Solving Tactile Internet By Faking Physics

by

Deniz Yıldırım

to obtain the degree of Master of Science in Embedded Systems
at the Delft University of Technology,
to be defended publicly on Monday September 28, 2022 at 09:00 AM.

Student number:	5375010	
Project duration:	November 18, 2021 – September 30, 2022	
Thesis committee:	Dr. R. R. V. Prasad,	Associate Professor, TU Delft, supervisor
	Dr. A. Panichella,	Associate 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

Tactile Internet (TI) allows kinesthetic interactions with a remote environment and haptic sensory feedback over a network, essentially adding a new sensory dimension to the internet. TI has a wide range of applications such as enabling remote work for professions that require human hands and tactile sensory input like car repairs or medical operations. Most TI applications have ultra low latency requirements that limit TI to short distances because the speed of light poses a constraint on how fast data can be transmitted over a distance. This thesis attempts to relax the delay requirements for enabling TI over long distances. This is achieved by locally simulating the physical interactions and tactile properties of the environment to provide instant haptic feedback regardless of network conditions.

Model Mediated Teleoperation (MMT) creates a local feedback loop using a model and lowers the network delay requirements of the system at large. We propose utilizing a physics engine with haptic functionalities as the local model, to create MMT based TI applications that can cater for a variety of complex scenarios. We implement a multi-purpose haptic framework attached to Unity that facilitates creating various TI testbeds and applications, and use it to analyze and benchmark our MMT design. We show that haptic feedback from stationary objects is completely unaffected by network delay with our MMT implementation, and narrow down the challenge to simulating dynamic objects. We demonstrate that the model inevitably diverges from the environment it simulates when movable objects are involved, and needs frequent corrections.

Deniz Yıldırım
Delft, September 2022

Contents

1 Introduction **1**

2 Related Works **7**
2.1 Challenges Of Tactile Internet . . . 7
2.2 Model Mediated Teleoperation . . . 7
2.3 Tactile Physics Engines . . . 8

3 Theory & Background **9**
3.1 Model Mediated Teleoperation . . . 9
3.1.1 Bypassing Network Delay With Models . . . 9
3.1.2 MMT With Stationary Objects . . . 10
3.1.3 MMT With Dynamic Objects . . . 11
3.1.4 Resolving Model Divergence . . . 14
3.2 Tactile Physics . . . 16
3.2.1 Existing Physics Engines . . . 16
3.2.2 Proxy Algorithm . . . 17
3.2.3 Finding The Point Of Contact With Ray Casting . . . 19
3.2.4 Acceleration Structures For Contact Point Detection . . . 20

4 Implementation & Methodology **23**
4.1 Hardware Setup . . . 23
4.2 Making The Physics Engine Work . . . 24
4.3 Enabling Mesh Objects . . . 26
4.4 Towards Model Mediated Teleoperation . . . 26
4.4.1 Different Scenarios . . . 26
4.4.2 Modular Software Design . . . 27
4.4.3 Controller Module . . . 28
4.4.4 Physics Module . . . 29
4.4.5 Network Module . . . 29
4.4.6 Sensing Module . . . 30
4.4.7 Thread Safety . . . 30
4.5 Logging and Benchmark . . . 31

5 Results **33**
5.1 Experimental Setup . . . 33
5.2 Timing Performance Analysis . . . 35
5.3 Static Model Divergence . . . 37
5.4 Dynamic Model Divergence . . . 39

6 Conclusion **41**

1 Introduction

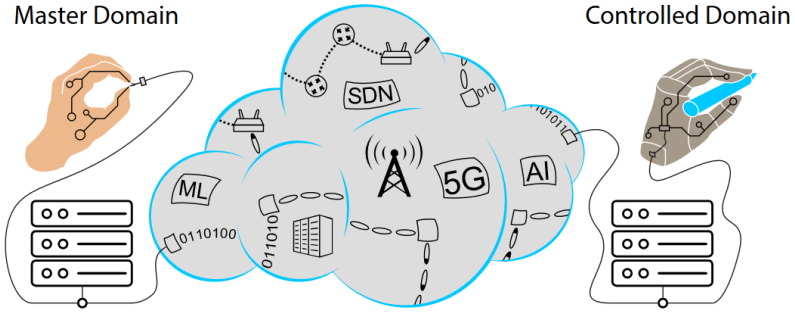


Figure 1.1: A schematic representation of Tactile Internet showing the master and controlled domains. This figure is taken from Kroep et al. with the permission of the authors. [1]

Tactile Internet (TI) aims to enable remote interaction with the physical environment at a distant location while also getting haptic feedback. After vision and hearing, which make up the core of our communication and transfer of information over the internet (and through many other mediums), touch is arguably the next most useful sense to communicate. In essence TI is two-way communication of haptic data over the internet which enables not only sensing a remote environment, but also interacting with it and manipulating it. This creates a wide range of use-cases from enabling remote work for professions that require hands-on interaction and tactile feedback such as medical operations, to allowing human tactile interaction in hostile environments such as deep sea exploration or external repairs on a space station. The possibilities are limitless and TI can make all of it a reality by creating a framework for reliable, accurate and fast tactile control and feedback over the internet.

A TI system consists of two main domains that communicate over a network. These domains which used to be called master and slave domains, are more recently called master/controlled or operator/teleoperator in literature. The master side consists of a human and a device that captures the human's movements and sends them to the controlled side over the network. The controlled side then mimics the movements that it receives and sends back haptic feedback data such as forces and textures from the surfaces it touches. The master side then communicates the haptic feedback to the the user. This feedback loop enables the user to in a sense embody the robot and physically be present at a distant location. Everything the human does, the robot mimics, and everything the robot senses, the human feels. There are different setups for both the master side and the controlled side, varied by the available devices and the requirements of the environment. For example the master side can provide haptic feedback by mimicking forces sent from the controlled side or simply with vibrations, the master-side device can be a wearable device or a separate controller, the controlled side device can be

a humanoid robot or just a robot arm etc. While all use cases come with their own set of requirements, there are certain requirements of TI that need to be fulfilled by any TI application, which have been the focus of TI research for a couple of decades.

Arguably the biggest challenge among the requirements of TI is delay, as is with any real-time system. A TI application always includes a feedback loop where the actions of the user directly effects the feedback it receives, such as pushing an object and feeling a reactionary force. Sufficient delay can render the system unusable where the sensory feedback no longer feels correlated with the action. When one thinks of the many examples of future use-cases of TI such as remote medical operations, it quickly becomes apparent why low latency and reliability of sensory feedback are critical for a TI application. While each component needs to conform to these requirements, i.e. the actuators and sensors also need to operate fast and reliably, the bottleneck especially in terms of latency is undoubtedly the network communication between the nodes.

Research shows that the upper limit of acceptable delay for sensory feedback in a TI applications is $1\,\mathrm{ms}$, which is often called "the $1\,\mathrm{ms}$ challenge" or the Ultra Low Latency (ULL) requirement in TI research. A $1\,\mathrm{ms}$ round trip means that even if the latency of every other component of the TI application is ignored, data should arrive from one node to the other in $0.5\,\mathrm{ms}$. This creates a difficult challenge where the distance between the nodes are limited by physical constraints, mainly the speed of light. Considering it takes roughly $55.65\,\mathrm{ms}$ for light to travel from Delft, Netherlands to Sydney, Australia, this requirements makes TI impossible except for over short distances, unless a way to loosen the requirement is found. It's not all bad news though, because the other network requirements, reliability and throughput, do not pose such an impossibility. The throughput requirements are low for a TI application especially when compared to other types of data such as video streams. In terms of reliability, TI community suggests an ultra-reliability requirement of 99.9999%. However this percentage is mostly speculation unlike the experimentally quantified ULL requirement, and recent research that does a variety of subjective experiments even with nearly 50% packet loss, operators barely feel any disturbance at all [1]. This suggests that the proposed ultra reliability requirement is in truth not a necessity for certain TI applications that share the same properties as the ones tested in [1]. Thus, the ULL requirement is unarguably the network bottleneck against a high quality TI application, and it is the challenge that this thesis aims to tackle.

The reason why low latency is so critical for a TI system is that a delayed system can create not only delayed but also inaccurate and unstable haptic feedback. Imagine the scenario depicted in Figure 1.2 where a finger at the master domain pushes a button in the controlled domain. In an ideal system without delay, the finger's movement would instantly be mimicked the controlled domain, causing force to be applied to the button and an equal reactionary force being applied at the master domain. In the realistic case where there is network delay, the user will keep pushing until a sufficient reactive force is felt, but with delay, they will push considerably past the point of contact. This will cause the robot to apply more force on the table than the user intended, and as a result cause the user to feel a much higher feedback force than expected. While there is research to minimize the effect of delay on user experience such as lowering the spring constant when delay is present, network delay is always detrimental to the user experiences and the quality of experience unavoidably drops below acceptable limits past a certain distance. The logical conclusion from this is that either TI is only feasible over short distances, or a creative way to loosen this requirement must be found to enable TI over long distances. This thesis aims to introduce the idea of enabling high-quality TI over long distances by reducing the perceived delay by simulating physics.

Our idea is that both the controlled side and the master side can have local simulations of each other that provide uninterrupted haptic data, the perceived delay of which is unaffected by the network delay. Simulations would still make use of the real haptic data that arrives over the network but use it to correct divergences in simulations. Since the local simulations would not be visually shown to the user, one could be creative with fixing divergences without making the user realize something was wrong to begin with. Theoretically speaking, if we were able to create a perfect model of the environment at the other side, then the need for a network would disappear entirely once the model is created because the model would be identical to the controlled environment. While such a perfect model is not possible with the technology of our time and gaps in our current scientific knowledge, unlike the ULL requirement limited by the speed of light there is no physical limit to this approach that we know of. The accuracy of the model adds a new dimension for performance and loosens the network requirements that are capped by the speed of light, which in turn make long distance TI communication possible.

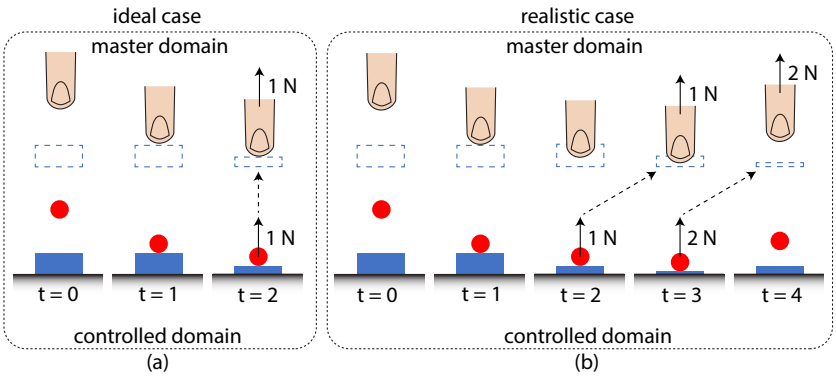


Figure 1.2: Illustration of the detrimental effects of the network on TI interaction. The operator intends to press a switch (blue block) through the teleoperator (red circle) using a tactile glove virtually (dotted blue block). (a) In the ideal case, force is experienced right at the instant of contacting the object. (b) In a realistic network case, the force feedback is transported with a variable delay and causes significant performance degradation. This figure is taken from Kroep et al. with the permission of the authors.[2]

This idea is fairly similar to a topic of research in control systems called Model Mediated Teleoperation (MMT) which aims to solve problems that arise with delayed haptic data with a local model. In simplest terms, a model of the controlled side environment is created at the master side, which provides instant haptic feedback to the user by simulating the controlled side, meanwhile the parameters used by this model are extracted and/or estimated at the controlled side and sent over to the master side to correct the model when it diverges. This design circumvents the network requirements by moving the source of sensory data from the controlled side to the master side. While MMT is an old topic of research in control systems, the idea is novel in TI research and it fails to address several key aspects of TI. The focus on MMT is solely on static environments with stationary objects (except for one preliminary paper [3]) so there is little to no consideration on the controlled-side interactions with objects which is evident by only the master side having a local model. MMT only focuses on creating a local loop at the master side and representations of MMT, such as Figure 1.3 indicate that the delayed movement data from the master side is directly used by the controlled side. Another effect of only considering stationary objects is that the master-side model can be fairly simple as it only needs to account for haptic data from a stationary surface. While this led to many developments for simulating haptic surfaces with varying textures and physical properties, it caused the developed models to specialize for static environments over time to the degree that some of them are completely unadaptable to dynamic environments, such as black-box neural networks that estimate the feedback force from the master-side controller movements which would practically never converge in a dynamic environment. Lastly, sensing at the controlled-side and parameter extraction in MMT also specializes for static environments and often only use positional and force data of the actuator interacting with the environment which makes it impossible to simulate an object before a haptic interaction happens. That being said there is research that makes use of point clouds (albeit being used only in static environments) which can be adapted to dynamic environments.

Considering that creating a perfect physical model of an environment is presently impossible, any local model would inevitably diverge from the environment it simulates. Because of this, the local model must still rely on network communication to update its parameters when it diverges from the environment it simulates. For this reason, the parameters of the model are extracted and/or estimated and are sent over to the model over a network which then uses them to determine if it diverged from the environment and to update its parameters if so. In this regard, the network latency directly effects the correctness of haptic feedback instead of its delay, so network conditions still directly affect user experience, but the burden is shared by the model depending on how accurate it simulates the environment. Figure 1.3 shows the state-of-the-art MMT approach for this idea where a master-side local model provides data without delay, regardless of network conditions. Our approach also envisions a local loop at the teleoperator side which we argue is crucial for dynamic environments. It is also important to note that the video and audio feed are not traditionally part of the local model and are simply streamed from the controlled side over the network. This choice can be justified by the fact that haptic feedback has much lower latency requirements compared to visual and auditory feedback in a TI application, and it is not possible to simulate a video feed at the master side. That being said, there is research that

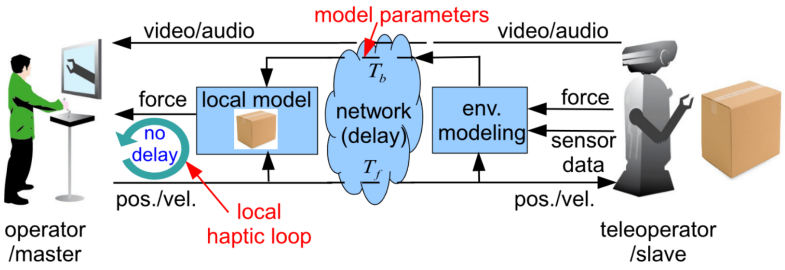


Figure 1.3: State-of-the-art MMT control flow. The environment modeling estimates the necessary parameters to update the local model and sends them over network. The local model provides delayless force feedback to the operator and is updated with new parameters when needed. [4]

proposes showing an artificial 3d environment to the user to mitigate visual delay which in theory simply expands the existing MMT design by allowing the model to also provide visual data. For the purposes of this thesis, it is assumed that delay in audio-visual data is less crucial for a good user experience and that the difference in delay between the audio-visual and haptic output would not be noticeably detrimental to the quality of experience.

