera are gathered for each image. In the case of more advanced photogrammetry, one could also use other information about the scene, such as symmetries. Every point in a 2D image specifies a light ray from the camera to the real point in a 3D space. Based on this information, a photogrammetry algorithm then uses a triangulation technique, which determines the geometric intersection of several light arrays. These intersections, in turn, help to locate the location of the corresponding point in a 3D environment. After the 3D point locations are discovered, photogrammetry combines all the information to create complete 3D models.

When through the help of photogrammetry, an object model is created, the unwanted parts of it are cleared. After the model is ready, it must be adjusted to fit the registered scene's exact location, rotation, and size. To do this, first, a point cloud has to be obtained through the Kinect that solely represents the object of interest. Obtaining this point cloud can be done by first filtering out as much of the surroundings as feasible, and when only the floor and the object are left, Random Sample Consensus (RANSAC) plane segmentation is applied. RANSAC is an algorithm that can identify the points that make up the floor plane and thus make it possible to separate object points from the points on the floor.

RANSAC is a method that estimates, based on gathered data, the parameters of a specific mathematical model (model hypothesis). In the case of the floor, this is a model hypothesis of a plane, but it can, for example, also be a cylinder, cone, or torus. RANSAC works based on some assumptions, with one being that the data is not perfect and thus consists of both ”inliers” and outliers. With inliers, one means the points that fit the model within a set error margin, while outliers fall outside that range. Another thing assumed by RANSAC is that, based on the inliers, it is possible to optimally estimate the parameters that explain a model. These optimal parameter estimates can accurately segment a plane from a scene. In the case of a plane segmentation in the current situation, the RANSAC algorithm goes as follows [47]:

1. Three random non-collinear unique points are selected

2. The planar model coefficients are calculated based on the three points

3. The distance from a point to the plane is calculated for each point of the point cloud
4. The points that fall within the distance threshold are grouped as inliers; the other points are grouped as outliers
5. The above steps are repeated for a maximum of 1000 iterations to increase the chances of successful segmentation.

When all iterations are completed, the iteration with the most inliers is selected as support for the planar model. All the outliers are regarded as the points that, in our case, make up the segmented object point cloud, as shown in Figure 3.4.

Figure 3.4: RANSAC plane segmentation applied in a real-life setting. The grey points in the point cloud are the inliers, while everything else is an outlier. These grey points represent the identified floor.

When a point cloud of the 3D model is available, and the pose and scale of the segmented object are known, one can start matching the point clouds. Principal Component Analysis (PCA) matches the point clouds by identifying each cloud's mean and primary axes of variation. The eigenvectors that result from these axes are always mutually orthogonal within the point cloud, and because of this, they can be directly used as a rotation matrix. Together with the point in the middle of the cloud, these eigenvectors make up a full transformation matrix for the respective point cloud. When this transformation matrix is applied to the point cloud it is created by; it will move that cloud in such a way that the midpoint is on the origin and the primary variation axes align with the Cartesian coordinate system. Through this transformation, both models can change their basis to the same location with approximately the same pose, making it easier to determine the scaling factor between them. By creating an oriented bounding box (bounding box in the eigenspace) of both clouds, one can determine the variation in each axis, and based on this variation, a scaling factor can be established. After the pose and scale are handled, the 3D model transforms to the position and rotation of the segmented object.

3.5. Object tracking

With the environment and the object ready for tracking, the central part left is the actual tracking of objects. In this project, tracking happens based on a particle filter through which the object's pose is tracked in 6 DoF. Before the tracking process starts, however, first, the object model point cloud used is downsampled. The environment point cloud received from the Kinect will also be downsampled. This downsampling, however, will happen with each newly received frame instead of only once before the start of tracking.

Downsampling is a resampling technique that aids in computational efficiency and filtering out noise. For downsampling, a fixed spatial decomposition technique is used for assembling a local 3D grid of the target point cloud. In the case of this project, it is chosen to make use of a fixed-width octree structure to downsample the point clouds. An octree is a tree data structure where each node that has children has precisely eight. A node without children is called a leaf node. In a 3D space, this means that an octree recursively keeps dividing the space into eight smaller 3D spaces until any further subdivision is undesired. For the application discussed in this thesis, a fixed leaf size width is set to determine how big of an area the leaf nodes should cover. Figure 3.5 shows an example of such a fixed-width 3D voxel grid assembly. Based on this local 3D grid that is created, the centroid of the set of points in each leaf area is determined. The respective centroids are combined to create a downsampled version of the original point cloud.

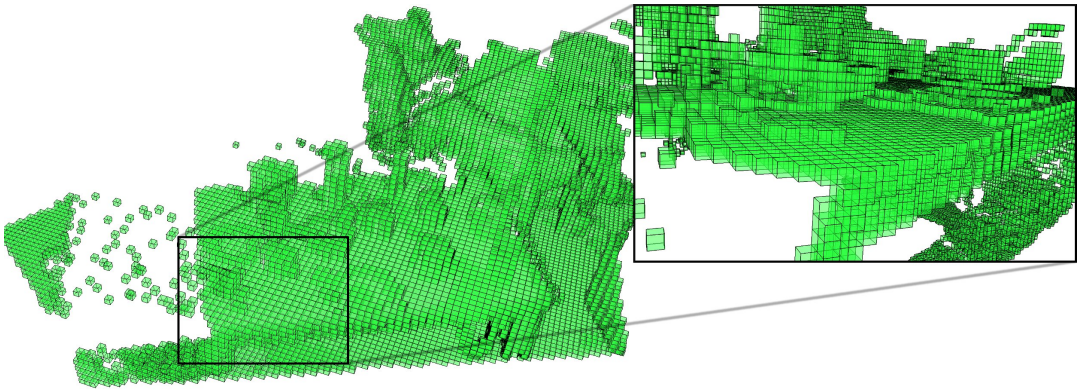


Figure 3.5: An environment represented using an octree structure with a fixed leaf size width of 1.5 cm. [47]

Next to downsampling, each frame received from the Kinect must go through another part of pre-processing before it can be used for tracking. As it turns out, the tracking works better when the floor points are removed from the image on which tracking is performed. Floor removal using the RANSAC approach might be the most accurate option; however, it is costly. Therefore, to speed up this process, a new approach was devised where RANSAC was only used to determine the normal vector of the first received frame. Since the floor is not expected to change orientation, assuming the sensor is stationary, this normal should be able to represent the floor for each frame. A limit can be calculated by projecting the floor's highest point on the normal. This limit serves as the cutoff point; all points below this value are considered part of the floor.

3.5.1. Particle Filter

When the pre-processing through downsampling and floor extraction is done, the actual tracking algorithms can run, which, in this case, is the particle filter algorithm. Particle filters are sequential Monte Carlo algorithms that estimate the posterior distribution of states in a dynamic system, given incomplete and noisy observations provided by sensors. A Monte Carlo algorithm is, in short, an algorithm that repeatedly uses random sampling to obtain an outcome with a certain probability of being incorrect. The particle filter's posterior state estimation (also called belief) is based on the Markov assumption that the posterior state only depends on the current state and is conditionally independent of all other past states. So, particle filters compute the posterior each time by recursively updating the belief based on the latest information. This belief is represented in a particle filter as shown in Equation 3.2.

$$S_t = \{\langle x_t^{(i)}, w_t^{(i)} \rangle | i = 1, ..., n\} \tag{3.2}$$

Here, the belief is a set S_t of n samples, where each x is a particle (or state) at time t with importance weight w. These importance weights sum up to one, forming the belief. This belief, in its turn, is updated through a sampling procedure which, in this case, is called Sequential Importance Sampling with Resampling (SISR) [48, 49]. This procedure consists of two significant steps, namely importance sampling and resampling. However, before the particle filter starts with importance sampling, it needs to (re)sample a set of particles to use. These samples are created by generating random particles from a Gaussian distribution. The mean for this distribution is the position and rotation of the current state. The covariance is a parameter that determines the spread of the Gaussian. The covariance value can be manually tuned and affects the performance, as is later shown in Section 5.

Once the set of particles is defined, the importance weight of each particle is calculated through likelihood estimation. For the associated calculation, several algorithms exist. In this case, a likelihood formula is used based on a combination of the K-Nearest Neighbour (KNN) and Approximate Nearest Neighbour (ANN) algorithms. These algorithms classify the closest point pairs between a point measured by the Kinect and a predicted point. While on itself KNN should be sufficient for the computation, the simpler, less detailed ANN algorithm serves as an extra background check for finding the right solution. KNN is a non-parametric supervised learning method, and the way it works is that it recursively goes through all the points within the indicated range and looks for the k nearest neighbors of the target

point. In turn, the ANN algorithm is less exhaustive and thus considerably faster because it processes the data in a more efficient index, however, at the cost of lower accuracy. For these identified nearest points, the coherence is computed following Equation 3.3, with $C(x_i, u_i)$ the coherence value between the predicted point x_i and the measured nearest point u_i. d is the vector length between the two points, and w is the weight of the coherence approach used for determining its importance. The importance weights sum up the individual particle's complete (non-normalized) weight.

$$C(x_i, u_i) = \frac{1}{1 + d^2 * w} \tag{3.3}$$

After the importance weights are normalized, the particles and their likelihood are used for approximating the posterior state, and consequently, a belief is obtained. When the belief is decided, the particle filter uses its position and rotation data to resample the particles in the next iteration. The described procedure that the particle filter goes through for each iteration is visualized in Figure 3.6.

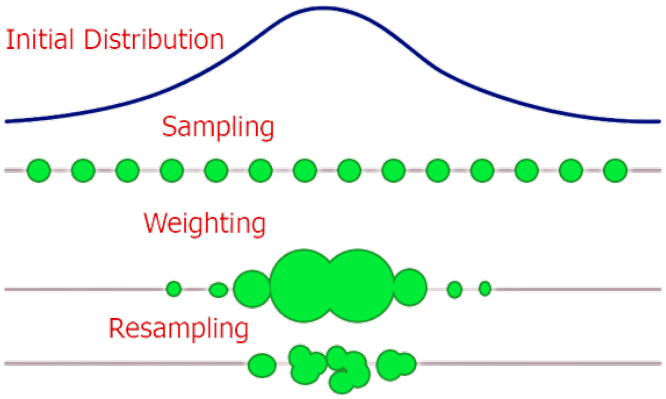


Figure 3.6: Simplified representation of the steps followed at each iteration of the particle filter [50]

3.5.2. KLD-adaptive Particle Filter

A variation of the particle filter is the Kullback-Leibler Distance (KLD)-adaptive particle filter. This variant uses a varying number of particles (samples) for each iteration of the particle filter. The idea behind it is that using the same number of particles for each situation is unnecessary, and reducing this could, with probability $1 - \delta$, reduce the computation time while keeping the distance between the belief and the truth within a pre-specified threshold ϵ. This distance is determined using KLD-sampling since it measures the distance between the belief and the true posterior state by the Kullback-Leibler Distance [51]. KLD is originally a method to measure the resemblance between two probability distributions using Equation 3.4, with p the true posterior distribution and q the estimated posterior distribution.

$$D_{KL}(p, q) = \sum_x p(x) * \log\left(\frac{p(x)}{q(x)}\right) \tag{3.4}$$

The pre-specified threshold consists of a minimum and maximum value for the KLD calculation results. Normally the true posterior is known, which makes implementing the solution relatively easy. However, in the case of the particle filter, the whole point is to estimate this posterior through the filter. Therefore, adjustments must be made to make this approach usable during each iteration of the particle filter. Knowing how the number of samples needed is determined is important. Equation 3.5 shows the derivation of the number of samples n_χ from a discrete distribution with k bins, threshold ϵ, and the upper $1 - \delta$ quantile of the standard normal distribution $z_{1-\delta}$ [52]. For the particle filter, the values for ϵ and δ are parameters to be tweaked manually, and the value for $z_{1-\delta}$ is readily available in statistical

tables. The main challenge for implementation in the particle filter is the determination of the number of bins (k) with support (i.e., containing at least one particle).

$$n\chi = \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3 \tag{3.5}$$

The trick used in the particle filter is estimating the number of supported bins k during sampling. For each new sample, it is checked whether the sample falls into an empty bin, if that is the case, k is incremented with one, and the number of samples required n_χ for the estimated k is updated using Equation 3.5. Since the number of samples n increments faster than the sample threshold n_χ, sampling can eventually be stopped when they are equal. This number is then the number of samples needed for the current iteration of the particle filter. For the current implementation, the bin size and the maximum number of particles can be fixed in the software. Hence the sampling process is guaranteed to terminate.