In an environment with stationary objects, which is the case for almost all MMT research, the model is only concerned with unchanging geometrical (position, orientation, shape) and texture properties of surfaces that are interacted with. Since these properties of objects are assumed to be static, with correct parameters the local model can yield near perfect real-time feedback and the divergence in this case is for the most part only caused by the wrong sensor readings or external disturbances in the environment at the controlled side. We claim that for a static environment, the MMT approach completely enables TI over long distances for most use cases. While the model still needs to be updated occasionally for wrong sensor readings and external interactions but since the ground truths for model parameters are unchanging, the model will eventually converge into near perfection. Furthermore, it is much harder to argue for a $1\,\mathrm{ms}$ requirement in a static environment because keeping objects stationary significantly reduces the amount of possible scenarios that can reduce the quality of experience. Since a static environment both reduces the latency requirements and can be simulated by a model that is much more accurate and reliable over time, this thesis aims to show that a local model solves TI completely for static environments.

This thesis then aims to expand MMT to work in environments with dynamic objects, discover the challenges specific to dynamic environments and propose solutions to said challenges. A dynamic environment brings numerous new challenges, first of which is that the required complexity of the model increases considerably which will in turn cause bigger divergences from the environment it simulates. In a dynamic environment by definition ground truth values of certain model parameters such as object positions and rotations change over time which requires a model that is able to also predict the first and second order derivatives of parameters like object position and rotation. Luckily, mechanical physics are fairly well understood and numerous physics engines that accurately simulate dynamic environments already exist. Furthermore, in a TI system feedback loop, most of the changes in the controlled side environment are directly caused by the master side movements and vice versa which means the model has a good chance to accurately simulate changes made to the other side. This interaction is also our reasoning for having a model of the master side at the controlled side that in theory simulates the master side movements but in a more practical sense attempts to mimic the intent of the master side rather than blindly copying movements. For example if the user at the master side pushes an object that was mistakenly simulated as closer to the robot than it really is, then instead of mimicking the movements of the user and not push the object, instead the controlled side attempts to mimic the intent of pushing the object regardless of positional differences. This thesis argues that having this local model at the controlled side aids minimizing the divergence of the model at the master side. While not explored in this thesis, future work can also focus on predicting user movements with machine learning to create a more complicated model of the master side. However this will undoubtedly be a more complicated and error-prone method because unlike physics, human intent is not deterministic (even if one does not believe in free-will, the technology simply isn't there).

It is important to note here that there are fundamental differences between the master side and the controlled side and that it is not a completely symmetrical system for the purposes of this thesis. Firstly as just mentioned, the movements of the master side are made with human intent while the movements at the controlled side are results of physics and interaction caused by master side movements. While there are certainly TI applications that have a human user at each side, this is not the focus of this thesis and such a scenario could majorly reduce the effectiveness of the local model. This difference causes master side model to be the main focus while giving the controlled side model a much more simpler, auxiliary function. When there is a divergence between the master model and the controlled environment, it could be that fixing the error at the controlled side is impossible. Imagine a box sitting at the edge of a table, a small force could cause the box to fall on the ground and if it doesn't fall in the master-side model then there is no way to rewind time at the controlled side to change the position of the box. As such, the controlled side model only takes on a preventative duty in reducing model divergence, while the master side model also takes on a corrective duty of fixing divergences that cannot be corrected at the controlled side. This aspect of dynamic environments is not addressed by state-of-the-art MMT implementations and is novel to both MMT research and TI research at large to the extent of our knowledge. MMT research on using a physics engine as the master-side model is non-existent to the best of our knowledge, predictably because they focus on static environments which does not require a physics engine.

Contributions: In essence this thesis intends to relax the timing requirements of TI. For this purpose an MMT with a physics engine is theorized, implemented and analyzed. The specific contributions can be listed as:

1. We added TI support to the Unity game engine which enables the creation of wide variety of TI applications.
2. Using reverse engineering we derived methods for haptic interactions in physics engines.
3. We propose an advanced MMT design that utilizes a physics engine as its local model to significantly reduce delay requirements of TI in complex scenarios.
4. We implemented support for MMT with a physics engine for both virtual and real environments, in our TI framework.
5. We demonstrated that the MMT implementation of our TI framework met the required timing requirements of $1\,\mathrm{kHz}$ refresh rate.
6. We demonstrated the our MMT implementation eliminated the negative effects of network delay for static environments
7. We identified and demonstrated that enabling accurate interactions with movable objects in dynamic environments is the key challenge of MMT.

2

Related Works

2.1. Challenges Of Tactile Internet

There is extensive research in both the potential of haptic communications over the internet and its challenges. E. Steinbach et al. and A. El Saddik are good examples to preliminary research that looked into the potential of haptic feedback over a network in late 2000s and early 2010s [5][6]. These early works explored the design space for haptic communications, determined quality metrics and hypothesized possible future use cases of the technology. Other papers further formulated and evaluated the many challenges in tactile internet [7][8]. There is also research on effectively measuring these quality of a tactile internet solution with quantifiable metrics [9]. One of the most important and hardest to achieve requirements of tactile internet turned out to be the 1ms challenge, which asserts that a quality haptic application would need to have a total end-to-end delay no higher than 1ms. Several studies show that for haptic applications, delay will be noticeable if it is larger than 1 ms which can cause an unrealistic and nauseating experience for the user. A big focus of 5G development has been on meeting the low-delay real-time requirements of tactile internet [7][10][11][12].

While reducing the end-to-end delay to achieve the 1ms second challenge has been the goal of much research, a big chunk of effort was also spent on finding ways to relax this tight constraint that is impossible to achieve in some scenarios because of laws of physics. The yet unpublished paper of Kroep et al. experiments with using virtual springs for feedback force calculation which they call ”network compensation spring” and reducing the spring constant relative to delay to improve the quality of experience [2]. Such research hints that tactile internet with more than 1ms of delay can be further relaxed by using solutions that improve user experience in scenarios with long delay.

2.2. Model Mediated Teleoperation

Using a model of the environment to circumvent the effects of delay in robot teleoperation with a tactile feedback loop is a relatively old idea. A number of research articles exist from early 1990s that briefly introduce it as a possible solution to reduce perceived delay [13][14]. These papers focus on robot control over a distance with kinematic feedback but don’t delve deep into haptic feedback and the field of tactile internet that was not a mature enough topic of research at that time. The idea of Model Mediated Teleoperation (MMT) is only revisited in mid-2000s and had continued interest thereafter with the increasing interest in VR and robot teleoperation, along with the improved models and simulations that made it a viable option. A number of people did research on using models to solve the network delay challenge of tactile internet during this time [15, 16, 17, 18, 19]. These papers from late 2000s and early 2010s focus on simple scenarios with reduced degrees of freedom and simplified dynamics such as stationary objects, and generally explore the main challenges of MMT in tactile systems which is almost always related to the updating of the model. It was quickly discovered that the divergence between the model and the environment is an issue even with static environments due to wrong sensor readings etc. and the focus mostly shifted towards reducing these divergences with better parameter estimation at the controlled side.

Most recent research on MMT and arguably the state of the art is set by multiple research articles by a group of researchers led by X. Xu. The earlier research from Xu explores point-cloud based sensing

of the environment and parameter extraction from point cloud data [20][21]. Data reduction is also seen as a crucial step by this team of researchers and several data reduction techniques are proposed such as deadband filters [22]. There is also some preliminary research on MMT with movable objects but the focus is still on parameter estimation on from the slave-side environment [3]. In general, research on the challenges of updating the master-side model is limited, especially with movable objects. A survey of researches on MMT further shows that existing research almost entirely focuses on parametric environment modeling and online parameter estimation at the slave side, and data reduction techniques to ensure the estimated parameters can be reliably sent to the master side [4]. While the survey talks about existing research about how the master side can be updated during transition-state, and the effects of model divergence, research on this topic is non-existent for movable objects.

2.3. Tactile Physics Engines

To the best of our knowledge, no MMT research considers using a physics engine as the master-side model but many use haptic rendering algorithms which, if used along with a traditional physics engine, make a tactile physics engine. Outside the field of MMT, tactile physics engines are used extensively in TI research as test beds that allow testing for different network conditions and techniques that aim to improve user experience, in a controlled, simulated environment. When used in this matter, the tactile physics engine is used as a replacement for the controlled side that would otherwise be a real environment that a real robot interacts with. Since these test beds are already created to simulate a realistic controlled environment, they are perfect for being used as master-side simulations of the controlled environment.

Chai3d seems to be the gold standard and the most common haptic rendering tool used for creating TI test beds. Chai3D was developed at Stanford University in 2003 as a framework for haptic computer applications. The early uses of Chai3D were mostly applied to virtual environments in AR and VR research [23] [24]. While Chai3D is still frequently used in AR/VR research, it also started being used for test beds for TI research primarily led by E. Steinbach and X. Xu in the second half of 2010s which is roughly when the research for 5G started to take off [25][26][27][28]. Besides Chai3d, V-REP and its later offshoot Coppelia are used for building TI test beds [29][30][31]. These papers are mostly in the field of robotics which explains why V-REP and Coppelia which are mainly robotics simulation frameworks are used as TI test beds. More recently research by K. Kroep, V. Gokhale et al. aimed to come up with quantifiable metrics to measure TI and developed an all-purpose testbed for TI that used Chai3d for its virtual environment [9][1][32].

3

Theory & Background

3.1. Model Mediated Teleoperation

3.1.1. Bypassing Network Delay With Models

As discussed, traditional TI applications have severe range restrictions capped by the speed of light and this thesis aims to bypass this restriction by reducing the perceived delay with local models. It is important to reiterate here that we were unaware of MMT as a research field until the late stages of this thesis. Instead, the idea was to use our already working dynamic virtual environment (initially developed as a test bed) to simulate a real environment, and reducing the perceived delay by doing using this simulation at the master side. Because our simulation was already made for a dynamic environment, we designed our TI solution to accommodate movable objects from the get-go. Later when we discovered MMT we realized that despite the similarity of the main idea several parts of our approach was novel including dynamic environments or using a physics engine as a local model.

Figure 3.1 illustrates our vision for how MMT changes the traditional TI control loop. The traditional TI control loop has the master side controller (C) and the controlled side environment (G) separated by network. Our design aims to create an internal control loop at each side that can run uninterrupted by network, by using simulations of C and G, named C' and G'. While the traditional control loop network is destabilized by poor network conditions, the MMT case is immune to these effects as long as C' and G' are accurate descriptions of C and G. To prevent models from diverging from the environments they mimic, a network connection is still needed between the model and what it simulates. The relevant parameters for the models are sent over network and are used by the models to fix their divergences.

When the master side user moves the controller C, on top of sending the movement data over network to be mimicked by the controlled side, the same data is also used by the local G' to produce instant haptic feedback. At the controlled side, C' attempts to mimic the intended behavior of the user and then haptic feedback data along with object data extracted from the environment G are sent back over the network. Upon receiving these G' is updated according to newly received data if needed, in a manner as unnoticeable to the user as possible. The haptic feedback received by the user always come from G' which enables undelayed feedback but also makes it critical to correct G' over time because in a dynamic environment G' will inevitably diverge from G and this divergence will increase over time if not corrected. Additionally, correcting these divergences instantly without care would bring back the original problem of force spikes present in delayed tactile systems, nullifying the benefits of using local model.

To initialize these models in the first place, relevant parameters from the environment need to be sent. Some of these can be one time packets, such as which objects exist in the environment, while others such as object positions can need periodical updates to detect and fix model divergence. Since the master side consists of a haptic controller device and a human operator, initializing C' requires little to no steps. If needed, device model and configurations can be sent at initialization. All the data required for the initialization and maintenance of C' comes from built-in sensors on the haptic controller device so no additional hardware is required at the master side. G' on the other hand attempts to simulate a physical environment with dynamic objects that can be interacted with by moving the teleoperator

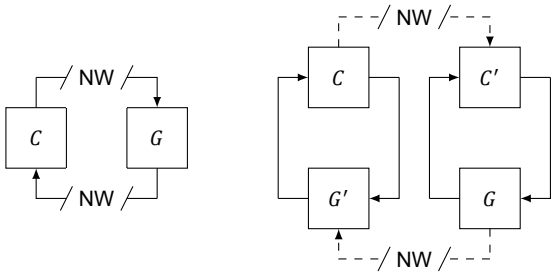


Figure 3.1: Control system flowcharts for traditional and model mediated tactile feedback systems. C represents the master-side controller while G represents the controlled environment. In the traditional system network conditions directly influence the quality of the feedback loop. MMT implements C' and G' which are simulations of C and G which allow for a continuous control loop uninterrupted by network conditions. C' and G' need periodic updates from their real counterparts but are able to provide an output regardless of the frequency of updates.

robot, therefore data from the controlled side device is not enough to initialize or to maintain it. For this purpose additional sensors that detect and track objects in the environment are needed at the controlled side. In this thesis, this is done with a Kinect device that extracts a point cloud of the environment.

3.1.2. MMT With Stationary Objects

Even in an environment with stationary objects, an MMT system has additional requirements and challenges compared to a traditional tactile system. Firstly, the objects in the real environment need to be simulated with correct properties such as friction and stiffness as well as shape and position, which requires observing the environment at the slave side with sensors and extracting relevant model parameters from sensor data. This step is called parameter estimation in MMT research and needs to happen continuously until model converges because wrong sensor readings and external changes to the environment can cause the model to be wrong at first. However the static nature of the environment dictates that the ground truth for parameters are unchanging without external interactions which makes periodic model updates unnecessary. Secondly, a reliable and accurate model of the environment is needed to make sure that the forces experienced by the user match those experienced by the robot, with little perceived delay. When objects are static, this model only needs to implement a haptic rendering algorithm for static surfaces. Lastly, as this sensor data can be wrong, the master side model should be corrected over time so a sensible network protocol to continuously send new data and a way to update the master side model are needed.
In summary, the requirements of an MMT system with stationary objects can be stated as follows:

1. Parameter estimation

   Accurate parameter extraction from the environment is needed to initialize the master-side model. In existing MMT research this is usually gathered from the actuators and sensors on the robot although there is also research that uses point-clouds to extract geometrical information from the environment which is what is done in this thesis. In a static environment, the extracted parameters from the environment focuses mostly on haptic properties of objects such as softness and friction, and simple geometric properties such as position and shape of the static object.

2. Master side model (G')

   The model of the static environment is concerned on estimating and mimicking the feedback force experienced by the teleoperator. In essence the ground truth for environment parameters is assumed to be unchanging so the model only needs to have a reliable haptic rendering algorithm for finding the feedback force and tune its parameters correctly at initialization. The model does not attempt to estimate the change of the environment either by external effects or the effects of teleoperator's interaction with the environment.

3. Teleoperator control scheme (simplified C')

It is very difficult if not impossible to reliably simulate human intent so in our vision for MMT, C' takes on more of an auxiliary spot. As previously mentioned in Introduction, the primary raison d'être of C' is to preventatively reduce model divergence. Since model divergence is not a concern in static environments, C' can be very simple or even non-existent in theory. In practice, a control scheme is still needed to determine how much force to apply on objects after contact, which we argue can be represented as a simple form of C'. It's important to note that G' needs the same input as G' so if force determination is considered to be a part of C', then the master side needs its own identical C' as well.

4. Network transmission

   A way to transmit data from C to C' and G to G' is needed. Since G' is mimicking a static environment, network delay has no effect on model performance after convergence of its parameters. Delay from C to C' and G will cause the teleoperator to lag behind, but in a static environment perfectly mimicked by the master side, the delay at the controlled side has no risk of imitating the master-side movements wrongly. Thus, poor network conditions are acceptable.

Most existing research on MMT focuses on stationary objects to simplify the problem and focus on correct haptic force feedback at the surface level of an unmovable object. Past research on MMT claims that even in static environments, the local model can require updates during which it enters "transient state". Several methods are proposed to maintain system passivity during transient state such as adaptive damping or moving and scaling the environment and its objects in a manner that doesn't cause force spikes. We claim that this is unneeded for static environments because the model needs to be updated only until its parameters converge to their true values. Likewise advanced control schemes that aim to maintain transparency and equipment safety at the controlled side are unneeded for static environments because transient state does not occur past the convergence of model parameters. Finally, MMT research puts much focus on data reduction methods such as deadbands that attempt to reduce network load by reducing packet sizes and frequencies. It is our claim that for static environments, network requirements are much more loose and that a satisfactory TI experience has been demonstrated with network delay upwards of 5 seconds. In our observation, when limited to stationary objects, the state-of-the-art MMT fully addresses the latency concern of TI and makes TI possible over long distances.

3.1.3. MMT With Dynamic Objects

In our view, dynamic objects are the main challenge in building a functional model-mediated TI system. One might think that if starting positions and physical properties of objects are known, one can use these to find the acceleration and in turn velocity and position of each object with simple Newtonian physics. In that case the only added difficulty of dynamic environments would be the higher number of parameters and more complicated models. This is not the case however because the models can never be as precise as real life. Even if the model technically has the right parameters, the simple fact that parameters of a model are bound to be discrete makes small differences inevitable. This is the case for static environments as well but unfortunately in a dynamic environment these differences accumulate, causing the model to diverge further from reality over time. This inevitable unbounded divergence is the core reason why dynamic environments are harder to model mediate. Another key difference is that interactions at the controlled-side now matter. In a static environment, the controlled side mimicking the master side would be practically useless except for making gestures or making sounds by hitting stationary objects, as opposed to a dynamic environment where the master side movements directly manipulate the controlled side environment.

The model of a dynamic environment also to be significantly more complex than a haptic rendering algorithm for a stationary surface because it now also needs to simulate objects positions, orientations, and their first and second order derivatives. Moreover the objects can now also collide with each other, not just the haptic device. However besides model complexity, the most obvious requirement that stems from this challenge is that the master side model needs frequent updates. Updating a model that the user is actively interacting with can cause unexpected force spikes that can reduce user experience and can harm the haptic controller. A sudden update to the model creates an effect similar to that created by network delay in traditional TI applications, where because of delay the controlled side penetrates too

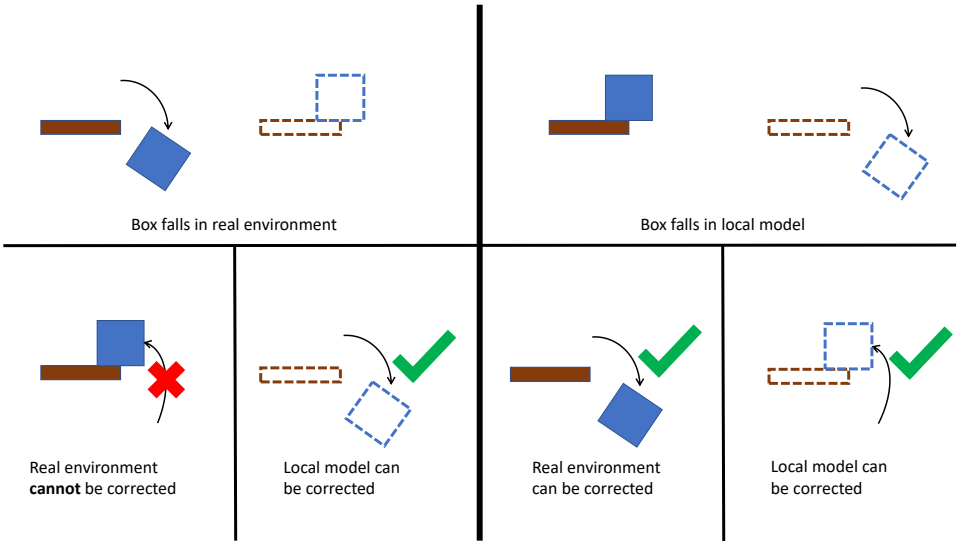


Figure 3.2: Example scenario for model divergence where a blue box is at the edge of a brown table. Dotted lines show the simulated objects in the local model. If the box falls off the table in the real environment this is an irreversible event so only the model can be updated to resolve the divergence. If the instead simulated box falls in the model, then either side can be corrected to resolve the divergence.

much into the object and experiences a sudden push back. In this case because of model divergence user can penetrate too much into an object and experience the same phenomenon. The benefit of using a model against the traditional TI design in this case is that having full control over the model allows updating its parameters gradually and smartly without making it noticeable to the user.

The network becomes a critical component as a result of the necessity of periodic model updates. The divergence of the model gets worse over time which means the frequency of updates directly influence model accuracy and in turn user experience. Coupled with the increased number of parameters for every object in the environment, the dynamic case introduces numerous additional network requirements. There is also an important result of network delay that the received parameters will have old values. The positional and rotational parameters of objects constantly change and so when the model receives a parameter over the network, its real value in the environment will have changed. If the model does not take this delay into account and updates its values blindly, then the model will be delayed and thus become completely useless. This means that upon receiving a parameter, the model needs to take into account network delay (which now needs to be measured) and compare it to one of the past values of the parameter (which now need to be recorded) to determine if the model actually diverged. Then if there is indeed a divergence, the model needs to somehow derive the real current value of said parameter using its received past value.

Another interesting property of dynamic environments is that the changes to the environment can be irreversible. For example imagine a box at the edge of a table, as in Figure 3.2. Because of model divergence, the box might fall on the floor in the real environment while it is still on top of the table in the local model, or vice versa. If the box falls in the real environment, then it is not possible to rewind time and bring it back to the top of the table, so the local model needs to be corrected. In the opposite case however, if the box falls in the local model then one option is still correcting the local model but another option is also possible: to correct the divergence by also pushing the box down in the real environment. This phenomenon allows the controlled side to implement C' to aid in reducing the divergence between G and G'. In this thesis C' is given a preventative role where it tries to mimic the intention of C to attempt to reduce the divergence of G from G'. So in this example if the box did not fall in G', then C' could reduce the applied force so that it also doesn't fall in G. In our proposed version of MMT C' does not have a mechanism to correct past mistakes so it can tune the applied forces on objects and the position of the haptic device in relation to other objects, but if that is not enough it will not make an attempt to correct the divergence even if it is possible. If the box fell in G', it will make it more likely that the box will fall in G, but if the box still doesn't fall C' takes no further action to push the box down. Future work can focus on exploring which dynamic scenarios create reversible and/or correctable situations and develop a corrective C'.

With these difficulties, the requirements of MMT with stationary objects are changed and new ones are added among them. Below is a list of requirements for MMT with dynamic objects. The changes to the requirements that also exist for the stationary case are *emphasized*.

1. *Fast* parameter estimation *with additional parameters*

   On top of the haptic properties of objects such as softness and friction, and simple geometric properties such as position and shape, a dynamic object has additional parameters such as first and second order derivatives of its position and rotation, as well as its mass. The positional and rotational properties of objects can change over time which requires not only do objects need to be discovered and their parameters identified, but they also need to be actively tracked.

2. Master side model (G') *with dynamic physics*

   Models specialized for static surfaces lack the functionality to simulate dynamic interactions between objects. Some methods such as neural networks that can work for static environments are rendered ineffective by the constantly changing parameters of a dynamic system. A tactile physics engine that merge haptic rendering techniques with dynamic physics engines is needed for an efficient model.

3. Stealthy Divergence Resolution

   With constantly changing parameters, the divergence of the master-side model needs to be fixed with periodic model updates. These updates must be done in a manner that prevents instant and elevated changes to the feedback force when an over-penetration or under-penetration happens due to divergence.

4. Teleoperator control scheme (*smarter* C')

   A smart C' component to minimize model/environment divergence at the environment side is needed. The irreversible nature of the real environment limits the functionality of C' but it can attempt to prevent divergence by smartly imitating the intentions of C rather than just its position. A smart C' is a requirement with dynamic objects because divergence resolution is a bottleneck and doing it on both sides is necessary with the current performance of models. With a good enough model in the future where divergence happens slower, there can be less need for C'.

5. *Fast* network transmission

   Delays on both directions cause the local model to diverge more. In that sense MMT in a dynamic environment does not fully solve the latency requirements but simply loosens them. While the aim is to allow delays higher than 1ms, a fair amount of delay will still cause the user experience to deteriorate as a result of increased model divergence. Additionally, the network needs to accommodate bigger and more frequent packets compared to the static case.

6. Parameter derivation with delay

   If network delay is present, then the environment parameters will have changed by the time they are received by the local model. The local model must take into account network delay and find which past value of the parameter it corresponds to. Then if divergence exists it must derive the present value of the received parameter. So a dynamic MMT system with delay needs to measure network conditions, keep track of old model parameters, and have a method of derivation to find present values of said model parameters.

Out of these requirements, this thesis mostly explores requirements 2, 3, 4 and 6. There is extensive research on network requirements for TI applications and ways to improve network conditions for TI, it is outside the scope of this thesis. Requirement 1 is the thesis topic of a fellow TU Delft student Kilian and while our implementation for MMT which is a collaboration of multiple students includes his implementation of point cloud sensing, it is not a part of this thesis. Out of the requirements focused on

in this thesis, requirement 2 is further explored in the Tactile Physics section while the requirements 3, 4 and 6 are explored in the Resolving Model Divergence subsection of the Model Mediated Teleoperation section.

While it can be argued that parts of existing research for MMT with stationary objects that we argued to be unnecessary for static environments would cover many of these requirements, this is not fully the case. The existing MMT solutions fall short when it comes to tackling dynamic objects because they are designed to solve problems present in static environments. The models in these research are unsuitable for dynamic environments. Some approaches such as neural networks that train to estimate experienced force as a function of position are rendered completely useless by a fast changing environment that the model cannot adapt to fast enough. The aim in existing MMT research is often to reduce the time spent in transient state as much as possible while the transient state inevitably happens periodically. This directs the focus of most research into perfecting parameter estimation and better network conditions that they believe are the main influence behind transient state which we argue is not only irrelevant for static environments, but also completely untrue for dynamic environments. The only existing research on MMT with dynamic papers is from Xu et al. [3] which is a preliminary work that in our opinion does a decent job in identifying new challenges but still fails to deliver results by focusing on parameter estimation that still includes only includes unchanging variables and not positional and relational variables which we believe are a necessity to reduce model divergence. On the bright side, the paper existing solutions to reduce the network load such as fixed or perception-based deadband implementations and other data reduction techniques can largely still be used for the dynamic objects with similar results [3].

3.1.4. Resolving Model Divergence

Most requirements of dynamic MMT deal with model divergence in some way. In this subsection our theorized solutions to these requirements are explained. Not all of these theorized solutions made it to the final implementation due to timing constraints of the thesis project. In particular, we explore how an MMT design with dynamic objects can address stealthy divergence resolution, a smart C', and parameter derivation with delay.

It is probably easiest to explain divergence resolution with another concrete example as shown in Figure 3.3. In both scenarios depicted, the user attempts to get a dynamic box object through a static gate. Similar to Figure 3.2, this can be an irreversible (or rather hard to reverse) event if the box gets past the gate in the real environment, therefore if the local model gets stuck then the only option is to update the local model. In the opposite case, either side can resolve the divergence. Since we only task G' with divergence correction, our proposed solution in both cases (and in any case of model divergence) is to resolve the divergence at the model rather than at the environment. Our model update in theory would work by instantly updating the position of the object with the correct one and to move the device and proxy pointers along with the object to attempt to keep the feedback force constant through the update. In other words the model tries to keep object positions correct at all times but instead updates the proxy and HIP positions gradually to prevent unstable haptic feedback. A challenge shown in this example is that a divergence in object's orientation can render it impossible to keep the proxy to device vector equal in both magnitude and direction, depending on the object's shape. In this case either the pointers can rotate along with the object which will keep the force magnitude unchanged while changing it's direction, or only the relative position of the device pointer to the object's center can be maintained in which case the direction of the force remains unchanged but the magnitude varies. In the second option, the proxy pointer's location needs to be recalculated. Further research and user study is needed to determine which of these yields better results.

In the example given in Figures 3.2 and 3.3, a smart C' could have prevented the divergence from happening in the first place as well. In the first example a slightly higher or lower force applied to the box can ensure that the environment gets the same result as the model. Likewise in the second example, ensuring that the force is applied at the same relative point of the box in the environment as it was in the model, can ensure that rotational divergence is minimal. We claim that a preventative approach in C' can be critical for minimizing model divergence. We propose 3 different C' implementations that increase in complexity. While we theorize that the smartest C' implementation will cause the highest reduction of divergence, but different scenarios can favor one over the other and more research is needed to quantify their effect on model divergence in different scenarios. The simplest implementation of C' would be to simply use an artificial spring to apply force in the direction from the robot's HIP

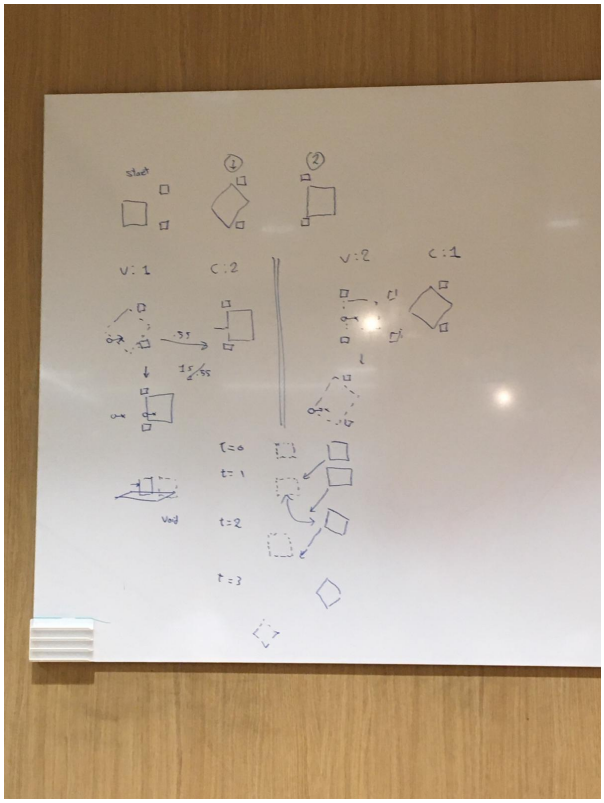


Figure 3.3: Two examples of divergence and how they are solved at the master side. At first example at the master side the box is stuck at the gate but not on slave side, so the master side object is teleported to the correct location while the device and proxy pointers are moved with it to keep the feedback force unchanged. The second example is the opposite scenario where the box is stuck at slave side but not at master side, in which case the object is moved to the stuck position while the device and proxy pointers are again moved along with it. The divergence in orientation can make it impossible to find pointer locations that will keep the force unchanged.

to the received controller HIP, which is not included in the list. The following are our proposed C' implementations:

1. Only use position from C. Move towards position in predetermined velocity if no collision, use artificial spring to apply force if there is active collision.
2. Use position and force from C and G'. Move to position if no collision, apply given force if there is active collision.
3. Use position, force, and position relative to closest object from C and G'. When in free space move to position. When close to object move to relative position, when in active collision apply given force.