3.5.3. WA-based Particle Filter

Next to the KLD variation on the particle filter, also another adaptation is created. This variant does not only make use of the previous state while sampling particles for the current iteration. It also uses the result of the previous state as input for the updating sequence of the current state. The original updating happens through normalizing the determined likelihoods of all particles. Then, a new state is identified based on the weighted sum of all parameters of the particles. In the WA-based particle filter, the same approach is used as in the original particle filter. However, the previous state's particle information is now weighed in when updating. The new state is determined based on the weighted average that results from this comparison. This approach increases tracking stability at the cost of a slower response to a changing environment. However, this solution might prove beneficial for the case of MMT given that stability is a higher priority than the actual pose.

3.5.4. Kalman-inspired particle filter

Eventually, a Kalman-inspired particle filter has been worked out, regarded as the next step of the WA-based particle filter. Due to time constraints, it was impossible to finish this variant, but a proof of concept was created in Python. In this approach, a Kalman filter is used in the updating step of the particle filter. The Kalman filter, at this point, produces estimates of the new state. This (optimal) estimate is based on the joint Probability Density Function (PDF) between the predicted state estimates and the measurements, as depicted in Figure 3.7. In the case of the particle filter, all particles of the current time frame are regarded as measurements. On the other hand, the particles of the previous step make up the predicted state estimates, which is possible as a zero-order hold is applied here. Combining these two groups of particles and determining their likelihood, an optimal particle/state estimate is obtained.

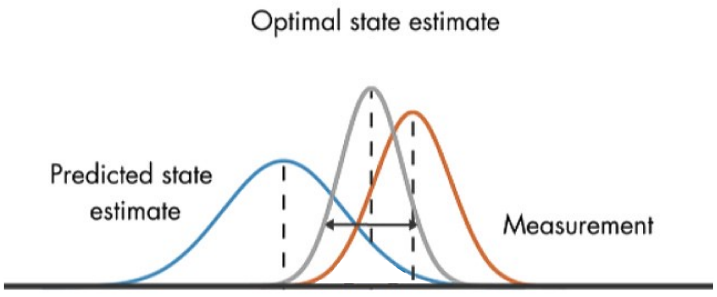


Figure 3.7: The idea behind the Kalman filter for 1 DoF. This same principle is applied in the particle filter, however, now with 6 DoF. Adapted from [53]

The idea of the PDF from the Kalman filter is used to ensure that in states with multiple well-fitting estimates, only the particles within a reasonable distance from the current fit are selected. Selecting these particles is especially useful in the case of rotations, where a symmetric object like a cube can have several poses that fit equally well in the current state. Therefore, using the principles of the

Kalman filter in the particle filter during the approximation of the posterior state is expected to improve performance drastically.

3.5.5. Tracking post-processing for Model-Mediated Teleoperation

Once the particle filter algorithm processes a frame, the pose result is sent to the local model in the master domain. Once the result arrives, the object model is placed in that pose in the local virtual environment. One problem is, however, that the center and set of basis vectors used in the sensor coordinate system differ from the ones used in the environment coordinate system. The eventual tracking result is transformed post-processing to correct this difference before sending the information over the network. A translation matrix describes the sensor distance to the environmental center. This translation matrix is then multiplied with a rotation matrix based on the already defined normal vector of the floor in the environment. The rotation matrix is built up from new axes. The z-axis is defined through the normal vector's projection on the normal vector's yz-plane. This vector is then normalized to make it a unit vector. The normal vector's inverse represents the y-axis; the x-axis is determined through the cross-product of the y and z-axes. Through this transformation matrix, the eventual pose is described from the point of view of the environment's center.

4

Methodology

When starting the thesis, the Embedded and Networked Systems group still needed to gain knowledge about real-time dynamic environment modeling. Hence, we started with the question of how an accurate model of the environment can be created. Soon we realized that this representation had to be done in 3D space, and eventually, two approaches were identified. One of the options was with the use of AR tags, and the other one utilized (colored) point clouds. Since the use of AR tags turned out to be more limiting in its possibilities than the approach with point clouds, we decided to explore the latter direction.

To be able to do research and conduct experiments in the space of environment modeling and object tracking, a decent hardware setup is desired. For this setup, the first and foremost thing we needed was a strong enough computer to run the software that was going to be created. Since, in the beginning, we did not expect to require a high-performance computer, a readily available one would suffice. The particular computer we used from the start, and eventually throughout most of the project, turned out to be an ADLINK MXE-5501. This computer has a 2.80 GHz Intel(R) Core(TM) i7-6820EQ CPU with an Intel(R) HD Graphics 530 graphics card and 32GB of RAM. Next to the computer, we also needed a depth and a color camera for creating the necessary point clouds. Luckily for us, the depth and color camera are regularly combined in what is known as an RGB-D camera. Generally, these cameras are expensive, with, for example, the ZED 2, Azure Kinect, and Realsense cameras all above 150 euros. Since limited monetary resources were available, we looked into a few options and performed a simple Design Space Exploration on them to select which camera to use throughout the project. A Kinect v1 was available at the office, and since its performance was sufficient for our goals in this project, we decided that this was the way to go.

Using Kinect the first step was to find out how to use this camera and whether it was possible to attain data from it reliably. Although Microsoft does not support using this Kinect on Linux, we still require Linux. Linux is a real-time OS, which, compared to Windows, has considerably higher stability and more predictable performance. Eventually, two good open-source libraries were identified with which we could extract the data from the Kinect. One of the libraries explored was based on Python and the other one used C++ as the primary programming language. As shown in Figure 4.1, the Python library lacked performance for our purposes of environment modeling as the point cloud was not accurate enough. Given that the C++ library, called OpenNI2 [54], could extract a decent quality point cloud, it was decided to use that option further.

At this point, it was evident what hardware to use for the sensing and how to obtain a point cloud with depth and RGB data. The next step was determining what we needed to obtain from these sensor measurements. We quickly realized that the parameters that make up the environment and the objects within it are most important to obtain through the camera. Hence, first, we devised a list of all possibly interesting parameters with which environments and objects are described. From this list, we concluded that the first step in environment modeling would be to detect the objects within and then track them while they change. Hence, three stages were identified for environment modeling: discovery, detection, and tracking. Before looking into the stages, we experimented with several libraries and existing projects concerning object detection and tracking. Once we mapped out the possibilities that existed, eventually, we started shifting our attention to solving each stage. Since we wanted to show a

(a) Making use of a Python library

(b) Making use of the C++ library OpenNI2

Figure 4.1: Point cloud extracted with the use of a Kinect v1

working proof of concept at the end, the first thing we did regarding each stage was to check its relative cruciality to our demonstration. Quickly we realized that spending much time on both discovery, as well as detection could be circumvented. Since tracking is more complex than the other two steps, this became the main focus of our research. However, before we could lock in on efforts regarding tracking, the other two stages had to be tackled first, given that tracking cannot occur without them.

4.1. Discovery and detection

Discovery and detection are, in our case, solved by creating a library of objects. In the library, important parameters of the objects and a 3D model of that object are stored and connected. Then, with this information, the object of interest is detected in a small, predetermined area of the environment. While most object parameters can be obtained before tracking starts (weight or shape), acquiring the correct scale and position for an object in the environment is more tricky. Since the object should be able to move in 6DoF, a 3D model should be created containing all sides of the object. Therefore, we set up a pipeline for object modeling involving different approaches. Unfortunately, not all parts of this modeling process are automated completely. A partly manual pipeline was chosen because, after numerous tests, it delivered the best results for object modeling in our specific situation. Since the eventual object tracking performs considerably better with high-quality reference models of the objects, it made the most sense to use this approach.

The pipeline for creating reference models is shown in Figure 4.2 and starts with creating a 3D model. There are only a few options readily available for off-the-shelf 3D model creation. After some experimenting, we found the app called KIRI Engine [46] to be the best free-to-play choice. Through the app, one can create a 3D model saved as a .obj file by making at least 70 2D pictures of the desired object from all sides. In the example of Figure 4.2a, a 3D model of a shoe was created based on the pictures made. Besides using already available apps, other options we considered were writing code ourselves or even readily available RGB-D-based approaches. However, the problem with all these other options was that they either had too low of a time-reward ratio compared to KIRI, lacked detail in the model they created, or all of the above. When a 3D model is correctly built-in KIRI, there are still some undesired areas present in that model which have to be removed before the design is usable. We tried directly removing these noise areas, shown in Figure 4.2a, with our code to create a more automated process. However, due to the randomness of the noise and the limited number of models to make, we decided to deploy a more manual process. In this process, the generated .obj file is loaded into a program called CloudCompare [55]. In CloudCompare, the mesh in the .obj file is converted to a point cloud (.pcd file format) from which all undesired parts of the model can then be removed manually. Eventually, this results in a quality point cloud model as shown in Figure 4.2b.

Before the 3D model could be used as a reference model for the object tracking software, a third step, as shown in Figure 4.2c, was taken. Through this step, object detection is largely avoided since the model's scale and position is adjusted to match that of the object in the environment. Since the application starts with the scaled and positioned 3D model, the discovery is redundant and detection trivial. The software looks at the current situation in the remote environment to achieve scaling and

(a) Model creation (b) Model cleanup (c) Model scaling and positioning - pink is 3D model, grey is object in the environment

Figure 4.2: Pipeline used for creating 3D models of objects

positioning. From the remote environment, it filters out the surroundings from the object of interest. When only the object and the floor are left, it segments the floor plane to decouple the object and the floor in two separate point clouds. Based on the floor normal, the world axes are determined. With these axes and the object point cloud, the program can infer the current object's size and approximate position and rotation. This data consequently scales and positions the previously created 3D model of the respective object accordingly.