The final point to consider in model divergence resolution is that both C' and G' need to take into account network delays. As shown in Figure 3.4 blindly correcting with the latest received data at either side causes the receiving side to lag behind, which eliminates the raison d'être of MMT which is to lower perceived delay using a model. If model itself is delayed then the model brings no benefit to the system. To overcome this challenge, the network conditions must be constantly surveyed to estimate the one-way delay over the network. The master-side model must keep a list of past local parameters up to a certain point in the past. This delay must then be used to find which version of the object in the past it must be compared to. If there is a divergence, the correct present parameters must be derived from the old parameters of the object. If the received parameter is sufficiently bigger or smaller than the local parameter at the same time step, then the rates of change at each time step since then can be applied to the received parameter to derive its current value. Formula 3.1 provides a simple algorithm to determine the master side position x at any time t. When new data x_c arrives from the controlled side, it is compared with the master side position with the same timestamp as the received position.

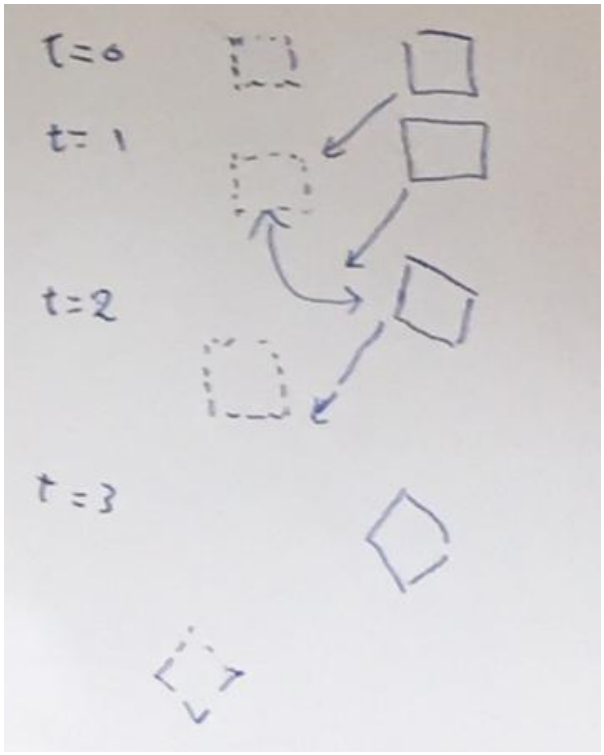


Figure 3.4: The effect of network delay on divergence correction. Correcting blindly causes the receiving side to lag behind, effectively nullifying the benefits of using a local model. The received parameters need to be compared to their past values in the local model. The current time local parameters need to be derived from received parameters that were from an earlier timestep.

If the positions are different, then the new position is derived using the velocity $\dot{x}_\mathrm{c}$ received from the controlled side. The algorithm can be improved by adding a filter to only fix when the error is larger than a threshold, or by having a better way to derive the current position. A more accurate derivation might be possible by also sending the second order derivative, acceleration, from the controlled side. The first and second order derivatives can also be averaged with their counterparts at that timestamp at the master side to get a more rounded result.

$$x_\mathrm{m}^t = \begin{cases} x_\mathrm{c}^{t-\Delta t} + \dot{x}_\mathrm{c} \times (t - \Delta t) & \text{if } x_\mathrm{m}^{t-\Delta t} \neq x_\mathrm{c}^{t-\Delta t} \\ x_\mathrm{m}^t & \text{otherwise} \end{cases} \tag{3.1}$$

In cases where storing past values of model parameters becomes too expensive, the constantly updated average rate of change of the parameter over a preset number of past time steps can be used to derive the present value of the received parameter which can then be used, with the help of a filter that accounts for error, to detect and correct model divergence. Neither of these proposed methods for parameter derivation with delay are implemented in this thesis and further research is required to determine their performance.

3.2. Tactile Physics

Since the main challenge of MMT is the divergence of the master-side model from the slave-side environment, the quality of the model at the master side is critical. The required frequency and efficiency of model updates can be relaxed with better models that predict the slave side environment more accurately for longer periods of time.

3.2.1. Existing Physics Engines

Both virtual physics, as well as 3D rendering and computer graphics, advanced tremendously over the past years. Commercially available game engines such as Unity or Unreal Engine provide developers with every tool necessary to create high-end games or simulations, such as state-of-the-art GPU

accelerated 3D rendering or highly realistic physics. However, despite all the recent developments in this area, these general-purpose physics engines fall short when it comes to tactile applications. The reason for this is that these physics engines have fundamentally different goals and constraints. Tactile physics demand precision in terms of the timing, position, direction, and intensity of the forces applied to objects, meanwhile mainstream physics engines focus on quickly and efficiently resolving collisions for as many objects as possible. Furthermore, the physics engine needs to be fast enough to not be a bottleneck for the 1ms challenge which is a requirement that not all physics engines achieve.

In an application where a haptic controller device is interacting with a virtual environment, the physics engine is responsible for making sure that the position of the controller is correct at each loop of the physics engine, and that both the force applied by the controller on objects as well as the feedback force applied to the controller are calculated correctly. It also needs to have a way for the controller to communicate how much force should be applied to an object because often the only available controller data is the position of the controller. None of these functionalities are available by default in non-tactile physics engines as it is not their focus. A traditional physics engine tends to handle collisions by detecting constraint violations when two volumes intersect, and resolve the violation by pushing objects out of each other in the most efficient direction. While this is a computationally efficient algorithm that results in a physically realistic behavior of objects, the actual forces applied to the virtual object representing the haptic controller can be wrong or jittery. Also, this method of collision resolution is concerned with the deepest point of penetration rather than the first point of contact, which makes it unsuitable for tactile purposes.

Tactile physics engines and simulations that are commercially available and are open source are few, provided that haptic simulations is an area of ongoing research. Chai3D is one of the most popular of these and has been used in several research projects, but it isn't being maintained since 2016. Perhaps more importantly, Chai3D isn't a physics engine on its own but instead a haptic rendering framework that focuses on force interaction with surfaces and textures. When a dynamic environment is needed, Chai3D is used with another physics engine because it does not natively support simulating a dynamic environment. Because of this, creating a simulation environment or making changes to an existing one requires changes in the code can be time-consuming compared to a commercially available game engine such as Unity. One needs to initialize objects in code and also determine interactions between chai3d and the physics engine in the code such as applying force to objects when touched.

The options for this project were either adding missing functionality to a tactile framework like Chai3d or picking a better-maintained game engine with most of the required functionality and adding the missing tactile components to it. The latter was chosen mainly for easier development and the opportunity to better understand tactile physics while reverse engineering it. Particularly in this project Unity is used for its clear documentation, well-maintained code base, and native functionality that allows easy scene creation and editing through its editor. However, because Unity isn't generally fast enough to accommodate the 1ms challenge, Bullet Physics is imported to Unity. Unity is used to edit the scene and to render the simulation, but physics is calculated by a separate thread running Bullet Physics with which Unity communicates. Bullet Physics is chosen because it is well-maintained, open-source, and fast. Tactile functionality is added to Bullet Physics code base by reverse engineering Chai3D. Doing this allowed us to not only understand the inner workings of Chai3D better but also mainly to create a Bullet-With-Tactile-Functionality library for Unity that allows fast and easy creation of environments in Unity for TI purposes.

3.2.2. Proxy Algorithm

As mentioned, there are precision requirements of a tactile framework in terms of the position, direction, and intensity of the forces involving the tactile device, that are not solved by traditional physics engines. Substantial work was put into reverse engineering algorithms used by Chai3D to create realistic haptic renderings of virtual objects. The proxy algorithm is the core tool used by Chai3D to create such a realistic experience.

The algorithm takes its name from the proxy object created in the virtual environment to be controlled by the haptic device. It is in essence an idea that solves the challenge of communicating how much force to apply to an object in the virtual environment using a haptic device that provides only positional data. In the lack of collisions, the proxy mimics the position of the controller perfectly. However, when there is a collision, the haptic device is allowed to move past the point of collision, through the object. In the meantime, the proxy object respects the laws of physics and does not go through the object.

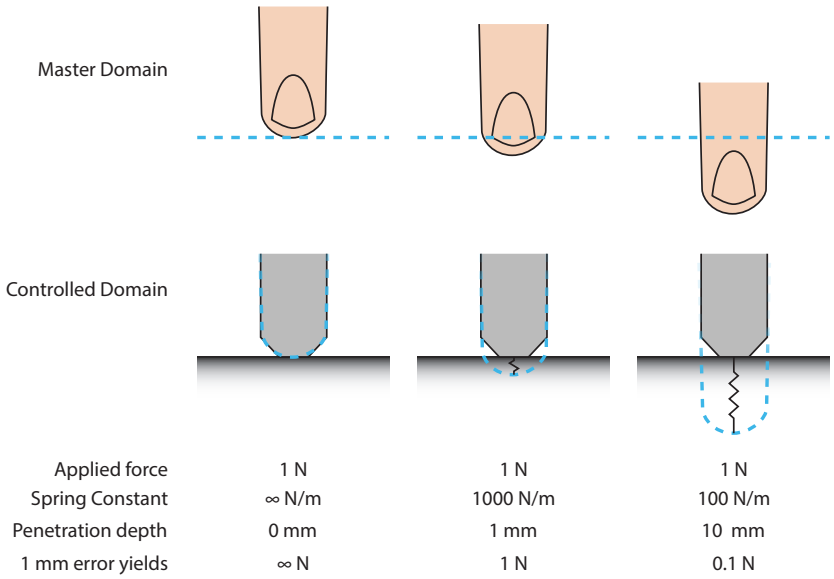


Figure 3.5: Interaction between a finger and a surface in a TI application: As the operator pushes down into the virtual surface (blue dashed line), the robot presses down on the real surface. An imaginary spring is drawn to the target position to calculate the force applied on the surface. This figure is taken from Kroep et al. with the permission of the authors. [2]

The distance between the controller and its proxy determines how much force is applied to the colliding object (and how much reactionary force is applied back to the controller). In other words, a spring is drawn between the controller and its proxy to determine the applied force. The spring constant corresponds to the rigidity of the surface. This part of the algorithm can be said to correspond to C' in the MMT control flow diagram depicted in Figure 3.1. While this figure is simplified and only contains a C' as an imitation of C at the slave side, in truth the master side also has a C' to deduct what force to apply on the objects in the local model. C' at both sides can in theory use the same algorithm but as discussed in the Model Mediated Teleoperation section, it is imperative to implement smarter C' algorithms at the slave side to help reduce divergence between G and G'.

Figure 3.5 illustrates how the proxy algorithm determines the applied force and how different spring constants yield different results. With a higher spring constant, feedback is arguably more realistic because the user feels a rigid surface with little penetration. However higher spring constants also cause a high amount of error in the feedback force if the position of the finger at the controlled domain or the surface at the master domain are wrong by some amount.

The applied force as a function of the positions of the pointers and the spring constant can be expressed as follows:

$$F_{\text{applied}} = (x_{\text{dev}} - x_{\text{phy}})k_f. \tag{3.2}$$

In theory, C' can be any function that uses the position of the device to determine the applied force. The formula can be extended for example by smoothing the effect of the position change on the force, with a formula such as:

$$F_{\text{applied}} = \frac{1}{2}F_{\text{applied}} + \frac{1}{2}(x_{\text{dev}} - x_{\text{phy}})k_f. \tag{3.3}$$

In addition to calculating the force, the algorithm requires finding the exact point of contact in the path from the proxy towards the controller, as well as sliding the proxy on the surface of the colliding object if the tangent force is stronger than static friction. The fact that there can be additional collisions during this sliding motion makes the algorithm even more challenging. The main 3 steps of the algorithm are to check for collisions in the path from proxy to controller, move the proxy on that path until the point of contact if a collision exists, and finally if there is a collision, slide the proxy along the normal of the colliding surface towards the projection of the controller's position on the normal plane. Figure 3.6 shows these steps in action. Since an additional collision can happen during the last step, Chai3D

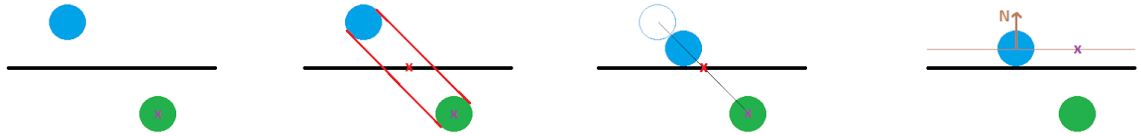


Figure 3.6: A single pass of the proxy algorithm illustrated. The blue ball represents the proxy pointer while the green ball represents the HIP of the controller. HIP is allowed to penetrate through objects to indicate force applied on the object. In this example the two pointers are separated by a wall because the HIP penetrated through the wall. The algorithm finds if there are objects on the path from the proxy to the HIP as shown in the second image. Then the proxy is moved towards HIP until the first contact with a surface (if any) as shown in the third image. Lastly in the fourth image, the goal position (shown as purple X) for the next pass is moved from the center of HIP to its projection on the surface + the radius of the proxy. The feedback force is calculated as orthogonal to the collided surface, and its magnitude determined by an artificial spring between the proxy and the HIP.

repeats steps 1 and 2 during sliding. In the worst-case scenario, the algorithm is run 3 times per physics step:

1. Moving proxy towards the controller until contact.
2. Moving proxy towards the controller's projection on the normal plane of the collision until contact.
3. Moving proxy towards the controller's projection on the intersection of normal planes of the first two collisions (which is a line), until contact.

3.2.3. Finding The Point Of Contact With Ray Casting

The most difficult part of moving the proxy pointer is finding the exact point at which the proxy will collide with another object in a single physics step, if at all. In essence, it aims to find if a volume traversing a path will intersect with another volume, and if so what will be the first point of intersection. This is a highly specialized feature for tactile applications and there is no trivial solution to it within a non-tactile physics engine framework. While it is possible to find whether or not there will be a collision, by utilizing the engine's built-in collision detection with a cylinder between two points (assuming the controller's Haptic Interaction Point is a sphere), it is not possible to find the first point of intersection along this movement using the built-in collision detection. This is because as previously discussed, a traditional physics engine would be considered with the deepest penetration point of the cylinder through the object (or vice versa) which is often a different point from the first point of contact of the sphere along the path of the cylinder.

The way Chai3D solves this problem is similar to ray casting used for high-quality graphical rendering of light and shadows in games and animations. The purpose of the algorithm is first and foremost to find the exact point of touch between the tactile controller and any object in the virtual world. This is similar in some sense to finding the exact points on objects where each ray of light touches which is a key component of ray casting. The key difference is that in our case we find the point of touch of two 3d objects which, unlike in ray casting, is not necessarily on a straight line from one center to another.

Each object in the virtual environment consists of a mesh of triangles, and the ray casting method is capable of finding the exact point on the exact triangle of the mesh that intersects with a given line. The geometric algebra required for this task is fairly simple and ray casting algorithms have additional acceleration structures to limit the number of objects and their triangles that are checked for intersections which will be further explained in the next subsection. However, in the virtual environment, the haptic controller is often represented as a sphere or an even more complicated shape, rather than a point. While finding the nearest collisions of a point on a path is as easy as finding the intersection of a line, doing the same for a sphere is significantly more complicated. To do so one would have to find the areas of intersection between a cylinder along the path and virtual objects, then somehow figure out which point in that area will be the first point of intersection for a sphere moving along that path.

To tackle this task in an easy and computationally fast manner, Chai3D inflates mesh triangles by the radius of the proxy and finds the intersection of the path with the inflated triangle. Truly, finding the intersection point with a sphere and another object is identical to finding the intersection point of the sphere's center with that object inflated at each point by the sphere's radius. Likewise finding the first intersection point of a moving sphere is the same as finding the intersection of its path with the inflated object.

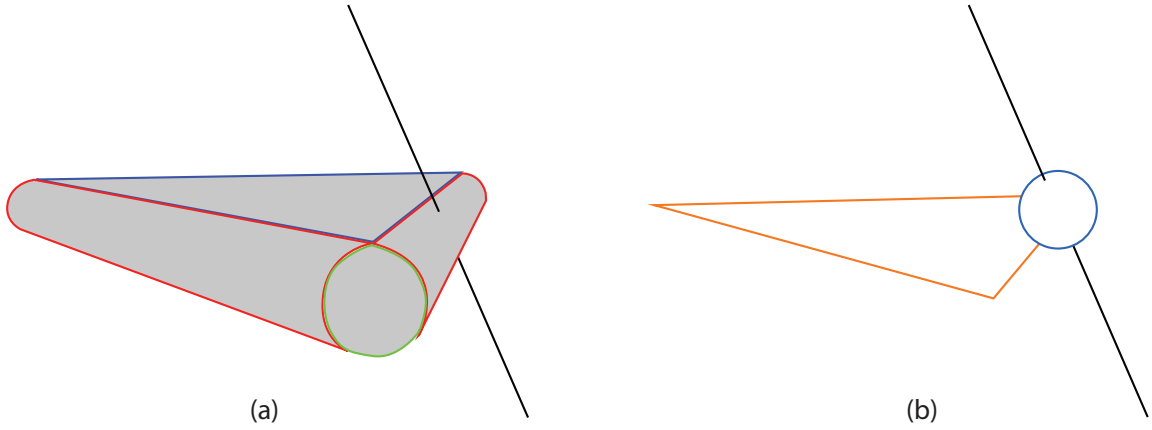


Figure 3.7: Finding the point of contact of the proxy sphere on a single mesh triangle. The triangle is inflated on all sides by the radius of the proxy sphere as shown in (a). Among the intersection points of this pizza slice with the path of the proxy, the one closest to the current position of the proxy gives the new origin position of the proxy along that path. Image (b) shows where the proxy sphere would end up on the mesh triangle after running the algorithm.

Figure 3.7 illustrates how this algorithm works. Given the proxy's radius r, a mesh triangle is inflated by creating 3 spheres centered at the corners with radius r, 3 cylinders along the sides with radius r, and 2 triangles with the same size and orientation as the mesh triangles traversed by r along the normal towards both directions. The combined shape draws the center of the tactile pointer if it were to slide all around the triangle. Among the intersection points of the path with mesh triangles, the closest one is the exact position of contact of the tactile pointer with the first object it collides with. Even more importantly the normal at this intersection on either of the spheres, cylinders, or triangles gives the exact direction of the feedback force the tactile controller would experience upon this contact.

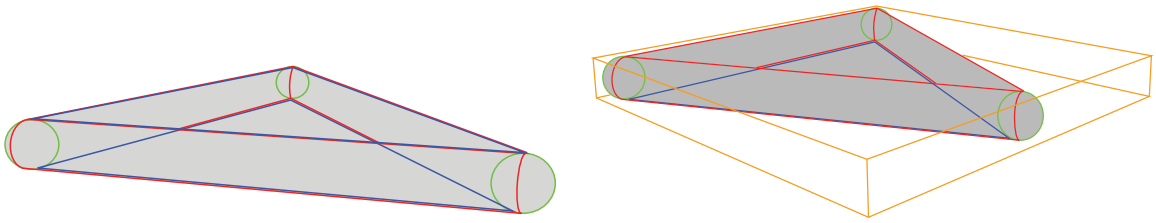
In theory to find the exact point of contact of an object with the proxy moving on a path, every triangle of the said object must be inflated by the proxy radius. However, in practice, several acceleration structures are used to ensure the program doesn't go through each triangle of a mesh to find the contact point.

3.2.4. Acceleration Structures For Contact Point Detection

Physics engines already have some techniques they use to accelerate collision detection which otherwise takes $O(n^2)$ time complexity because each object needs to be checked against every other object. These same methods are also used in ray casting for optimization. These optimizations are crucial for a tactile application like this project because of the 1kHz constraint.

Collision detection is usually divided into two parts: broad-phase and narrow-phase. During the broad-phase, collisions are detected quickly and inaccurately, such as by only checking collisions between bounding boxes of objects while ignoring any complicated shape. After eliminating objects that are obviously not colliding in this step, then a computationally more demanding narrow-phase collision detection algorithm is used to find the exact correct collisions.

There is a range of broad-phase algorithms that are optimal for different scenarios but both Chai3D and this project uses Bounding Volume Hierarchy (BVH) for broad-phase, which is overall a popular choice for both collision detection and ray tracing. BVH is a tree structure where the root is an axis aligned bounding box (AABB) that contains every object in the environment, and every node is the AABB of a single collision object. Each intermediary node is the AABB that contains its children. Using axis aligned boxes enables easy calculation of intermediary boxes and also quickly checking if two boxes intersect. The tree is traversed by comparing a node's AABB to the AABB of another object, to eliminate all those that have no collision. Because the boxes are aligned with the axes of the environment, they can simply be represented as a maximum and a minimum coordinate. Then to calculate the parent of two AABBs one can simply take the biggest maximum and the smallest minimum coordinates. Similarly, to see if two AABBs intersect, one simply needs to compare the minimum of one box with the maximum of other, and vice versa. Keep in mind that BVH can eliminate objects that have no collision,



(a) Inflated triangle for narrowphase

(b) Axis Aligned Bounding Box (AABB) of triangle for broadphase

Figure 3.8: The broadphase and narrowphase bounds used to find the exact point of collision on a triangle mesh. Image (a) shows the narrowphase bounds for a single mesh triangle. To find the exact point where the HIP will stop when it collides with the triangle, the triangle is inflated by the radius of HIP. The intersection point of the HIP's trajectory with these bounds gives the exact point where the HIP will stop after hitting the object. For simple calculation, the inflated triangle is represented by 3 spheres (shown in green), 3 cylinders (shown in red), and 2 triangles (shown in blue). Image (b) shows the broadphase bounding box around the triangle. The narrowphase detection is only calculated if the ray intersects this axis aligned bounding box. The bounding box is aligned with the axes of the environment so does not necessarily have the proportions depicted in this figure.

but it can still have false positives because the AABB of an object has a greater volume than the object itself.

The BVH tree is usually a binary tree, which is also the case in this project. The optimal depth of the tree i.e. how many objects are represented by the leaf nodes can be different for different applications. Either more time is spent traversing the tree and less time on narrow-phase, or vice-versa. In our case, because we have a very computation heavy narrow-phase where a single triangle is inflated by using 8 objects and each of them is checked for intersections with a line, we opt for an unconventionally deep tree where each leaf is a triangle of an object. In practice however this project takes advantage of the fact that the Bullet Physics engine already handles the broad-phase up to each object, we only extend it by creating a BVH tree for each mesh object.

In our use-case, during the broadphase among environment objects, the BVH tree is traversed from the root towards the leaves, where at each node we check if the AABB of that node intersects with the AABB of a cylinder along the path of the proxy. The cylinder has the same radius as the proxy and represents the entire volume in space where the proxy sphere passes through along the given path. This cylinder is a ghost object which does not cause collision response with other objects and goes through them, but still records when it collides or intersects with other objects.

If the broad-phase pass of the Bullet Physics engine finds colliding objects along the path of the proxy, then among all the colliding objects the first point of contact along the proxy's path needs to be found. While it might seem enough here to just select the object that is closest to the starting point of the path, this is not the case. An extremity of an object can be closer to the starting point of the proxy, despite having its center further away than another object. The proxy's first point of contact will be with the closest extremity of any object in the environment, regardless of their origin positions. For this reason, the BVH tree of each object's mesh is checked if the bounding box of the object shows and intersection with the ghost cylinder.

Once the triangles of interest are found through BVH, then the most computationally expensive part begins which is to find the exact point of intersection between the triangle and the tactile pointer over a path. As mentioned in the previous subsection, this computation is simplified by inflating the triangle by the radius of the tactile pointer and finding its intersection with the path line. This particular approach is evidently the easiest if the tactile pointer is a sphere, otherwise, the rotation of the pointer along the path would require inflating the triangle in a non-consistent way. Figure 3.8 shows the collision bounds of mesh triangles at broad-phase and narrow-phase steps.

4

Implementation & Methodology

4.1. Hardware Setup

Figure 4.1: Hardware setup used for experiments and demos. The setup consists of a computer and one or two Novint Falcon devices. When a virtual environment is used, only one Novint Falcon is used. The computer can simulate the network or can connect to a second computer over a real network.

For this thesis, a Novint Falcon haptic device was chosen as both the controller and the controlled haptic device. Novint Falcon is a an easy to use, commercial haptic device that is capable of precise force output as well as precise positional readings. It is low-cost and a number of them were readily available at the Embedded and Networked Systems department at TU Delft, to be used for the purposes of this thesis. While there are other different haptic devices that can be used for this purpose, the price and availability of Novint Falcon made it a strong choice for this thesis. Using the same device at the master and controlled domains allows for perfect mimicking of movements and forces and easy implementation.

The library we use for I/O with the Novint Falcon is developed by the company Force Dimension. There are several haptic devices similar to Novint Falcon that are designed by this company. Our framework technically supports all of their devices because we use their SDK which is mainly developed for their own devices. Additionally the code is built in a way that would make adding support for other devices low hassle.

The setup consists a minimum of one computer powerful enough to pass the performance requirements of the application and two Novint Falcon devices. While the exact hardware requirements required to run the application is beyond the scope of this thesis, a 3.30 GHz 11th Gen Intel Core i7 processor and an Nvidia GeForce RTX 3060 graphics card were enough to run the application smoothly.

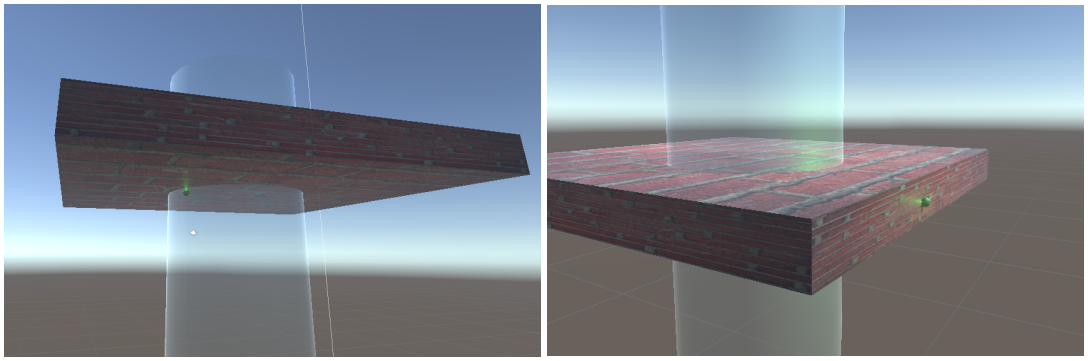


Figure 4.2: The collision points found by Bullet Physics when the ghost cylinder collides with an object. The transparent cylinder is the ghost cylinder and the green point shows the found point of collision. Instead of finding the first point of contact along the trajectory of the cylinder, this method finds the most efficient point to apply force on to resolve the collision by pushing objects out of each other. As the first image shows, the found point can be uncentered which would cause the proxy to diverge from its path, and it can be at the wrong side of the object which would cause the proxy to tunnel through object. As the second image shows there can even be cases where the collision point isn't on the cylinder at all, such as when the object is in the middle of the cylinder and it's more efficient to push the object out sideways than to push along the length of the cylinder.

The network can be simulated on one computer if both Novint Falcons are connected to it, or a real network can be used between two computers each with their own Novint Falcon connected to them. If the environment at the slave side is real rather than virtual, also a point-cloud device such as Xbox Kinect is required.

4.2. Making The Physics Engine Work

At the beginning of my thesis, the idea of MMT was not there yet. Phu Nyugens, a masters student before me, had worked on importing the functionality of Chai3D to Unity during his thesis for the purpose of creating a testbed for TI applications that had the haptic render features of Chai3d but also the benefits that come with Unity such as an Editor with an easy GUI that allows fast creation and modification of different scenes and the objects within them. This testbed which then became the local model for our MMT design uses Unity with C# code for creating and displaying scenes but C++ running on a separate thread which uses Bullet Physics engine to handle all the physics, the spring algorithm for haptic feedback, and libnifalcon (an open source Novint Falcon I/O and kinematics library) to get the position from and apply forces on a Novint Falcon.

Our general approach for design was to discover a problem with the program through testing, then to check if the problem also exists in Chai3d, and if Chai3d seems to lack the problem try to reverse engineer their code and reimplement it. This approach was established after theorizing about how to solve a particular bug, discovering new problems along the way, then checking Chai3d after being stuck. Once we understood the solution of Chai3d we realized that Chai3d often has an elegant solution to challenges of haptic rendering, and that attempting to blindly solve the problem is often a waste of time until we understand the requirements of tactile physics engines.