To achieve scale synchrony, we took a simple but effective approach of measuring the length of one side of the object and comparing it to that same side in the model. Based on the ratio between these lengths, the size of the 3D model is subsequently adjusted. This approach is possible because the setup of creating the 3D model is static, which makes it possible to always position the model the same way as the object. For the position tuning, two different approaches were considered. One tactic was based on placing multiple markers on the floor. We could use these markers to calculate the floor's normal vector and the object's position in the world coordinates. This information made it comfortable to transform the model to approximately that same location. The other, eventually chosen approach did not use the manually placed markers to obtain a normal vector. Instead, the entire floor point cloud obtained through the earlier plane segmentation was used, providing a more robust solution. The details on how these techniques work are explained in more detail in Section 3.5.

4.2. Tracking

When the 3D models of the objects were adequately prepared, it was time to start looking into one of the main focus points of this thesis: tracking dynamic objects in 6DoF. To find a promising object tracking approach, we again conducted a literature review, this time on currently existing research in the field of 6DoF tracking. What is striking when looking at the set of existing studies in Section 2.4 is that more and more research has started to focus on Deep Neural Network (DNN) as of lately. We compared these approaches with approaches solely utilizing the classical methods and noticed that combining both might be the most promising solution. However, as this is a relatively young field of study, and the research group is relatively new to the topic, our goal is not only to create a good tracking approach but also to gain more insight into the problem. When trying to understand a concept better, a black box model based on DNNs might not be ideal. Hence, we decided that for this project, it would make more sense to focus on using filters and other algorithms that are part of the classical methods to understand object tracking better.

Once more familiarity with the theory behind object tracking was obtained, we could slowly start building the necessary software. The first version of the software consisted of two separate threads, one for the calculations performed and the other for visualizing the results. First, the Kinect captures a colored point cloud based on the depth and RGB data it processes. This data is then fed to the calculation thread, which is eventually the most important as it is the part that enables us to analyze the performance of the tracking algorithms. The calculations, however, do not only encompass tracking efforts; also, pre-processing takes place here. Pre-processing is necessary for two reasons. One reason is that segmenting the floor in the scene improves tracking performance drastically. The other reason is optional when the computational power needs to be stronger. In that case, the point cloud has to be downsampled since using all available points for tracking is computationally too expensive. In Section 5, we show how significant this effect of downsampling is on the timing performance of tracking.

How these pre-processing efforts exactly work is explained in more detail in Section 3.5. After the pre-processing of a frame is concluded, the point clouds of the environment and the 3D model are ready to facilitate the object-tracking algorithms.

For object tracking, several algorithms were considered for implementation. The most promising ones in existing research are the Bayesian-based solutions where a Kalman or particle filter, or a variation thereof, was implemented. To decide which of these two commonly used filters we wanted to test, we looked at the characteristics of both solutions and their variations while considering both the suitability for tracking and MMT applications. While the options do not differ much in the results, one difference between the two approaches is that it has been mentioned multiple times that the particle filter tends to give better results [56, 57, 58]. These better results come at a higher computational cost; however, given the ever-increasing computational capabilities of modern systems and the assumed importance of accuracy of tracking for the MMT system, we decided to explore the particle filter solutions for this project.

In our code, tracking happens by deploying an advanced particle filter that uses both the point cloud of the target object and the environment. By tracking an object, the position and orientation of the object are determined, which can then be used for further usage in MMT applications. Before the eventually advanced particle filter was created, we went through an iterative process of implementing a basic particle filter and slowly attempting to improve it. Three significant steps were identified in a particle filter: prediction, importance sampling, and re-sampling. While newer versions have different techniques for handling one or more of these steps, the overall process remains the same. First, the position and rotation are predicted for a predetermined number of particles/samples based on the information of the particles in the previous frame. After this, the likelihood of those particles being correct is calculated based on a chosen algorithm. This algorithm can be based on several characteristics on which the point clouds are compared, such as distance, normal vectors, or even color between points. Since our setup was placed in an environment where the lights were not so easily controlled, it quickly became apparent during testing that predicting the formula based on color differences was not ideal. Eventually, the use of normal vectors also did not work sufficiently, and thus a likelihood based on the distance between points near each other was chosen as the likelihood predictor. The reason for eventually choosing these parameters was because of the results we obtained backed by the promising indications given about its effectiveness in previous research. When the likelihood is eventually calculated, based on this, re-sampling of the particles takes place. This description briefly explains how the particle filter works at its core, a more detailed picture of all the implemented improvements is painted in Section 3.5.

After we managed to implement the initial particle filter algorithm (Figure 4.3), we noticed that, at first, it was too slow to be used possibly. We printed the execution times to the console to get a rough estimate of exactly how slow the code was. Later on, this approach changed when a better picture of execution times and thread interaction had to be painted. The eventual timing measurement approach used is explained in more detail in Section 4.3. This simple printing approach was used for high-level debugging throughout the code implementation process. It turned out that the point clouds that we used contained many points, and the computer was not the greatest. Hence, we came up with the idea of downsampling both point clouds, as mentioned earlier. This speedup, however, logically came at the cost of having lower tracking stability for the basic particle filter implementation. Besides the downsampling, we also looked at other parameters which could be tuned, such as covariance values and the number of particles used during calculation. The tweaking of these parameters was eventually analyzed for every iteration of our approach. Its results are presented in Section 5. Tuning these parameters for our first effort eventually made a big difference. However, as we gathered more knowledge on particle filters, we identified some opportunities that could improve the performance considerably. Eventually, the idea of having fewer particles when the algorithm is more likely correct led us to try a variation of the particle filter approach called the KLD-adaptive particle filter [52].

The KLD-adaptive particle filter adapts the number of particles it uses for prediction based on the likelihood level that the tracking is correct. Hence, the more confident the tracking algorithm is about a specific position the object is in, the fewer particles it will use for its current calculation. With this approach, the tracking time should drastically improve in simple situations. Since the approach is based on how ”certain” the algorithm is about the actual object location and pose, one would reasonably expect the tracking stability to stay the same while speed improves. Since the KLD adaptation of the algorithm mainly focuses on calculation efficiency, it made sense for our next iteration to look more into how to restrict the position and rotation fluctuations, as this was another concern.

Figure 4.3: Visualization of the particle filter-based tracking algorithm. The blue points on the object represent the actual position of the object that is tracked; the green points serve as a reference to get a better idea of its behavior.

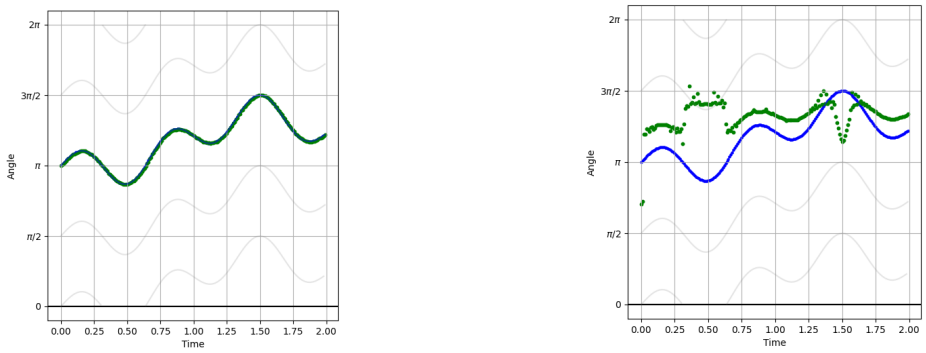
When looking further into how to improve the stability problem of the tracking, we noticed an interesting thing in the theory behind the particle filter. The approach only uses the final pose of the object in the previous frame to generate new particles for the current frame. However, it seemed wasteful only to use some of the information from the previous result throughout the entire process. After all, more data is more knowledge in this case. Therefore, the following improvement applied used the previous result also in the other phases of the particle filter instead of only during prediction. This approach is still a particle filter; however now starts to include more of the principles of a Kalman filter as well, explained in more detail in Section 3.5. In Section 5, one can see an analysis of this implementation compared to the others.

The latest improvement still showed fluctuations in the position and rotation values, although less than before. Especially positioning and rotation over the x-axis seemed to fluctuate too much for haptic systems' liking. Hence, to tackle the problem better, we devised a more drastic Kalman-inspired approach. Since this solution proved to be more complicated to implement than expected due to its mathematical complexity, we decided to test the proof of concept in a different, more easily adjustable test environment. In this test environment, we created an experiment for a 1 DoF rotation, to be expanded to 3 DoF later. Due to the timing constraints of the project, however, we had to leave this possible improvement for what it was, as our priority was to track a digital model of a dynamic object in real-time as part of the MMT system. Since the tracking functioned relatively well, we decided that integrating the software with the MMT system was, for now, of higher precedence.

The experiments on the Kalman-inspired approach proved promising, as seen in Figure 4.4. Here it is clear that, in a 1 DoF situation, the Kalman-based approach tracks the rotational change of a fully symmetric object more precisely than the basic particle filter. This approach functions so well because the PDF that is used in the Kalman filter avoids confusion about the correct degrees of rotation when multiple poses fit equally well. However, keep in mind that so far, these tests only serve to give us a better understanding of its potential, while unfortunately, they could not give a quantitative prediction of the performance within the 6 DoF tracking process itself. Therefore, the main focus of these experiments was to prove that the devised concept is promising regarding tracking at least one DoF. Based on this exploration, the first signs show promising for its implementation in the eventual solution.

4.3. Model-mediated Teleoperation

At a certain point when developing the tracking part, we also had to integrate this into the larger MMT system. Software development was crucial for this system integration, and our existing hardware setup had to be extended. As can be seen in Figure 4.5, the setup of the MMT system exists of a core computer running two peripherals. One of these peripherals is the Kinect that we used for object tracking; the other one is new, the Novint Falcon. The Novint Falcon is the haptic device that will be the controller on the master side of the setup, while the Kinect, as already discussed, is desired for the tracking on the controlled side. Other devices could be chosen as the haptic and sensing devices;



(a) Using a Kalman-inspired particle filter

(b) Without using a Kalman-inspired particle filter

Figure 4.4: Tracking the rotational position of a completely symmetric object over time in 1 DoF

however, these two devices were both readily available at the TU Delft and sufficient for obtaining the defined research goals.

Regarding the computer, the same one was still primarily used as it showed to be sufficiently capable. For one of the experiments, we also used another different computer to evaluate the effect of computing power on the speed performance. For the network of the MMT system, there were two options, either emulate the network on one computer or have a real network between two computers. In the case of this project, it was decided to use an emulated network to ensure that experiments would be repeatable within a controlled environment. Using this setup, either entirely or partially, all desired experiments could be carried out to answer our research questions.