One of the first bugs we discovered that had to fixed,the proxy sphere behaving weirdly in some cases. Namely the proxy sphere seemed to vibrate occasionally, was lagging behind when a collision was happening even on surfaces with no friction, and the force feedback was acting drastically wrong at times. It was quickly discovered that an earlier solution implemented to prevent the proxy from going through thin objects was at the source of these bugs. At high speeds, the proxy was able to pass through thin objects because the proxy could easily penetrate past the middle section of the object causing the bullet physics engine to push the object towards the wrong side, or to pass through the object without even a collision being detected. The solution implemented was to limit the speed of the proxy if it had a collision in the last physics step, which among not addressing the occasional problem of the collision not being detected at all, also introduced artificial delay in the system by making the proxy lag behind. This artificial delay was causing the proxy to lag behind on all surfaces as if it had friction, which was then causing a secondary collision to have an artificially high force spike which common in delayed TI applications.

After contemplating several solutions to this problem, as we already knew that Chai3D had a well

Figure 4.3: Collision point along a trajectory found in Chai3d. Unlike the collision points found by traditional physics engines, Chai3d finds the exact position where the center of the proxy will end up on a given trajectory.

working haptic rendering algorithm without these bugs, the decision was made to tackle the daunting task of going through the poorly documented Chai3D code and reverse engineering it. It was discovered that Chai3D doesn't use the traditional physics engine method of allowing objects to penetrate first and then to push them out. Instead, the first point of contact on the path from the proxy to the HIP, and the proxy is moved to the point of contact without actually penetrating the object. The proxy then slides on the surface towards the HIP's projection on the surface, which is done by repeating the algorithm with but towards HIP's projection on the surface instead of the HIP itself. Repeating the algorithm for the new goal point on the surface allows detecting additional collisions that can happen while sliding. While the Theory section of this thesis paper talks more in depth about how this algorithm works, it is important to note here that at first it wasn't obvious to us how this first point of contact was found. Our first attempt involved creating a ghost cylinder between proxy and HIP, with the radius of the proxy, that recorded all collisions with objects in the environment without actually applying or being applied any force by these collisions. This method proved unusable because when the cylinder went through an object, none of the collision points found by Bullet Physics were the point of interest for our algorithm.

This finding forced us to delve deeper into Chai3D code to understand how they find the first point of contact along a path, which in return made us realize that traditional physics engines are unsuited for many haptic rendering algorithms that are key to a TI testbed or MMT local model. As explained in depth in the Theory section, Chai3D uses a method similar to raycasting to find the contact point. The algorithm takes the AABB tree algorithm that is traditionally used for broadphase collision detection amongst objects, and applies is on a lower level down to each triangle. This was discovered during reverse engineering the Chai3D code base, when printing the AABB tree nodes of a cube object yielded 12 leaf nodes rather than just 1 for the cube object. A cube has 6 sides each made out of 2 triangles, which showed that the leaves of the AABB tree were in fact triangles of meshes. This broadphase step allows fast elimination of far away triangles whose bounding boxes do not intersect with the path ray of the proxy. Then a more costly narrowphase detection checks the exact point of contact on each triangle (if any) and chooses the one closest to the proxy. Both the AABB of each triangle and the triangles themselves are inflated by the radius of the proxy sphere. This is an ingenious method which is further explained in the Theory section allows using only the line of the proxy's path while also taking into account its radius, and considerably simplifies the algorithm. Upon discovering this method, Chai3D's algorithm was implemented with minimal changes to suit our code base.

Finally, considerable effort was spent on understanding the many epsilon values used in the Chai3d code and to adapt them to our code, because their default values initially did not work for the scale of our environment. The main epsilon value is used for stopping a certain distance above a contact point when moving the proxy. After the contact point is found with the previously discussed ray casting algorithm, we calculate how much to move back along the trajectory to be of epsilon distance away from the contact point's normal plane. We make sure this distance does not surpass the radius of the proxy. If the collision distance was smaller than epsilon, then we update the epsilon value to be equal to that distance. There are also other epsilon values with their own purposes, to summarize:

- **epsilon:** used for pushing away from contact point.
- **epsilon_base:** initial value of epsilon, also used as a margin to determine if the proxy reached the goal.

- **epsilon_min:** minimum value for epsilon when updating it.
- **epsilon_collision_detection:** used to extend the path segment past the goal, a sort of look ahead.

4.3. Enabling Mesh Objects

Upon finding out about Chai3D's algorithm to find the first point of contact along a path which utilizes meshes, enabling mesh objects in the simulation which was originally a stretch goal became a necessity. A simple data structure is chosen for the core mesh implementation, which simply involves:

- A Triangle object that has 3 vertices, the max AABB point and the min AABB point.
- CollisionTree object mostly copied from the Chai3D and adapted to our codebase. The tree is build from an array of Triangle objects.
- A TriangleMesh object which holds array of Triangles and their CollisionTree.

When the physics engine starts, first each object is created in the Bullet Physics environment, then a TriangleMesh is initialized for each interactable object in the environment, and their AABB trees are built. The TriangleMesh has a findClosestSegmentIntersection which first checks its AABB tree to find triangles whose AABB intersects with the given segment, then loops through them to find the exact point of intersection on each triangle (if any) and returns the closest one. Observe that this function does not involve the *broader* broadphase step that involves the objects in the environment rather than their triangles. This is because it was decided that the Bullet Physics engine is already capable of finding broadphase collisions between objects efficiently and since the engine needs to run anyways to handle non-haptic interactions between objects in the environment, we might use it to simplify the code without any performance drawbacks. At every time step, the Bullet Physics engine is ran and the objects that had a collision with the ghost cylinder are looped through to find the closest point of contact among them.

For creating the meshes at the beginning of the simulation, functions are made for simple native objects in Unity and Bullet such as cubes that turned them into meshes. Additionally a library of meshes is created where a folder held triangle mesh .obj files with specific names and if a mesh object was added in Unity that matched the name of the .obj file, the TriangleMesh object is created for it by reading said file. The mesh object is also added to the bullet environment to enable interaction with other objects in the environment. Our TriangleMesh object is a separate entity from the mesh object in the Bullet Environment so when finding the point of intersection, the relative position and orientation of the ray to the mesh is used. It is also important to note here that an exception was made for sphere objects where instead of triangulating them, a ray is cast on the sphere with its radius incremented by the radius of the proxy sphere. For objects that can be easily inflated by a radius on all sides such as sphere that can simply be scaled up, using a mesh is more complicated and time consuming.

4.4. Towards Model Mediated Teleoperation

4.4.1. Different Scenarios

Enabling meshes and fully implementing the proxy algorithm used in Chai3D produced a testbed that has the haptic rendering capability of Chai3D and the flexibility and functionality of Unity. While the original purpose was to use this as a simulation of the controlled environment for controlled testing of TI applications, we concurrently came up with the idea to use this model at the master-side to simulate the controlled side environment and bypass network delay for haptic feedback. At the time we were not aware of research on MMT and thought this was a completely novel idea that would solve TI forever. To change the direction of thesis towards implementing this idea, several scenarios were devised that would allow us to test the effectiveness of our local model assisted TI application against the traditional model. These scenarios are the following:

- **Scenario 1:** Single node without network. The controller directly interacts with the virtual environment. This allows us test and benchmark the physics engine in an isolated environment.
- **Scenario 2:** Traditional TI implementation with a virtual environment. The controller and physics engines of scenario 1 are separated into two different nodes that communicate via a network.

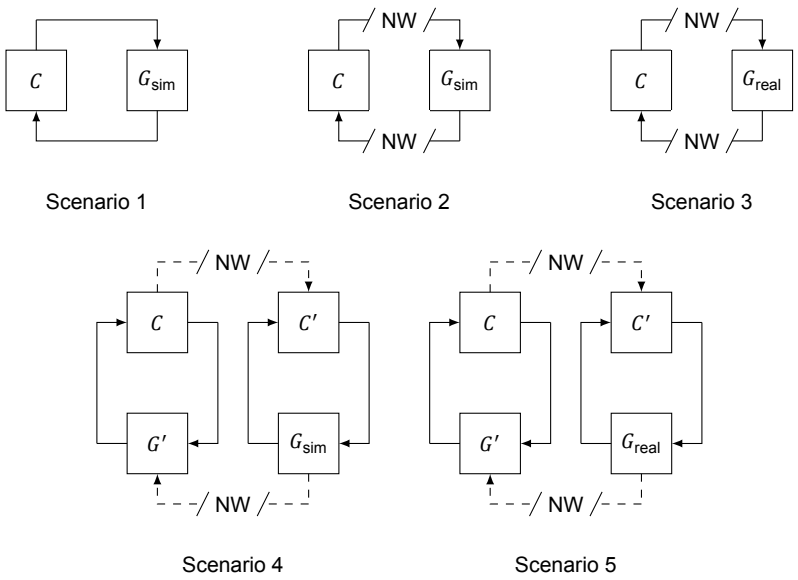


Figure 4.4: Control system flowcharts for all 5 scenarios.

This vanilla tactile internet implementation vith a virtual environment allows us to test the network and observe the effects of network delay, while the controlled node being simulated ensures that the results are deterministic.

- **Scenario 3:** Traditional TI implementation with a real environment. This is the same as scenario 2 except the controlled side is no longer a simulation. Another Novint Falcon device is used for the controlled side robot. This scenario serves as a benchmark for the vanilla tactile internet implementation in the real world and enables anecdotal experiments.

- **Scenario 4:** MMT design with a virtual environment. An enhanced controller with an internal virtual physics engine is implemented. This design aims to lower the perceived latency and relax the network delay requirements which is the core point of this thesis. This scenario allows for controlled tests and benchmarks of our MMT implementation.

- **Scenario 5:** MMT design with a real environment. The controlled node with a local physics engine communicates with a robot in the real world. This scenario provides the final product and allows qualitative analysis of our MMT implementation.

Note that Scenario 4 and 5 are in fact implementations of MMT where the local model is a physics engine, except we did not know about the existence of MMT research until after starting to implement these scenarios. These scenarios gave an incentive to separate the codebase into modules, some running on their separate threads. Among other benefits of a modular design such as having a readable code base consisting of reusable blocks, it allows easily setting up several different scenarios for different testing purposes.

4.4.2. Modular Software Design

The starting point of the software is always the Unity engine which initializes all the threads and modules of the tactile application with desired parameters depending on the scenario and whether or not the node is master or slave. It then communicates with these threads both to set up the virtual environment if needed and to display the environment to the user. The software consists of at most 3 main threads for 4 modules (2 of the modules share a thread). While more threads can be created per each of these 3 main parts to increase performance, the separation between these 3 parts is always apparent. Figure 4.5 shows how these modules and threads interact with each other in Scenario 5. In accordance with MMT, regardless of network conditions there is always a complete control loop between the Novint Falcon (NF), Controller Module and Physics Module.

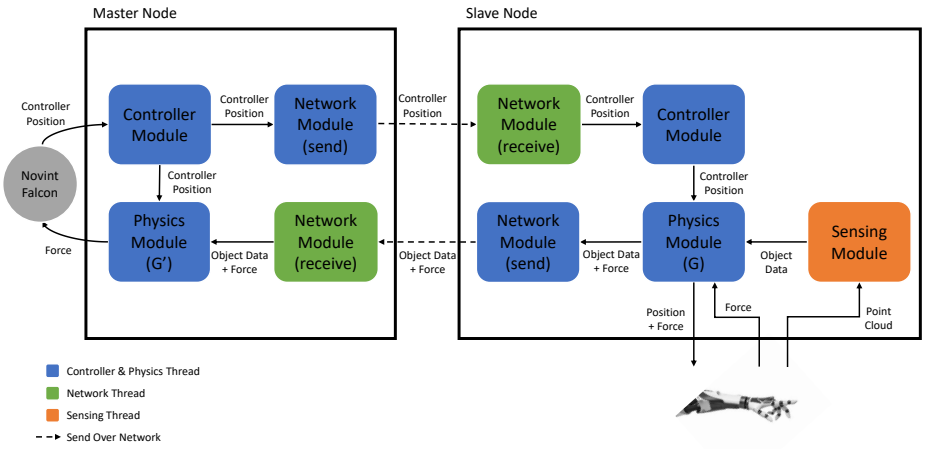


Figure 4.5: Modules of 2 nodes communicating

The first of the 3 main threads is the physics & controller thread which consists of the physics module and the controller module. The controller module's task is to provide the position of the haptic controller, from the haptic controller itself or from packets over network. The physics module is tasked with using the input from the controller module to move the controlled robot either in real world or in the simulated world, and with running the physics engine in the latter case. The controller and physics modules run on the same thread because they can run synchronously one after another as the controller's data is immediately used by the physics module. The network module is responsible for sending data over network when needed and receiving data on a separate thread. While the receiving functionality of the network module is done in its own asynchronous thread, sending happens in the physics & controller thread. The data in question consists of positions of objects of interest in the environment, the position of the robot, and the feedback force experienced by the robot. In the case of the real environment, technically also a video stream from a camera feed would be included but that is beyond the scope of this thesis. Finally, the sensing module is used to get the necessary data from the controlled environment, directly from the physics engine in the case of a simulated environment, or with the help of sensors in the case of a real world environment. When sensors are used, more threads are used by this module to keep up with the performance requirements of the application.

4.4.3. Controller Module

The controller module is tasked with handling the I/O and kinematics of the controller, which is in our case a Novint Falcon haptic device. Initially, an open-source library called libnifalcon was being used for this purpose, along with forward kinematics and inverse kinematics code copied from one of the examples in the libnifalcon codebase. The library allowed reading pure encoder positions values of the device and apply forces by setting motor voltages. While this library allowed low level access to the Novint Falcon device it required additional kinematics calculations and complex pieces of code for simple actions.

Towards the end of this thesis we realized that the feedback force was occasionally in the wrong direction, and the difference was wrong enough to be noticed which put the quality of the final demo into danger. After debugging and verifying that the force calculation from the physics module was done correctly, the only possibilities were either a problem with the device, or a problem with our kinematics code. This was further proven by applying a constant upward force and observing that the force was wrong at certain positions of the device, noticeably around the limits of its moving space. We tried with a different Novint Falcon and saw that the same problem remained so the kinematics was certainly the culprit. We did not want to repeat our past mistakes by trying to solve on our own something that works seamlessly in Chai3D, we decided to use Chai3D's code instead of fixing our kinematics code. During this we discovered that Chai3D was using a closed source SDK library for controlling haptic devices.

The SDK used by Chai3D for this purpose is called DHD. Despite searching rigorously we could

not find what DHD stands for although we assume that the last two letters stand for ”Haptic Device”. This SDK is developed by Force Dimensions which is also the company that developed Chai3D, on top of developing several other haptic devices similar to Novint Falcon. The library supports all haptic devices made by Force Dimensions and the Novint Falcon. While it is closed source, the functions provided by the SDK allow both high-level and low-level control of the devices. It is an incredibly easy to use library that handles by default practically everything we used to have in our code with libnifalcon. In the latest version of this project, all uses of libnifalcon are replaced with DHD which minimized the code immensely while also solving the feedback force bug.

While this closed source SDK that supports a number of different devices makes up the bulk of the controller module, the code is actually set up in a way that allows easy implementation of controls for any new device. An abstract class HapticDevice has a child class called NovintFalcon which implements all of HapticDevice's virtual functions and calls the necessary DHD functions in them. Any new device type can be added by making another extended class of HapticDevice and implementing its virtual functions.