(a) Kinect

(b) Adlink MXE-5501

(c) Novint Falcon

Figure 4.5: Complete hardware setup containing one Kinect v1, one Adlink MXE-5501 and one Novint Falcon

To add the tracking capabilities and other parts, later on, we deemed it essential to have an architecture in place for the application that makes it easy to add or remove capabilities from the system. When starting the project, the idea of what the overarching MMT system would look like was already clear based on existing research in this domain [8, 22]. Since the research group's efforts with regards to MMT had mainly focused on implementing the physics engine, a clear architecture within the software was still missing, however. Since the tracking had to be included, we uncluttered the system and realized a clear division into separate modules, most using individual threads. Besides the standard benefits of the modular programming approach that we applied, such as an easy-to-understand code architecture and reusable modules, it also eased the setup for different testing scenarios later. Modular programming is, therefore, an excellent choice for the design of this project as it makes it relatively simple to add new functionality or leave out existing processes. In addition, it also makes it easier to analyze individual components in isolation or with respect to other modules.

The core component for this system to function correctly is the Unity engine since this engine plays a crucial role in the local model construction. Hence, at first, we created the software so that from the Unity engine, all additional modules were initialized with their respective parameters. There are four main modules (haptic device, physics, network, and tracking), with each their threads. The way the modules in the designed system communicate with each other is shown in Figure 4.6. As one might have guessed, the tracking module contains most of the additions from this research.

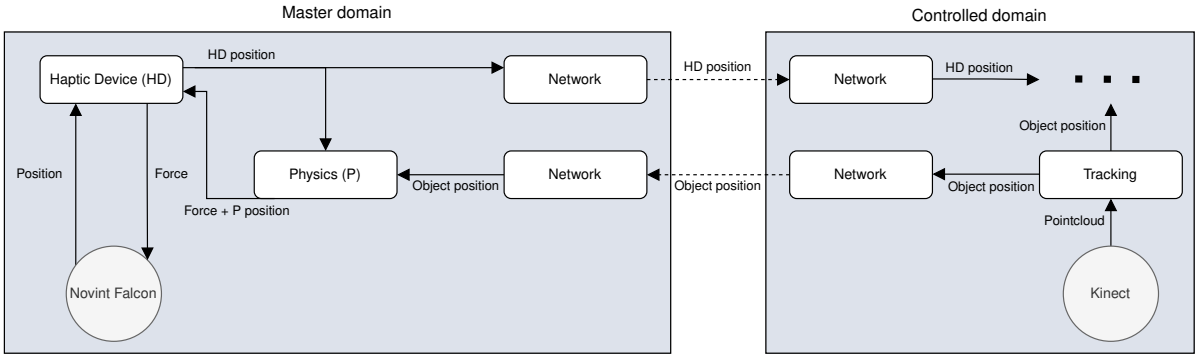


Figure 4.6: Modules of both domains communicating within and between the domains. The Novint Falcon and the Kinect are hardware; however, since they provide data to the software, they are added to the overview. Note that the physics module on the master side takes care of the G' simulation.

Given that there was no desire for this project to have a local model in the controlled domain yet, the module communication regarding this element is also not portrayed (witness the three dots in Figure 4.6). In this system, irrespective of the network, a control loop on the master side eventually always exists, just as MMT intends. Once the entire system is complete, with a local model in the controlled domain, there will also be a control loop specifically for that domain.

In the master domain, the physics module receives interaction data from the haptic device, in our case, the Novint Falcon, and data from the network module. This network module solely takes care of the communication between the two domains, receiving data from one side and directing it to the physics module on the other side. The tracking outcome provides the data that the network transfers from the controlled domain to the master domain, which processes the raw data from the Kinect. Looking at this, we realized that the physics engine was not even needed for tracking in the controlled domain. Hence, we decided to make the tracking functionality standalone and separate it from the Unity engine. Doing this resulted in a speedup of approximately three times for the tracking. When one would want to add a local model on the controlled domain side, there would only be one extra step involved when tracking is standalone. This step would be to connect the physics module in the controlled domain through different network ports than the ports the tracking part uses for connection to the master domain. The network modules for both would also be slightly different as tracking data only has to be sent, while the physics data has to be both sent and received.

The network module can have two different components, one for sending and one for receiving. When data is being received or sent, it has to be done asynchronously, meaning a specialized thread is needed for both operations. The packets sent over the network are UDP packets since the application operates in real time. With this approach, security risks arise for some applications. However, since this research focuses not on networking, it was decided that a UDP protocol was sufficient. One thing we did to make the connection more secure for the future was about packets critical for proper execution. For these packets, the receiver is expected to return an acknowledgment with the hash of the contents when it receives a packet correctly. The sender, in its turn, waits for the acknowledgment for a while; if it does not receive it, the sender will resend the critical packet. The reason this bilateral UDP packet implementation is done for critical packets becomes clear when we take a look at the purpose the network serves. Communication over the network is essential for two reasons: discovering new objects and maintaining the already established objects. While both packets are sent over the network, the maintenance of object information is much less critical than when a new object is discovered. This difference in criticality is because when object discovery takes place, the corresponding packet is only sent once, while for object updating, the data is received constantly. Missing a discovery packet might thus have significant consequences compared to the minor effect that missing a maintenance packet has.

Next to the communication between the domains, the communication within the domains must happen safely. Because of the multi-threaded nature of the code, it is vital to ensure the safe functioning of these interdependent threads. This thread safety for read and write operations between the threads is guaranteed through mutexes. Mutexes ensure that only one thread can interact with a particular part of the code; if a second thread wants to interact, its execution is halted until the other thread is done.

Even though in C++, atomic variables are superior to mutexes concerning performance, mutexes are crucial when writing to multiple addresses in a thread-safe manner. If, for example, one thread would read a position or force vector while another thread is actively writing new values to that vector, a partly updated vector could be read. This erroneous reading could crash the system or diverge the model depending on the situation. These problems would be caused by misplaced objects or wrongly determined force feedback.

4.4. Logging and benchmarking

The results required for evaluation were logged into a CSV file which contained all the essential pose and timing data. While the pose data was logged the moment the result of the iteration was known, the timing measurements were gathered using a benchmarking tool developed by The Cherno [59]. This benchmarking code made it possible to measure the duration of each tracking iteration and even its sub-components. Not only did this benchmarking code help us measure the timing performance, but due to its visualization capabilities, it eventually also proved helpful for even further improving the timing performance. By visualizing the function execution timeline and the used threads, bottlenecks and simple adjustments could easily be identified. Due to this tool, we realized that when using tracking in the MMT system, we could spread the work for tracking over two threads: one occupied with retrieving and pre-processing, the other one with the tracking computations based on the earlier pre-processed frame. By implementing the tracking functionality this way, we achieved a higher frequency.

5 Results

In this research, the focus is mainly on the performance of the tracking module and its integration within the larger MMT system, as shown earlier in Figure 4.6. Several experiments are conducted to give more insights into the conditions discussed earlier. The main object used in the experiments was a rectangular box of 21 by 9 by 6 cm. This box was placed in a controlled environment that was easy to model for the tests, without any closeby objects.

5.1. Tracking performance experimental setup

For the analysis of the tracking performance, three different scenarios are identified. One situation is about an object that does not move and is hence static. The other two scenarios happen when an object changes its position or rotation. Three point cloud videos are recorded to test these scenarios. In one video, there is a static rectangular box; in one video, the box moves by ~$25\,\mathrm{cm}$ in the x direction; in the third video, the box rotates around its center with ~$25\,\mathrm{degrees}$. With these videos, it is possible to conduct experiments on tracking performance in identical situations using different parameter values and versions of the algorithm.

Several parameters are analyzed to find the optimal configuration for the particle filter. While several parameters have been experimented with, three of the most interesting ones are identified and discussed in this section; these are:

- Number of particles
  The number of particles being used in each iteration of the particle filter on paper should be one of the parameters with tremendous influence on the speed of computation. For the solution version utilizing KLD sampling, this number indicates the maximum amount of particles to be used rather than the exact number.

- Level of downsampling
  The level of downsampling applied to the point clouds used for tracking. These are the environment point cloud and the 3D object model point cloud. Instead of looking at the number of particles directly, the level of downsampling influences the number of points used to calculate a particle's fit to the model.

- Sampling covariance
  Sapling covariance is the spread of the area in which particles are randomly sampled in the sampling phase of the particle filter.

Unless otherwise indicated, the parameter values used in the experiments are the same for each experiment. In the case of the three parameters mentioned before, this means that downsampling is at a $20\,\mathrm{mm}$ level, sampling happens with a covariance of $15\,\mathrm{mm}$, and a total of $2000\,\mathrm{particles}$ are used. Besides analyzing the particle filter parameters, different particle filters are also evaluated on their performance. In total, four different particle filters are closely examined:

- Basic

- KLD-adaptive
- WA-based
- Combined KLD-adaptive and WA-based

These different filters and several parameter configurations are evaluated based on timing performance. For timing, a distinction is made between processing/computation speed and filtering delay. Computation speed in this thesis is defined as the time it takes for a new frame from the Kinect to be received and processed. This processing time can be further divided into three parts: downsampling, extraction, and tracking. Of these three parts, the first two are part of the preprocessing that each frame has to go through, which is explained in more detail in Section 3.5.1. Filtering delay describes how far the filter is lagging behind environmental changes. Since no means were available to measure the movements and rotations to mm and degree level accuracy, the filtering delay uses relative changes for comparison. Hence, the performance is evaluated for filtering delay relative to other filter implementations and configurations.

Besides timing, another vital aspect of the particle filter is its tracking accuracy. Because of the inability to measure movements and rotations to mm and degree level accuracy, a relative approach is used rather than an absolute one. In the static scenario, with the box standing still, the positional accuracy is defined as the Euclidian distance between the current and the previous point. The rotational error is determined as the change in degrees between the current and the previous point. This error determination is done separately for roll, pitch, and yaw since no single measure exists to define rotation in three dimensions. The current and previous points are used each time because, in a static situation, the object is not supposed to move; any change concerning the previous pose can be regarded as a tracking error. Also, an important motivation for this choice is that, for TI applications, it is first and foremost important for the QoE that objects that do not move also feel like it. Hence, it is important to quantify how much change occurs between frames, as this change is supposed to be zero.

5.2. Tracking performance analysis

5.2.1. Particle filter parameters

The effects of different parameter configurations on the timing and accuracy performance of the fundamental particle filter are shown in Figures 5.1 to 5.10. Figures 5.1, 5.4, and 5.8 represent the static tracking scenario where the accuracy evaluation is based on the positional tracking error. The rotational tracking error is not shown here, as those results follow the same trends.

In Figure 5.1, the processing time increases somewhat linearly with the number of particles used by the filter. Fluctuations in the positional tracking error, on the other hand, decrease drastically when only small numbers of particles are used. When a certain number of particles is reached, however, this decrease in tracking error starts to smoothen out. Hence, increasingly using more particles tends to cost more time at a certain point, while there is only a limited decline in the tracking error. This situation is essential to remember to benefit the most from the trade-off between processing time and accuracy.
Inference 1: A range of particles can be identified in which a significant amount of processing time can be gained with marginal loss in tracking error.

In Figure 5.1, it seems to be that after at least 1000 particles are used, the tracking error does not decrease much anymore. In a static situation, a value of around 1000 particles would seem appropriate. However, when looking at the filter's behavior in a situation with positional displacement, Figure 5.2, a slight filtering delay starts to appear when 1000 particles are used. This delay is even more apparent with 500 particles as in both Figures 5.2 and 5.3, rotational and positional changes are delayed critically.
Inference 2: The velocity with which an object changes its pose affects the number of particles needed for the tracking algorithm to track that object properly.