4.4.4. Physics Module

The physics module needs to keep track of object positions through its physics engine but also needs to update them with incoming data if needed. The Bullet Physics engine is used for interactions between objects other than the haptic device, and also for broad-phase collision detection for the proxy algorithm, meanwhile implementations that are mostly adopted from Chai3d allow instant haptic rendering using the proxy algorithm. To allow instant haptic feedback at master-side, each callback loop of the physics engine needs to run in 1ms or less. We keep the average execution time at 1ms by checking at the end of each loop the time passed since the end of the last loop and waiting until it is 1ms. If for some reason a loop or a number of consecutive loops take longer than 1ms, the following loops wait less than 1ms or don't wait at all until the average execution time gets back to 1ms. This approach was chosen because the initial implementation, which either waited until 1ms or continued to the next loop cycle, caused the delays to accumulate and made the program lag behind.

4.4.5. Network Module

Before the idea of MMT and a local model, the idea was to just have a TI testbed and so the scenarios 4 and 5 did not exist. So the initial idea for the network idea was to simply send NF positions from the master side, and feedback force from the controlled side. Having a local model that needs periodic updating required a slightly more complicated network module. Depending on the scenario and the node type, the network module needs to handle sending and receiving packets for controller position, applied or feedback force, and environment data to update the local model. The network module has 2 components for sending and receiving. Sending data is always synchronous with the controller and physics thread and therefore is done in that same thread, however receiving has to be done asynchronously so a separate thread is needed for receiving. The ports to listen to and to send packets to are specified when starting the unity application, at the scenario selection step.

All packets are UDP packets due to real-time nature of the application. It might be argued that security might be a concern for certain TI applications but since this thesis isn't concerned the networking aspect of TI, we decided that UDP packets would suffice. For our purposes establishing a secure connection and error checking is too time consuming. However, for some critical data such as initializing a new object, the receiver has to send back an ack to ensure the packet was received correctly. For these critical packets, the sender waits a set amount of time for an Ack, and if it is not received sends the critical packet again. The Ack packet contains the hash of the contents of the received packet which is used by the sender to ensure the packet was received without error.

The network communication between nodes serves 2 purposes: discovery of objects and maintenance of objects. Discovery of the object and its initialization in the master side model is slower than maintaining an objects position and other properties. The reason for this is that even with a library of objects that ensures the collision tree for that object is pre-calculated, new memory needs to be allocated for that object and it needs to be instantiated in the Bullet Physics library for broad-phase collision detection and collisions between non-haptic objects. Furthermore, discovery of objects is much more critical than maintaining of objects because in order to reduce the network load, object discovery packets are only sent once. In contrast, an object on the move will constantly receive data to update its positional and other parameters so missing a packet has minimal effect. This separation is the prime

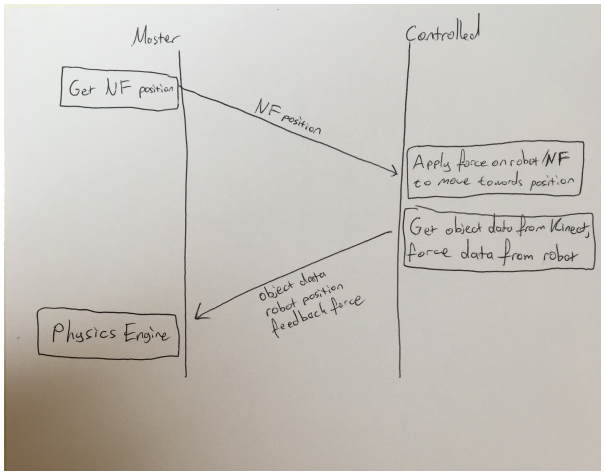


Figure 4.6: Network flow

reason for the bilateral UDP packet implementation for critical packets.

4.4.6. Sensing Module

The sensing module is tasked with detecting the objects in the environment and to provide those objects' properties to other modules. Only the slave node needs a sensing module since that is where the controlled device interacting with the environment is.

A point-cloud input device, in our case Kinect, scans the environment and extracts a point cloud. Then the separate objects are detected and classified with an object detection and classification Machine Learning (ML) algorithm. The module determines the relative position and orientation of the objects and provides them to other modules. While in the future estimating properties of objects such as mass or friction will also be necessary, this is outside the focus of this thesis. For the purposes of this thesis, it is assumed that the masses provided in the library of objects are always correct.

Since object discovery is not the focus of this paper, this part of the project was largely covered by the thesis of another fellow TU Delft student Kilian. His thesis delves deeper into the topic of object discovery using point clouds.

4.4.7. Thread Safety

As mentioned, the program is multi-threaded and share data with each other. Some examples to this are the Unity thread reading position data from the physics thread or the physics thread reading received data from the network thread. To keep these read and write operations thread-safe, a number of mutexes are used. While atomic variables in C++ provide better performance compared to mutexes, using mutexes is a necessity when thread-safe writing to multiple data addresses are needed. Since most of the data involved in our case are vectors that hold positions or forces, the only way to update them or read them safely is with a mutex. If one thread reads a vector while another is actively changing its value, even if each element of the vector is atomic, the reading thread can read the vector when only one or two of the vector's 3 elements are updated. Depending on the context, this can cause the force feedback to be wrong or cause objects to get wrong updated positions causing the model to diverge significantly.

When a thread attempts to lock a mutex when it is already locked by another thread, its execution holds until the other thread unlocks it. To minimize the time spent waiting by each thread and to reduce overhead, mutexes are only used before and after critical sections of the code where a variable used by multiple threads is read or updated. In our case these consist of variables received over the network, and variables sent to or received from the Unity thread to display the scene and other information on Unity or to get object data from Unity to create instantiate in the Physics thread.

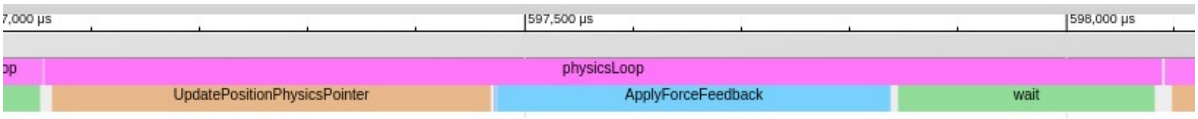


Figure 4.7: Benchmark visualization example with Scenario 1. Zoomed in to show one loop of the physics thread.

4.5. Logging and Benchmark

To benchmark the performance of the program in different scenarios and conditions as well as to optimize the code, logging the execution time of different components is needed. In early stages of development prints to the console were used for both debugging and to benchmark when needed. Additionally the execution time of the physics thread was sent to Unity to be displayed on the screen. Then logging to a file was implemented which simply opened a CSV file and dumped timestamps along with other data of interest such as the number of collisions. Finally, a piece of code by a youtuber called TheCherno that allowed easy benchmarking of separate functions.

The benchmark code writes the start and duration of every loop iteration of the callback function of the physics thread, as well as the start and duration of every function called in the loop. Thanks to the modulated software design that separated the code into neat functions, this helps tremendously in figuring out which parts of the code present bottlenecks and which functions are affected the most by certain scenarios. For visualizations, Google Chrome's Trace Event Profiling Tool is used. The benchmark logs are put into a JSON file that can be parsed automatically by this tool. The tool creates a timeline of function calls and can also visualize concurrent threads. Figure 4.7 shows a zoomed in example of this visualization tool for a single physics thread execution in Scenario 1.

5 Results

5.1. Experimental Setup

In this chapter, the different scenarios and conditions that are discussed in previous chapters are experimented with to put our claims to test and to gain new insights when possible. Firstly a timing analysis is made to ensure the program itself is fast enough to accommodate the $1\,\mathrm{ms}$ challenge regardless of network. Different configurations are tested to examine the effects of different components on timing performance. Next, traditional and MMT designs are compared in a static environment to see how they affect haptic feedback differently in varying network conditions. Finally, the model divergence of MMT is examined in depth for dynamic environments and the efficacy of periodic model corrections is discussed.

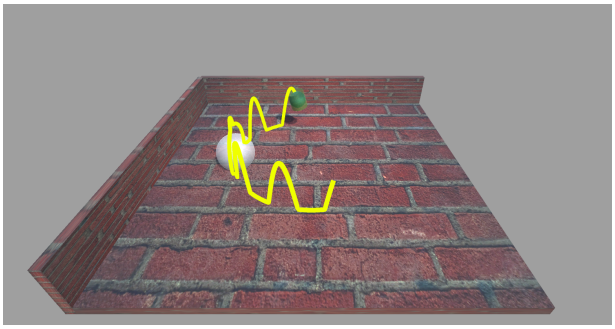
The way our application is designed allows quick and easy creation of test beds. Unity editor allows us to add or remove objects to the environment or to change object properties without effort. This allows us to change the environment used in the experiments according to our needs. Furthermore the program has different scenarios that represent different designs and conditions. These different scenarios allow us to use the same program for different TI designs such as with or without a master-side local model.

The 5 different scenarios can be summarized as:

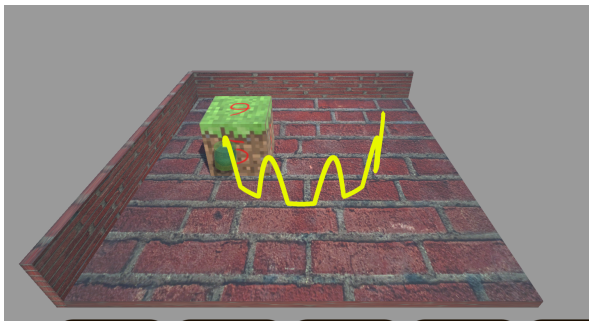
- **Scenario 1:** Single node physics simulation, no network
- **Scenario 2:** Traditional TI design without local model. Virtual environment.
- **Scenario 3:** Traditional TI design without local model. Real environment.
- **Scenario 4:** MMT design with local model. Virtual environment.
- **Scenario 5:** MMT design with local model. Real environment.

Out of these scenarios, we will investigate the performance of scenarios 1, 2 and 4. These scenarios involve virtual environment instead of a real environment and a real haptic actuator at the controlled side. Using a virtual environment makes it easy to create deterministic, repeatable experiments unlike a real environments which would introduce the risk of external interactions. Among these scenarios, scenario 1 is used solely to analyze the timing performance of the physics engine. Scenario 1 can be used for this purpose because G' or the Controller & Physics thread runs independent of network conditions, in terms of performance. Scenario 2 and 4 are used to represent the traditional TI application and model-mediated TI application respectively. For the rest of the chapter these scenarios 2 and 4 are referred to as "Traditional TI" and "MMT".

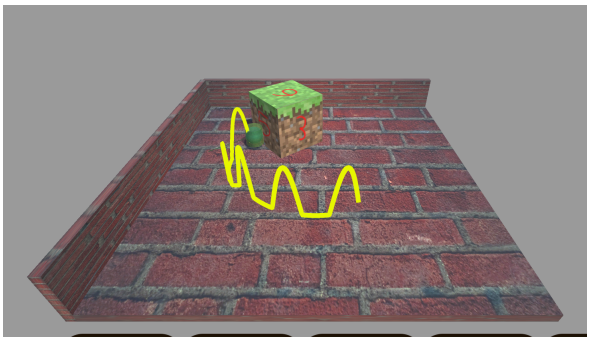
The first experiment of this chapter is timing performance analysis. he $1\,\mathrm{ms}$ challenge that is often framed as a network requirement, also introduces upper bounds for program execution times. What matters is the latency of the full round-trip delay from the movement of the controller to the haptic feedback. This means that all components of a TI application share the burden of the same latency constraint. For this reason we aim to measure the execution time of the Physics & Controller thread which has the main bulk of the program, and ensure that it achieves the necessary timing requirements.



(a) Static Environment



(b) Dynamic Environment (1)



(c) Dynamic Environment (2)

Figure 5.1: The setup of the virtual environment for testing model diverge. The circular & sinusoidal trajectory shown in yellow is used to examine the difference in feedback force between the master and controlled domains. The HIP makes a circle on the the X-Z plane that takes 10 seconds, while also following a sine wave trajectory on the Y axis. The static environment is shown in (a) while (b) and (c) show the dynamic environment in motion. In the static environment a stationary white sphere is put on the trajectory to make the test less artificial. In the dynamic environment the static white sphere is replaced with a movable box. Images (a) and (b) show the movement of the box over time as the HIP pushes it.

The timing performance is measured by logging the execution time of each loop of the Controller & Physics Thread, writing it to a CSV file. The thread tries to achieve a $1\,\mathrm{ms}$ execution time at each loop, and if the execution takes longer it makes up for the delay in the following loops by waiting less than $1\,\mathrm{ms}$. As such, an average of $1\,\mathrm{ms}$ execution time is achieved unless the thread is consistently late. A $1\,\mathrm{ms}$ thread period means that running the program for 15 seconds produces 15000 data points which don't give much useful information when directly plotted. For this reason a CDF graph is used to visualize the cumulative probability of the execution time being below a threshold. A python script is created to parse the CSV file which has a timestamp for every time the loop is ran. The timestamps can then be used to find the total execution time of each loop including time spent waiting. These execution times along the entire time the program was ran are then plotted into a CDF graph using the matplotlib library. We do this first for scenario 1 to analyze the thread being run without interacting with other threads. Then scenarios 2 and 4 are used to ensure both of our traditional TI and MMT designs run sufficiently fast.

After timing analysis, the strengths and weaknesses of the traditional TI design and the MMT design are compared. For this, we separate our experiments by static and dynamic environments. The static environment provides a simpler case where none of the objects are movable. By contrast the dynamic environment has one movable object that can be interacted with. In the static case the effect of network delay on traditional TI is compared with that on MMT. Additionally the error introduced by wrong local model parameters is explored. Then in a dynamic environment, the divergence caused by a movable object is examined and the efficacy of correcting the model is discussed.

In order to keep tests between different setups comparable, the same predetermined path is used in each test run. The environment is also kept the same among test runs with a few exceptions. Namely for testing MMT with wrong model parameters in a static environment, the ground surface is wrongly modeled as slanted by 10 degrees, and the sphere's location is moved so that the collision happens earlier. Additionally when testing the dynamic environment, the stationary sphere is replaced with a movable cube. Figure 5.1 shows the trajectory of the HIP in yellow. A sinusoidal movement in the Y

axis while moving around a circle on the X-Z plane was chosen to create a predetermined but sufficiently complicated trajectory in 3 axes. The trajectory goes through the ground which makes the proxy pointer slide along the surface as shown in the figure. This trajectory was programmed as a function that takes a timestamp and outputs a 3D position which enables different runs to follow the exact same trajectory despite each of their thread loops running in different times. For static environment tests stationary sphere in put in the path to introduce an additional source of feedback force different than the uni-directional reactionary force from the ground. In the dynamic environments, there is a movable box that the HIP pushes around to simulate a dynamic haptic interaction.

$$E_{\vec{F}} = ||\vec{F}_{\text{mas}} - \vec{F}_{\text{con}}|| \tag{5.1}$$

As a quantifiable measure of divergence between the model and the static environment, the feedback force is chosen. Both the force and error in force are graphed and examined Formula 5.1 shows how the error in force (written as E_{F}) is calculated. $\vec{F}_{\text{mas}}$ represents the master side force vector and $\vec{F}_{\text{con}}$ represents the controlled side force vector. To illustrate the difference in force, the magnitude of the difference of the two force vectors is found. This method accounts for differences in all 3 axes as opposed to directly comparing force magnitudes which would yield 0 error if the same amount of force is experienced in a wrong direction. For dynamic tests, the same measure is used but the position of the movable object is also plotted over time. For dynamic tests, the same measure is used but the position of the movable object is also plotted.