Figure 5.4 clearly shows the effect that the level of point cloud downsampling has on particle filtering. As expected, with a lower downsampling range, the tracking error decreases, and the time needed increases. As shown in Figure 5.5, this exponential increase in time with lower levels of downsampling directly correlates with the number of points left in the point cloud after downsampling. Hence, one should try to balance the timing and accuracy for the downsampling level to ensure the particle filter's proper functioning.
Inference 3: The gain in processing time is directly correlated to the number of points used in the particle filter calculations.

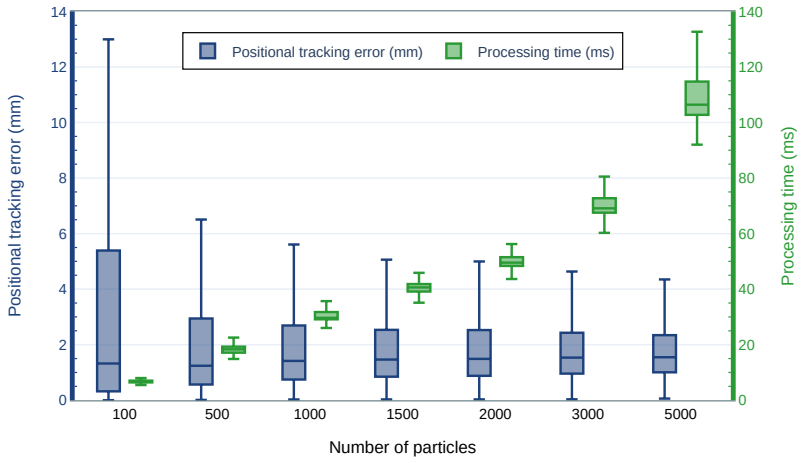


Figure 5.1: The effect that the number of particles has on the positional accuracy and processing time of the particle filter. Important to note is that the difference in the number of particles is not evenly spaced over the entire x-axis.

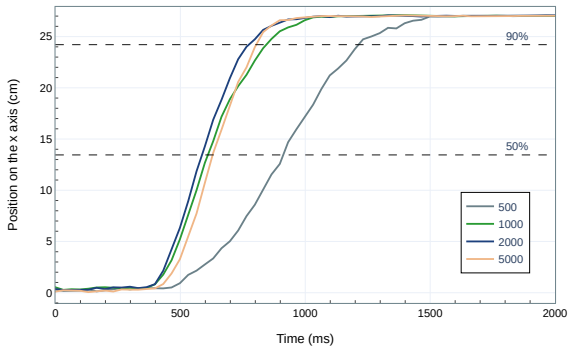


Figure 5.2: The effect that the number of particles has on the positional filtering delay in the x direction

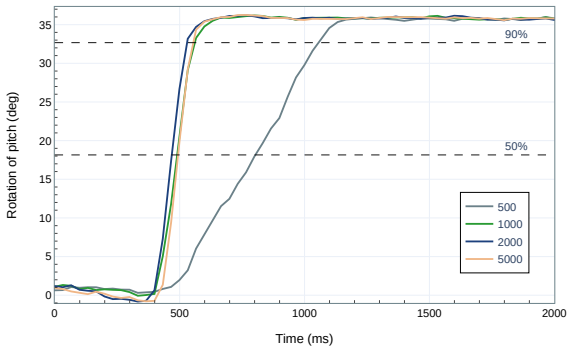


Figure 5.3: The effect that the number of particles has on the rotational filtering delay in the pitch rotation

Looking at Figure 5.4, one might think that the best option, in this case, is to choose $30\,\mathrm{mm}$ of downsampling as compared to $20\,\mathrm{mm}$ the accuracy is barely affected while the processing time almost halves. In a static scenario, this might be true; however, as Figures 5.6 and 5.7 show, this does not hold in dynamic situations. While the position is still sufficiently kept track of at $30\,\mathrm{mm}$, it loses track of the rotation altogether.

Inference 4: There is a minimum number of points needed for the tracking algorithm to keep track of an object's pose.

Figure 5.8 shows that the sampling covariance leaves the processing time nearly unaffected. For accuracy, however, an apparent effect can be observed as it tends to get more stable and accurate when the covariance is smaller. This effect is only observed up until a certain point where a lower covariance becomes unusable. For static scenarios, this is not immediately apparent. However, when an object moves, as seen in Figures 5.9 and 5.10, it shows that the lower the covariance, the smaller the changes in the object that the algorithm can track. At a certain level, the particle filter cannot sample particles close enough to the correct location to adjust its prediction. Hence, even the slightest change becomes too difficult to track. The question to be asked for deciding the correct value is, hence, the maximum level of change in the object pose per time frame.

Inference 5: The change in processing time is negligibly small for a change in sampling covariance but a lower level aids with better accuracy.

Inference 6: A low covariance can cause the tracking algorithm to not keep up with objects moving above a certain velocity.

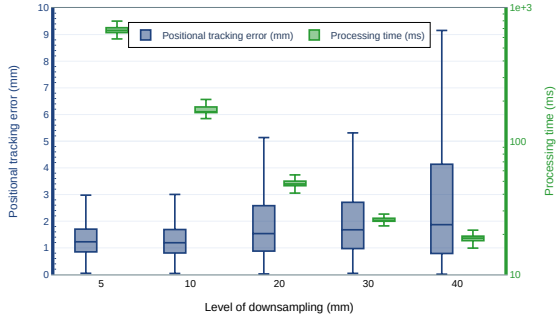


Figure 5.4: The effect of tuning the level of point cloud downsampling on the positional accuracy and processing time of the particle filter. Note that the y-axis on the right about timing is shown with a logarithmic scale.

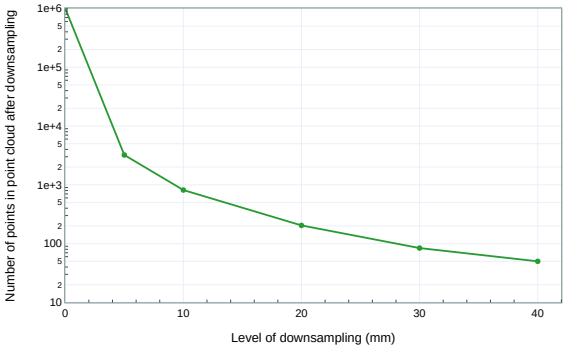


Figure 5.5: The number of points left in the object point cloud of the box used in the experiments after downsampling.

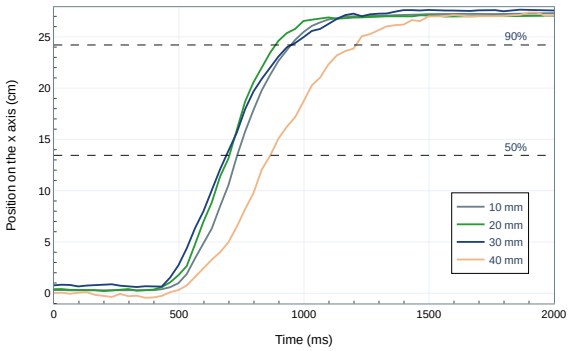


Figure 5.6: The effect that the level of downsampling has on the positional filtering delay in the x direction.

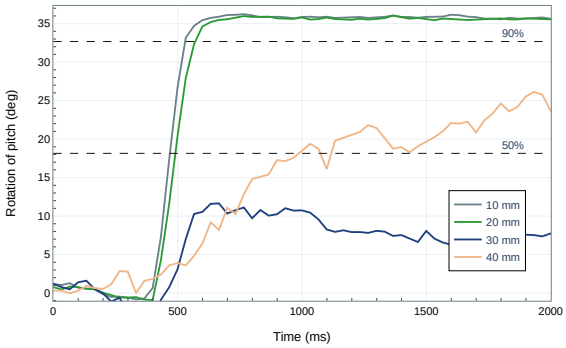


Figure 5.7: The effect that the level of downsampling has on the positional filtering delay in the pitch rotation.

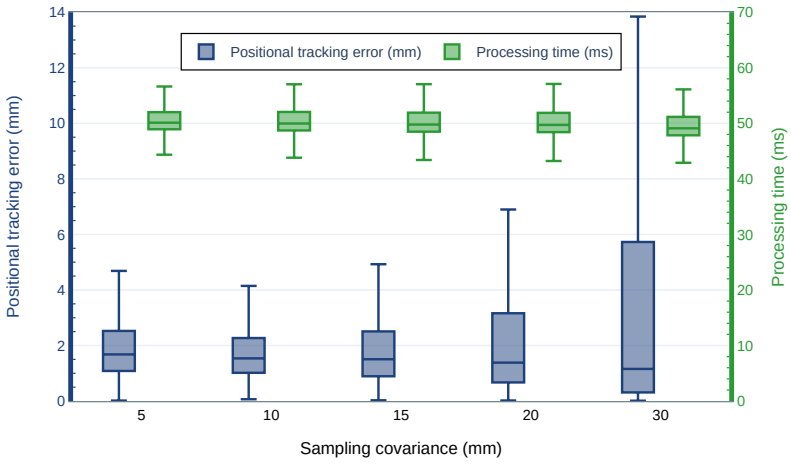


Figure 5.8: The effect of tuning the sampling covariance on the positional accuracy and processing time of the particle filter

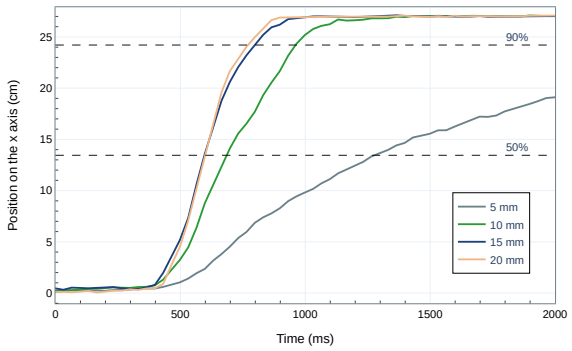


Figure 5.9: The effect of the sampling covariance on the positional filtering delay in the x direction.

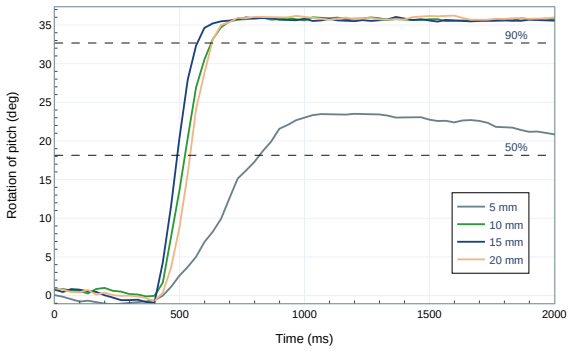


Figure 5.10: The effect of the sampling covariance on the positional filtering delay in the pitch rotation.

5.2.2. Particle filters

From the analysis of the three parameters above in both static and dynamic scenarios, it becomes clear that there is a trade-off between how well one wants to be able to track changes in a dynamic object versus how much error one wants to allow in situations where the object is static. Different implementations of the particle filter also affect the timing and accuracy performance. This analysis is shown in Figures 5.11 to 5.15. Figures 5.11 and 5.12 evaluate the static performance in both the timing and accuracy of each version. All other parts below focus on the timing analysis amongst all scenarios. For the particle filter without KLD, the standard number of 2000 particles is used, while for the particle filter with KLD, 2000 is the maximum number of particles used. For KLD, that means the number of particles could be less if the algorithm decides so.

The performance results for different particle filter implementations in a static scenario can be seen in Figures 5.11 and 5.12. In Figure 5.11, a time-frequency domain transformation is performed with the help of the Fast Fourier Transform (FFT). Since the tracking error is not constant, it means it consists of different levels of noise that are uncluttered this way. Since there should be no frequency in a static situation, anything detected is basically due to an error. Therefore, the bigger the amplitudes of a version, the bigger the power of the noise. The frequency domain of Figure 5.11 shows that WA outperforms the other implementations, closely followed by WA + KLD. The basic and KLD implementations seem comparable in performance. Figure 5.12 supports the observation for WA to be the best-performing solution in a static environment. WA also shows here that, both with and without KLD, it decreases the error rate while not affecting the processing time. Something interesting to note here is that KLD implementations take longer in their processing time, while the core idea of KLD is to diminish the time needed in situations like this.

Inference 7: The WA implementation of the particle filter significantly reduces the tracking error over the entire frequency domain while leaving the processing time unaffected.

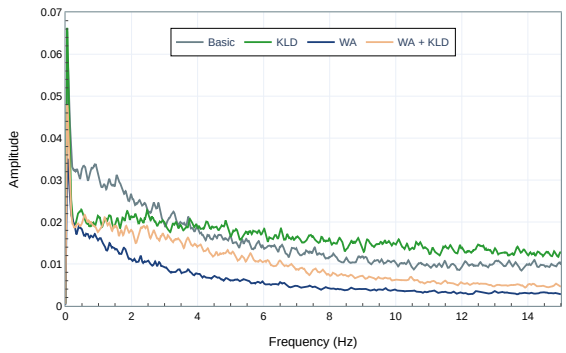


Figure 5.11: Low-pass filtered frequency domain for positional tracking, comparing different iterations of the particle filter

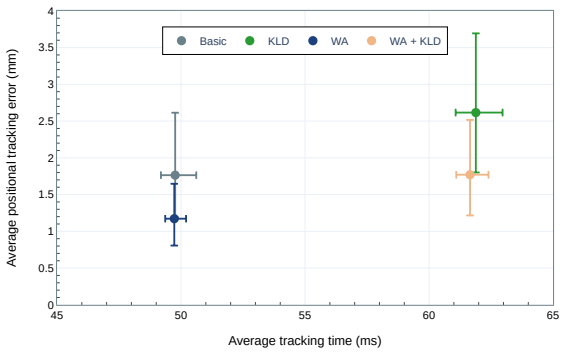
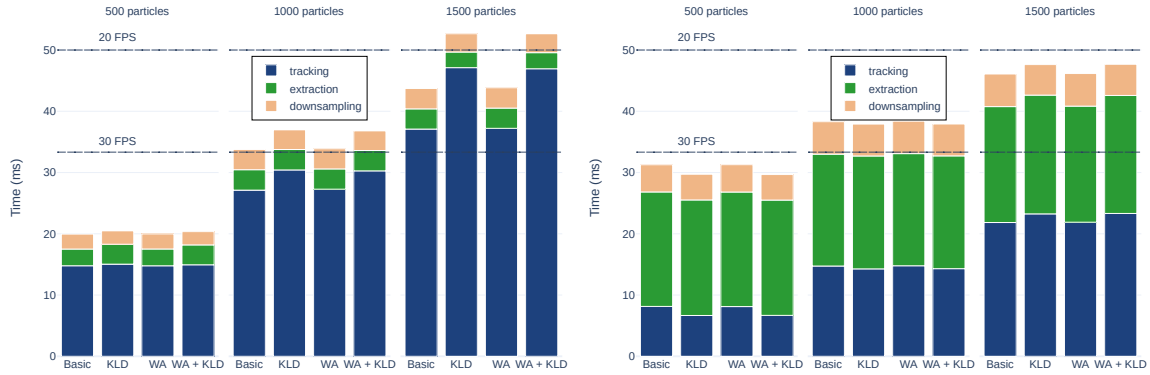


Figure 5.12: The average positional tracking error balanced against the average tracking time. The whiskers show the 25th and 75th percentile around the median.

To ensure tracking is not the limiting factor in the system, it has to perform at a higher Frames per Second (FPS) than the speed with which the Kinect receives new frames. Since the Kinect functions at 30 FPS, ideally, the tracking algorithm runs at equal or lower FPS. Figure 5.13a shows that reaching 30 FPS with the current setup is attainable when 1000 particles are used. When more than 1000 particles are used, the processing time becomes higher than 30 FPS. When this same application is run on a more powerful computer, as shown in Figure 5.13b, the tracking efforts can use even more than 1500 particles.

Inference 8: The processing time for tracking depends on the computing capabilities of the system significantly.



(a) Processing times on the original computer

(b) Processing times on a different computer. This computer had trouble dealing with some parts of the Point Cloud Library for unknown reasons, leading to long extraction times. The most important in this graph is to look at the times given for tracking.

Figure 5.13: Representation of the processing times for different numbers of particles. Since the tracking runs on a separate thread from extraction and downsampling, the frequency is usually unaffected by the extraction and downsampling. These two preprocessing efforts hence only affect the processing time of a frame.

While WA outperforms the other implementations based on processing time and accuracy in static domains, it shows a lag in terms of filtering delay. In Figure 5.14, the filtering delay shows to be relatively small during the object's movement ($50\,\%$ point), where some variations of the filter have more delay than others. The moment an object stops moving, the transition from dynamic to static takes place (around the $90\,\%$ point). These results show increased differences in delay between different particle filters as some adapt faster to changes than others. Not only does this happen for positional changes, Figure 5.15 shows this is also the case for changes in rotation.

Inference 9: A higher accuracy for tracking static objects comes at the cost of a more significant filtering delay when keeping track of the dynamic object's changes in pose.

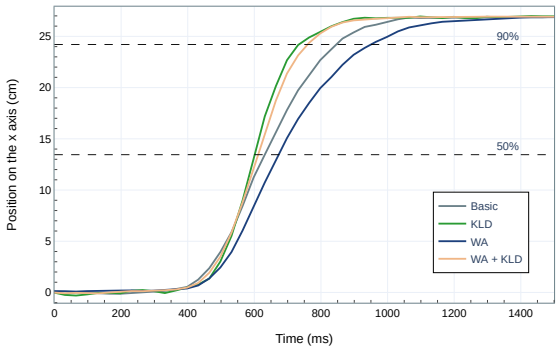


Figure 5.14: Effect that the different particle filter implementations have on the positional filtering delay in the x direction.

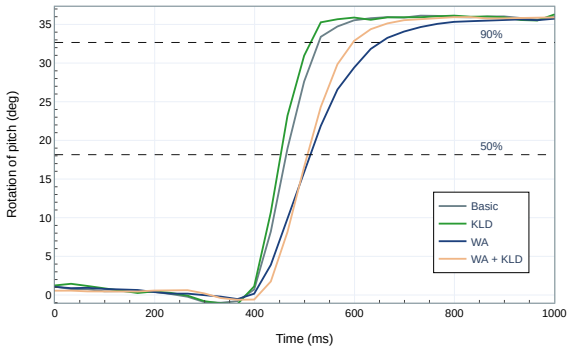


Figure 5.15: Effect that the different particle filter implementations have on the rotational filtering delay in the pitch rotation.

Combining each filter's processing time and filtering delay shows that both are essential measures to consider when looking at timing performance. Figure 5.16 shows an apparent decrease in time for positional tracking of objects when a KLD filter is used. This effect of KLD is amplified if the filter works

based on WA. In Figure 5.17, the same effects are observed when rotation is tracked. Here, however, the effect of KLD is negligible and less effective in decreasing the delay that WA introduces to the tracking module.

Inference 10: Overall, KLD significantly decreases the time the application needs to keep track of changes in an object, especially in the positional domain.

Inference 11: WA significantly increases the time it takes for the application to respond to an object's dynamic changes, especially in the rotational domain.

Figure 5.16: Timing of the position tracking process (1500 particles used)

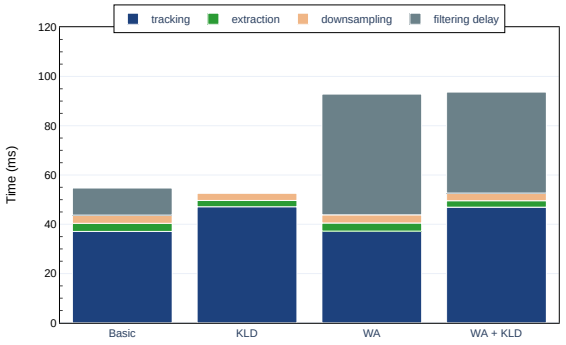


Figure 5.17: Timing of the rotation tracking process (1500 particles used)

5.3. Model-Mediated Teleoperation experimental setup

Minor user tests are conducted to analyze the tracking performance concerning the MMT system. With these tests, a first impression can be obtained of the QoE of the tracking performance at different levels of haptic feedback sensitivity. Through these tests, better insight is gathered into how good the tracking has to be before people experience limits in operation. To test the entire system, next to the computer and the Kinect, also the Novint Falcon is needed. For the evaluation of the QoE of the system, we combined the standalone tracking software with the physics engine running the local model of the environment. The parameter values used in the experiments are the same for each experiment. The only difference with the previous experiments is that now all tests are performed with 1500 particles instead of 2000.

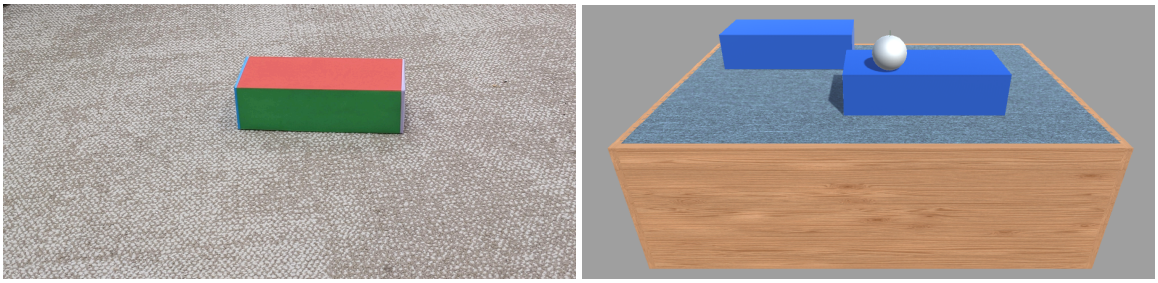
For the analysis of the system performance, two different tests are identified. One experiment evaluates the positional tracking error, and another evaluates the rotational tracking error. Both tests are conducted while tracking a static rectangular box of 21 by 9 by 6 cm. On this static box, the QoE at the center of the object is used for evaluating the positional tracking error. The QoE at the wide ends of the rectangle are used for rotational error assessment. No dynamic object interactions are evaluated because only one Novint Falcon is available at the time of writing. For the evaluation of dynamic interaction, the master domain should be able to interact with the controlled domain as well. With the current setup, only the opposite interaction direction works.

In the user study, three participants, aged between 25 and 35, were invited to Delft University of Technology. None of the participants had any known neurological disorders. Three requirements are adhered to gather statistically relevant test results [1]:

- A task should be performed several times under identical circumstances
- Participants of all skill levels should be able to concentrate equally well
- To prevent fatigue, the duration of the experiments should be brief

An experiment is created where participants are presented with sixteen static scenarios. However, each scenario has the same setup with different combinations of particle filter implementations and spring constants (representing the system's sensitivity as explained in 3.2). The spring constants used in the experiments are the same as the ones used in research by Kroep et al. [60]. The participant has to sit in front of the computer while controlling the Novint Falcon. In each scenario, the participant

has to move the Novint Falcon around in the virtual environment and rate the QoE of the tracking error. The virtual environment shows a platform representing the floor in real life, with two equally sized rectangular boxes on top. One of these boxes is static, used as ground truth, and only present in the virtual environment. The other box moves according to the tracking results received. The real-life setup and the virtual environment are shown in Figure 5.18 to get an idea of the experiment.



(a) Real-life environment (b) Virtual environment

Figure 5.18: Real-life and virtual environment used in the experiments

5.4. Model-Mediated Teleoperation performance analysis

Figure 5.19 and 5.20 both show the subjective performance of the object tracking of an object, one position-wise and one concerning rotation. A grade of 8 or higher is interpreted as excellent performance, while everything below 5 is considered insufficient. In both situations, the particle filters without KLD are preferred over the ones with KLD. This preference partly matches our expectations as the quantitative analysis shows that KLD sacrifices a small amount of accuracy during static situations in a trade-off for faster response times in dynamic situations. Essential to note once more is that the experiments with the complete system are regarding a situation where a static object is tracked since experiments with dynamic objects require a more advanced system. Another critical observation based on the user tests is that the acceptable accuracy of the tracking module depends on how sensitive the haptic feedback is. Sensitive systems take note of small changes relatively quickly compared to scenarios where weaker spring constants are used.

Inference 12: KLD-based approaches generally create a lower QoE in static scenarios while WA-based implementations slightly improve it.

Inference 13: A lower haptic sensitivity makes for a more stable experience. Hence, the lower the spring constant, the more tracking error is acceptable.

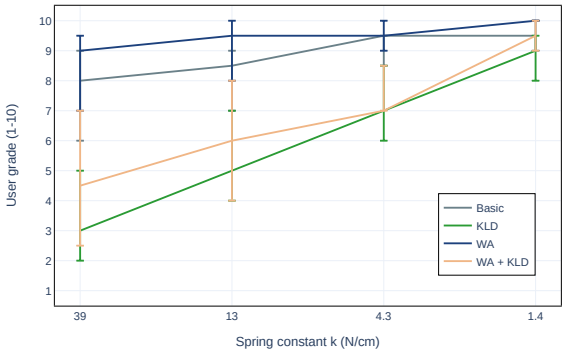


Figure 5.19: User grades rating the positional tracking performance of several particle filter implementations across different haptic sensitivity levels. These levels are expressed by the spring constant.

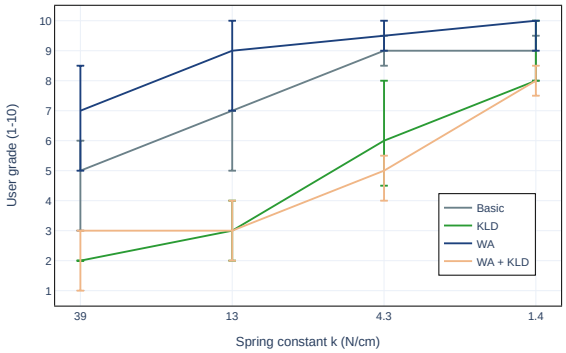


Figure 5.20: User grades rating the rotational tracking performance of several particle filter implementations across different haptic sensitivity levels. These levels are expressed by the spring constant.

All participants noted during tests that with the lowest spring constant of 1.4, the experience overall was slightly confusing since the feedback was too weak to even register distinctions between, for example, the floor and the boxes. Also, a surprise for them was the amount of jitter observable on screen

compared to how well one could feel this movement through the haptic device.
Inference 14: If the haptic sensitivity becomes too low, the experience is not regarded as pleasant anymore as it becomes difficult to distinguish between objects.

6

Discussion

We have progressed towards a more robust TI with the current work. Throughout the process, many things went well, and others, not so much. Overall we made much progress reasonably quickly. As the funding was limited, we had to get creative in obtaining things. For example, when designing the 3D models, we had no accurate way to create them. We dealt with this challenge appropriately through an existing app called Kiri Engine [46] and some creative tinkering. Also, the computer used throughout the process was barely good enough to run everything correctly. The lack of computing power available, however, can also be seen as positive as it forced us to be as efficient as possible when designing the software.

Regarding the tracking algorithm development, perhaps other approaches should have been tested instead of zooming in on the particle filter and using this as the basis for all improvements. Kalman filter tracking and many DNN approaches currently exist that try to tackle the same problem. Since these approaches exist, the comparison between ours and the others is missing in the thesis. The missing comparison, however, is because there is no generic testing data set and no standards available to compare 6 DoF tracking algorithms. Moreover, even if there were, they would not have been focused on using tracking algorithms for TI applications. Due to the limited time available, it was decided to compare the performance only to our proposed solutions.

For the analysis of the results, substantial quantitative experiments were conducted, which helped get a good indication of the effect of our research. A controlled environment was used that was identical in each test when needed. However, in these experiments, we always used relative ground truths rather than absolute ones since there were no easy ways to obtain reliable measures of this. However, the relative ground truths we used have solved our problems of the missing ground truth quite proficiently. Another criticism that one could make of our qualitative research is that the overall positional tracking error was investigated while not mentioning anything about the errors over the axes separately. Perhaps when this has been zoomed in on, different inferences could be obtained about the tracking performance if one axis performed worse than others. Besides the generalization of positional tracking, also the generalization of our performance based on experimentation with only one object could be questioned. Different objects could indeed lead to different details in the results. However, it is expected that the same general principles hold across different objects.

The subjective analysis was rushed due to bad timing and should have been done more extensively. While the current qualitative results can provide reasonable indications of what to expect, significant conclusions cannot be drawn due to the small number of participants present. However, the experiments conducted with the participants were correctly executed without any anomalies.

6.1. Future work

In this thesis, an MMT system that incorporates tracking is successfully created; however, as mentioned in Section 6, there is still much room for improvement and, thus, future work. The easiest improvement would be to conduct more extensive user studies on the static experience of the tracking algorithm. Instead of only a few participants, the group should be extended to gather statistically more substantial results. This way, inferences can be made that are statistically significant. Next to more extensive

existing user studies, extending the user studies to dynamic object QoE evaluation is an exciting step. From the results of the user studies, promising indications can already be obtained about the possible tracking and sensitivity requirements of the MMT system. When user studies on dynamic object tracking are performed, results can be combined with the outcomes from a study of Kroep et al. [7], which could, for example, give us an idea of the minimal tracking delay requirements for smooth operation. However, it could also be inferred whether the tracking capabilities are a problem for the system or whether network latency is still the bottleneck. However, a more complete MMT system is needed to conduct these dynamic QoE experiments. Currently, the system can only provide feedback from the controlled domain to the master, while for proper dynamic object interaction, feedback from the master domain to the controlled domain is also required. Once that is done, one can also consider testing the setup with a real network instead of an emulated version to see if unexpected problems arise.

Besides the next step of dynamic object experiments, one could also consider first doing more extensive tests while using different types of objects. Different types of objects might require different tracking algorithm compositions. Big objects could, for example, be easier to track with fewer points, or objects with lots of odd shapes might require more. Nevertheless, before more objects are tested, improving the object modeling pipeline is a higher priority mainly because higher quality 3D models of objects make it easier for any tracking algorithm to track that object.

While improving the quality of the 3D models is essential, the quality of other parts of this setup can also be improved. Cameras with a higher frame rate capability or different types of sensors could be used. However, more powerful computers can also be deployed to boost the system's performance. When a powerful computer is used, the tracking algorithm can be further multithreaded on a GPU, whereas up to now, we only ran it on a CPU with a limited multithreaded implementation. Instead of improving the hardware, the software behind the tracking algorithm could also be improved. The first step would be to implement the proposed Kalman-based particle filter.

Nevertheless, other ways should be identified to improve the tracking performance besides implementing the Kalman-based particle filter. Perhaps more complicated machine learning algorithms could be experimented with, and DNNs could be used. As we have seen in several studies, this shows promising results for the future [40, 41, 42, 38, 39]. To be able to do this, however, general guidelines should be constructed that makes it possible to compare different tracking algorithms to each other. Currently, there is not much available or even one guideline for evaluating object tracking performance in the TI domain.

Another direction one could take in the future instead of improving the performance and extending the tests of the tracking functionality is expanding its capabilities. Eventually, the system would have to consider an entire dynamic environment. Multiple objects will then move around; for proper operation, all moving objects must be tracked. Hence, tracking could be extended to track multiple objects at once. These objects in a dynamic environment do not necessarily have to be there from the beginning; an environment can change. Hence, the system could be extended to be more flexible in real time. One could develop a way to recognize new objects that enter the scene. These objects may need to be discovered by the system. Another addition is making the system model new objects it does not recognize on the spot. Currently, the system is already aware of which object it is tracking and where its starting position is; hence these two improvements have not been required. One more limiting factor of the current system is that the camera is static while the environment is dynamic. A more immersive experience is expected to emerge if one can move the camera around.

Of course, other parts of the MMT system can be improved or added besides improving how real-time dynamic environments are captured. No local model, C', is currently available in the controlled domain representing the controller's behavior in the master domain. This part is crucial to creating a fully functional MMT system. One could also focus on the interaction in the local models. In the local models, the data from their domain and that from the other domain must be used together. This model correction scheme must consider the network latency when the local model is corrected. The effects of this latency have to be researched, and methods have to be identified with which model parameter values can be derived from delayed information received over the network. The interplay between local and remote data is one of the crucial questions in MMT as it is about how the system deals with model mismatch. While current research in MMT has proposed solutions, these are based on static environments. Hence, these ideas must be reevaluated for use in dynamic scenarios. As stated in Section 1, dynamic situations require more parameters to be taken into account, and the static solutions so far will probably lack at least partially. Thus, the importance of newly identified parameters has to

be assessed for different scenarios.

6.2. Conclusion

Through this thesis, we are a few steps closer to understanding how TI can be used globally. In the future, TI will be able to enable interaction with other people in ways not experienced before. The reason is that besides remote visual and auditory interaction, TI also makes it possible to experience remote kinesthetic interaction. The network latency constraints that haptics deal with form the biggest obstacle to designing usable TI applications. In order to solve these latency constraints, the majority of research focuses on reducing the latency produced. While reducing the latency helps partly, it will still be limited by the speed of light.

In this thesis, the focus is not on reducing the latency but rather on relaxing its constraints. Therefore, an MMT system that uses local models of the actual environments for interaction is developed. Because these local models are less dependent on data from the network, the network latency requirements can be loosened. A local model, however, will never be perfect when the environment is dynamic. Therefore, feedback from the remote dynamic environment is still needed, and registering this is currently regarded as one of the critical challenges for long-distance TI.

With our research, we take a deep dive into how to capture real-time dynamic environments for use in MMT. We identify and demonstrate accurate tracking of objects in the environment as the critical building block to enable MMT with dynamic objects. For this, we develop the first 6 DoF tracking application designed explicitly for TI applications. This tracking application uses an improved particle filter-based tracking algorithm to gather data about the remote environment. This data can then be used to correct for mismatches of the local model in the master domain of the MMT system. To establish this as well as possible, we conduct several experiments that help identify the performance trade-offs required when tracking objects, both in the static and dynamic situations with which MMT deals. While our approach is in no way solving all of the issues that TI is currently facing, it does help move the developments of a small subset of the challenge slightly forward. As the community involved with TI research is growing, the advancements to enable the TI revolution are getting more promising every day.

Bibliography

[1] H.J.C. Kroep et al. “ETVO: Effectively Measuring Tactile Internet with Experimental Validation”. In: *ArXiv* abs/2107.05343 (2021).

[2] Abdelhamied A Ateya et al. “Model mediation to overcome light limitations—Toward a secure tactile Internet system”. In: *Journal of Sensor and Actuator Networks* 8.1 (2019), p. 6.

[3] Gerhard P Fettweis. “A 5G wireless communications vision”. In: *Microwave Journal* 55.12 (2012), pp. 24–36.

[4] Oliver Holland et al. “The IEEE 1918.1 “tactile internet” standards working group and its standards”. In: *Proceedings of the IEEE* 107.2 (2019), pp. 256–279.

[5] Gerhard P Fettweis. “The tactile internet: Applications and challenges”. In: *IEEE Vehicular Technology Magazine* 9.1 (2014), pp. 64–70.

[6] Martin Maier et al. “The tactile internet: vision, recent progress, and open challenges”. In: *IEEE Communications Magazine* 54.5 (2016), pp. 138–145.

[7] Kees Kroep et al. “Quantifying User Experience in Sessions of Tactile Internet: The QUEST for a realtime objective metric”.

[8] Xiao Xu et al. “Model-mediated teleoperation: Toward stable and transparent teleoperation systems”. In: *IEEE Access* 4 (2016), pp. 425–449.

[9] Xiao Xu, Sili Chen, and Eckehard Steinbach. “Model-mediated teleoperation for movable objects: Dynamics modeling and packet rate reduction”. In: *2015 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. IEEE. 2015, pp. 1–6.

[10] Abdulmotaleb El Saddik. “The Potential of Haptics Technologies”. In: *IEEE Instrumentation & Measurement Magazine* 10.1 (2007), pp. 10–17. DOI: 10.1109/MIM.2007.339540.

[11] Eckehard Steinbach et al. “Haptic Communications”. In: *Proceedings of the IEEE* 100.4 (2012), pp. 937–956. DOI: 10.1109/JPROC.2011.2182100.

[12] G Fettweis et al. “The tactile internet-ITU-T technology watch report”. In: *Int. Telecom. Union (ITU), Geneva* (2014).

[13] Konstantinos Antonakoglou et al. “Toward Haptic Communications Over the 5G Tactile Internet”. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3034–3059. DOI: 10.1109/COMST.2018.2851452.

[14] Adnan Aijaz et al. “Realizing the tactile Internet: Haptic communications over next generation 5G cellular networks”. In: *IEEE Wireless Communications* 24.2 (2016), pp. 82–89.

[15] Meryem Simsek et al. “5G-enabled tactile internet”. In: *IEEE Journal on Selected Areas in Communications* 34.3 (2016), pp. 460–473.

[16] Dimitris Mourtzis, John Angelopoulos, and Nikos Panopoulos. “Smart Manufacturing and Tactile Internet Based on 5G in Industry 4.0: Challenges, Applications and New Trends”. In: *Electronics* 10.24 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10243175. URL: https://www.mdpi.com/2079-9292/10/24/3175.

[17] Blake Hannaford. “A design framework for teleoperators with kinesthetic feedback”. In: *IEEE transactions on Robotics and Automation* 5.4 (1989), pp. 426–434.

[18] Tetsuo Kotoku. “A predictive display with force feedback and its application to remote manipulation system with transmission time delay”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 1. IEEE. 1992, pp. 239–246.

[19] Tim Burkert, Jan Leupold, and Georg Passig. “A photorealistic predictive display”. In: *Presence: Teleoperators & Virtual Environments* 13.1 (2004), pp. 22–43.

[20] J Sheng and MW Spong. “Model predictive control for bilateral teleoperation systems with time delays”. In: *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No. 04CH37513)*. Vol. 4. IEEE. 2004, pp. 1877–1880.

[21] Probal Mitra and Günter Niemeyer. “Model-mediated telemanipulation”. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 253–262.

[22] Carolina Passenberg, Angelika Peer, and Martin Buss. “Model-mediated teleoperation for multi-operator multi-robot systems”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4263–4268.

[23] Bert Willaert et al. “Towards multi-DOF model mediated teleoperation: Using vision to augment feedback”. In: *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE 2012) Proceedings*. IEEE. 2012, pp. 25–31.

[24] Xiao Xu, Burak Cizmeci, and Eckehard G Steinbach. “Point-cloud-based model-mediated teleoperation.” In: *HAVE*. 2013, pp. 69–74.

[25] Xiao Xu et al. “Point cloud-based model-mediated teleoperation with dynamic and perception-based model updating”. In: *IEEE Transactions on Instrumentation and Measurement* 63.11 (2014), pp. 2558–2569.

[26] Li Huijun and Song Aiguo. “Virtual-environment modeling and correction for force-reflecting teleoperation with time delay”. In: *IEEE Transactions on Industrial Electronics* 54.2 (2007), pp. 1227–1233.

[27] Andreas Achhammer et al. “Improvement of model-mediated teleoperation using a new hybrid environment estimation technique”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 5358–5363.

[28] Carolina Passenberg. “Transparency-and Performance-Oriented Control of Haptic Teleoperation Systems”. PhD thesis. Technische Universität München, 2013.

[29] Patrick Stotko et al. “A VR system for immersive teleoperation and live exploration with a mobile robot”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 3630–3637.

[30] David Valenzuela-Urrutia, Rodrigo Muñoz-Riffo, and Javier Ruiz-del-Solar. “Virtual reality-based time-delayed haptic teleoperation using point cloud data”. In: *Journal of Intelligent & Robotic Systems* 96.3 (2019), pp. 387–400.

[31] Dejing Ni et al. “3D-point-cloud registration and real-world dynamic modelling-based virtual environment building method for teleoperation”. In: *Robotica* 35.10 (2017), pp. 1958–1974.

[32] Dejing Ni et al. “Point cloud augmented virtual reality environment with haptic constraints for teleoperation”. In: *Transactions of the Institute of Measurement and Control* 40.15 (2018), pp. 4091–4104.

[33] Xiao Xu and Eckehard Steinbach. “Towards real-time modeling and haptic rendering of deformable objects for point cloud-based model-mediated teleoperation”. In: *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*. IEEE. 2014, pp. 1–6.

[34] Manuel Wüthrich et al. “Probabilistic object tracking using a range camera”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 3195–3202.

[35] Jan Issac et al. “Depth-based object tracking using a robust gaussian filter”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 608–615.

[36] Martin Rünz and Lourdes Agapito. “Co-fusion: Real-time segmentation, tracking and fusion of multiple objects”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4471–4478.

[37] Martin Runz, Maud Buffier, and Lourdes Agapito. “Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects”. In: *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2018, pp. 10–20.

[38] Xinke Deng et al. “Poserbpf: A rao–blackwellized particle filter for 6-d object pose tracking”. In: *IEEE Transactions on Robotics* 37.5 (2021), pp. 1328–1342.

[39] Nicola A Piga et al. “Maskukf: An instance segmentation aided unscented kalman filter for 6d object pose and velocity tracking”. In: *Frontiers in Robotics and AI* 8 (2021), p. 594583.

[40] Chen Wang et al. “6-pack: Category-level 6d pose tracker with anchor-based keypoints”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10059–10066.

[41] Bowen Wen et al. “se (3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10367–10373.

[42] Bowen Wen and Kostas Bekris. “Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 8067–8074.

[43] *Home*. URL: https://www.chai3d.org/.

[44] Bulletphysics. *Bulletphysics/bullet3: Bullet physics SDK: Real-time collision detection and multi-physics simulation for VR, games, visual effects, robotics, machine learning etc.*. URL: https://github.com/bulletphysics/bullet3.

[45] *Time-of-flight: What you need to know about these new means of computer vision*. URL: https://www.avsystem.com/blog/time-of-flight/.

[46] URL: https://www.kiriengine.app/.

[47] Radu Bogdan Rusu. “Semantic 3D object maps for everyday manipulation in human living environments”. In: *KI-Künstliche Intelligenz* 24.4 (2010), pp. 345–348.

[48] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. “On sequential Monte Carlo sampling methods for Bayesian filtering”. In: *Statistics and computing* 10.3 (2000), pp. 197–208.

[49] Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. *Sequential Monte Carlo methods in practice*. Vol. 1. 2. Springer, 2001.

[50] Martin Dimitrov. *Particle filters (part I)*. May 2021. URL: https://www.lancaster.ac.uk/stor-i-student-sites/martin-dimitrov/2021/05/14/particle-filters/.

[51] Thomas M Cover and Joy Aloysius Thomas. *Elements of Information Theory*. New York: Wiley, 1991.

[52] Dieter Fox. “Adapting the sample size in particle filters through KLD-sampling”. In: *The international Journal of robotics research* 22.12 (2003), pp. 985–1003.

[53] *Understanding Kalman filters, part 3: Optimal State Estimator Video*. URL: https://nl.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html.

[54] *The world’s leading healthcare 3D scanning platform*. URL: https://structure.io/openni.

[55] Daniel Girardeau-Montaut. URL: https://www.danielgm.net/cc/.

[56] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[57] Arnaud Doucet, Adam M Johansen, et al. “A tutorial on particle filtering and smoothing: Fifteen years later”. In: *Handbook of nonlinear filtering* 12.656-704 (2009), p. 3.

[58] Katalin György, András Kelemen, and László Dávid. “Unscented Kalman filters and Particle Filter methods for nonlinear state estimation”. In: *Procedia Technology* 12 (2014), pp. 65–74.

[59] The Cherno. *Visual benchmarking in C++ (how to measure performance visually)*. Nov. 2019. URL: https://www.youtube.com/watch?v=xlAH4dbMVnU.

[60] Kees Kroep, Vineet Gokhale, and R Venkatesha Prasad. “TIM: A Novel Quality of Service Metric for Tactile Internet”. In: (2022).