The data for force feedback and object position are logged in CSV files similar to the one used for timing performance analysis. The two nodes of the program are run simultaneously and the predetermined trajectory is activated with a button push. Each row of the CSV file has a time since epoch timestamp in microseconds, the 3D components of the feedback force at that instant, and the 3D position of the dynamic object at that instant if needed. A python file then plots the force or position over time using the matplotlib library. For plotting the force difference, two CSV files generated by the master and controlled nodes are read and the forces that were being experienced at the beginning of each millisecond are subtracted from each other and their magnitudes stored. These force error magnitudes are then plotted over time.

5.2. Timing Performance Analysis

In this section, the execution time of the Physics & Controller thread is analyzed to ensure it achieves the timing performance required for TI applications. We first check Scenario 1 which consists of a single node, a single thread, and no network connections. Then the execution times of traditional TI design and the MMT design are examined. In all experiments a virtual environment is used.

Figure 5.2 shows the CDF graph for execution times of Scenario 1, where 90% of thread executions ran in below $1.05\,\text{ms}$. The symmetry around 1ms is caused by the software attempting to get the average execution time fixed at 1ms and running less than 1ms if the average is above 1ms. This shows that in our program can read the position of the controller, simulate its movements and interactions in a virtual environment, calculate the resulting feedback force and apply said force to the device, all in a single thread and quick enough for the big majority of the time. The exceptional occasions where the thread is not fast enough, the program can always correct the delay by making future executions last shorter.

Inference 1: The physics engine and controller communication can achieve the necessary timing requirements in a single uninterrupted thread.

In a TI application the main thread can be interrupted when communicating with other threads, such as when asynchronously receiving data over the network. When multiple threads are involved, mutexes are used to ensure data isn't accessed by two parallel processes at the same time. If a thread attempts to unlock a locked mutex, it halts until the mutex is unlocked again which can introduce significant overhead. Both traditional and MMT designs make use of a network and thus multiple threads that share data with each other, thus they both experience the same detriments on time performance. Both the traditional TI application and the MMT design are used to determine the effect of network communication on execution time. By comparing the two, we can also see if periodic model corrections have a significant effect on execution time.

Figure 5.3 shows the timing performance CDF graph for the master and controlled nodes for both designs. In both cases the controlled side is essentially the same as scenario 1 except instead of

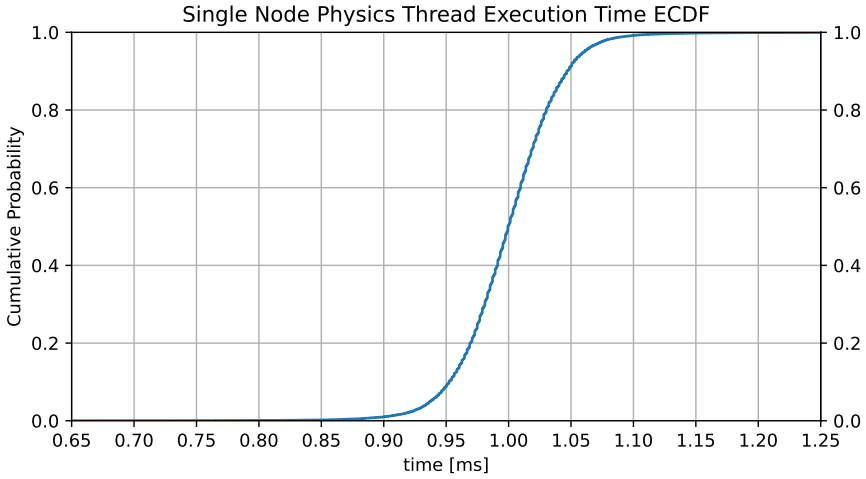
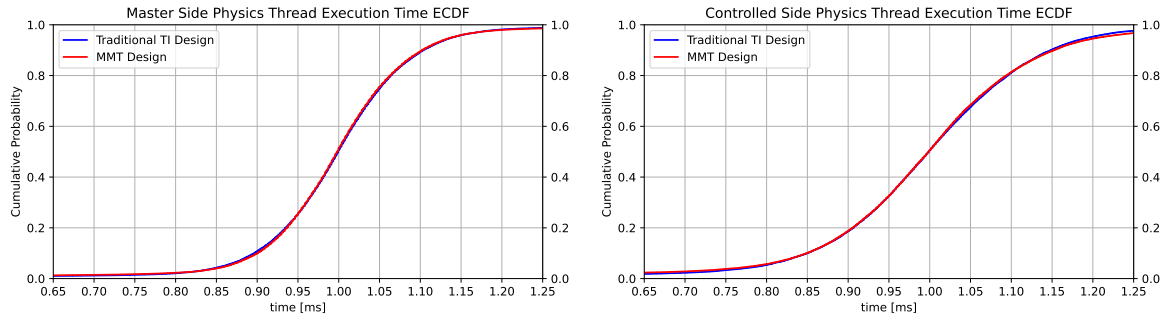


Figure 5.2: **Scenario 1 Timing Performance CDF Graph.** In this scenario only there is only one node communicating with a single haptic device which interacts with a virtual environment. There is only 1 thread and no network component.

I/O with the Novint Falcon, the controlled node sends data to the master node over a network. In the meantime a separate network thread receives position data which is accessed by the physics thread with a mutex. The plots for both nodes of both scenarios still show $1\,\mathrm{ms}$ at 50% but especially the controlled side is a lot more horizontal. This shows that there are many more cases when the execution takes longer significantly longer than 1ms, but they are always accounted for by making later threads shorter. All things considered, the fact that both nodes show symmetry around the 50% mark of the CDF graph assures that while the performance can be poor at times, it is not a consistent phenomenon and it can be accounted for by making future loops shorter. 90% of the time the thread executes in less than $1.15\,\mathrm{ms}$ in both scenarios which we argue is not a very significant amount of variation but is still indicative that there is need for optimization. Having relatively frequent execution times above $1\,\mathrm{ms}$ would introduce additional delay to the system, which would not only harm quality of experience but also make it difficult to accurately test the effects of different network conditions.
Inference 2: The execution time of the physics thread is affected negatively by multiple threads that share resources. However the effect is not consistent and the average execution time can always be restored to $1\,\mathrm{ms}$.

For both nodes, the MMT scenario and the traditional TI scenario have almost identical CDF graphs. While there are some differences, neither line is consistently below or above the other when it comes to executions that took longer than $1\,\mathrm{ms}$ which means neither outperforms the other consistently. This indicates that MMT introduces no additional overhead in execution time compared to the traditional TI applications. MMT is different mainly in its master node running its own physics thread as a local model. Running this additional physics engine on top of the one running at the controlled node on the same computer does not seem to affect the execution time, which is expected to be the case as long as the CPU has enough cores to spare for these threads.
Inference 3: As long as the CPU has enough cores and the communication overhead is negligible, running multiple instances of the application causes negligible increase execution times.

An interesting observation from these plots is the vast difference between the two nodes. The execution times of the controlled side are significantly worse than those of the master side. If this was only for the traditional TI application, this could be explained by the lack of a physics engine at the controlled side, however in MMT both nodes run an identical physics engine which disproves this explanation. This is a peculiar behavior because for MMT, the master side has not only the same physics engine and network communication, but also controller I/O which we later found to be one of the biggest contributors to the overall execution time. We believe that the a possible explanation for the poor performance of the controlled side is that the mutex for the controller position is much more detrimental to the execution time than the mutex for object data or feedback force used in the master node of both designs. A big target for optimization in this case would be to reduce the frequency of mutex locks in the code.



(a) Master Side Performance

(b) Controlled Side Performance

Figure 5.3: **Timing Performance CDF Graph for the Traditional TI design (in red) and the MMT design (in blue).** In both scenarios the controlled side runs a physics engine as the virtual environment and sends & receives similar data to/from the master side. The master side simply handles controller I/O in traditional TI, but also has a local physics engine in the MMT design. The two design show no significant difference in timing performance.

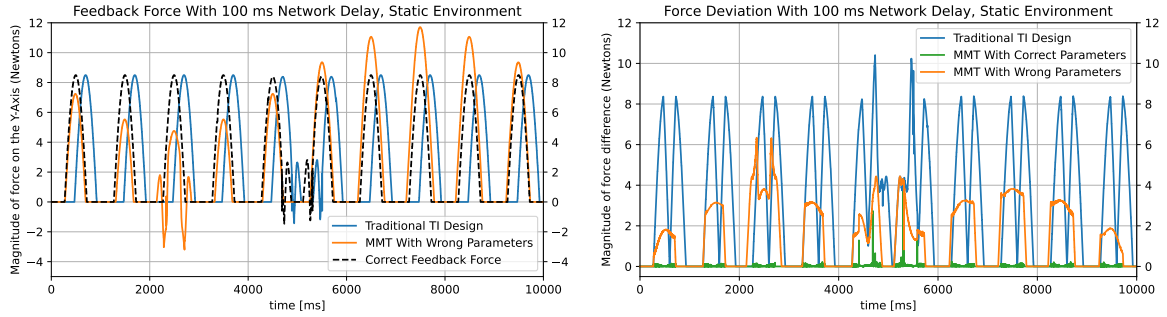
5.3. Static Model Divergence

In this section, we compare MMT to the traditional TI design, and see what kind of results an inaccurate model can create. A major claim of this thesis is that the local model of MMT is a perfect copy of the controlled side environment, then MMT solves TI completely. While the traditional TI suffers from delayed haptic feedback as a direct result of network latency, a perfect model could provide perfect instant feedback to the user. Unfortunately, no model can the perfect representation of the real environment. Even if all the parameters of the model converge to their true values over time, model is a simplification of the real world and does cannot simulate every physical interaction perfectly. Additionally, its parameters are not continuous and have limited precision.

Figure 5.4 (a) shows the Y-axis component of the feedback force experienced at the master side over time. The delay of the traditional TI system shown in blue causes a shift in the force as expected, meanwhile MMT with the inaccurate model parameters have the peaks at correct times but in wrong magnitudes because the network delay is eliminated but the ground is incorrectly modeled as slanted. It's also visible that earlier collision with the sphere not only changes the timing of the force caused by it but also its intensity and direction. The correct timing of the peaks indicates that MMT is completely unaffected by delay in a static environment. While not plotted here, repeating the same experiment with no network delay yields identical results. A model with perfectly accurate parameters yields no error regardless of delay. The only source of inaccuracy is the inaccuracy of model parameters.

Figure 5.4 (b) shows the magnitude of the force error calculated as in equation 5.1. This measure takes into account all axes so it allows a better view of how wrong the feedback force was. As expected, wrongly modeling the ground as slanted and changing the position of the sphere causes consistently wrong force feedback. The average magnitude of force error in this experiment is $1.4\,\mathrm{N}$ for MMT with wrong parameters which one might say is unacceptably inaccurate. However perhaps more importantly than these quantitative measures is that the user is interacting with a fake environment and the interactions are consistent in that fake environment. While it would be immediately obvious if the sphere is felt to be in a completely different spot than it actually is in this case, mismatch between visual and haptic sensory data can be unnoticeable in some cases. The parameters were chosen to be highly wrong on purpose for this experiment to magnify the effect, but with small differences the effect might be ignored. Even with these highly wrong parameters, the force error is consistently lower than that of the traditional TI with 100ms delay, which experiences an average of $3.25\,\mathrm{N}$. When the shift caused by delay is accounted for, the error drops down to same levels as MMT with correct parameters but it is important to reiterate that in a read feedback loop network delay causes additional error as a result of over-penetrating surfaces. Future study is needed to quantitatively compare the effects of realistically wrong MMT parameters to the effects of network delay in traditional systems. The current data makes us believe that in a static environment, MMT vastly outperforms traditional TI systems under delay. This is augmented by the fact that traditional TI systems deteriorate more as network delay increases meanwhile MMT systems are completely unaffected by it.

Inference 4: The feedback force error in MMT is not affected by network delay for static objects.



(a) (b)

Figure 5.4: Effects of MMT and traditional TI design on the feedback force in static environment with network delay. Plot (a) shows the force experienced on the Y-axis. Plot (b) shows the magnitude of force error (on all axes). The Y-axis force is not plotted for the MMT with correct model parameters, because it is almost identical to the correct force. Its slight differences are plotted in green in the second plot.

The delayed force experienced in the traditional TI design is shifted by network delay but the previously mentioned force spike cannot be seen. This is because the HIP follows a pre-programmed trajectory and does not get affected by the feedback force. Eliminating the effect of the feedback force on the position of HIP breaks the feedback loop in a sense. The amount of penetration does not change by the force in this experiment meanwhile a real person would keep pushing until a sufficient force is felt. This shows that using pre-determined path for HIP is not adequate for testing the feedback loop of traditional TI systems under delay. We still used this pre-determined path in our experiments because the focus of this thesis is MMT which can be accurately tested with a pre-determined path. When comparing MMT with traditional TI, enough insight can be gathered without the effects of over-penetration.
Inference 5: Using a pre-determined path for HIP does not allow analyzing closed loop feedback behavior of traditional TI systems.

In Figure 5.4 (b), the force error of MMT with perfect model parameters is also plotted over time. While the error is mostly 0, slight errors can be seen during collisions. We believe this is caused by the master and controlled nodes running out of sync and the feedback force being calculated asynchronously. When plotting, the forces experienced at the beginning of each millisecond are compared because both nodes run at $1\,\mathrm{kHz}$ but the actual timestamps are out of sync by various amounts of microseconds. This means that the experiment is not fully deterministic and no two runs of the experiment will yield identical results. The same error is also visible for the traditional TI system with 0 delay so it is same to say this is a baseline amount of error that cannot be eliminated from the system unless the two nodes are perfectly synced in time. The average magnitude of the force error in MMT with perfect parameters is $0.02\,\mathrm{N}$ which is not only extremely low, but is too precise a force for the Novint Falcon device that we used for our experiments [33]. For this reason these small errors across repeated simulations are insignificant.
Inference 6: The experiments are not fully deterministic. However small errors across repeated simulations are not significant.

A final important finding from this experiment is that as expected, the model error is not divergent in a static environment. Both the force and its error would repeat the same pattern forever in time if the HIP kept looping on the same pre-determined path. The ground truth values of the parameters do not change over time so a parameter value cannot get more wrong over time. If the parameter error was plotted, it would be a straight horizontal line. This means that model parameters do not need to be updated periodically unless a more accurate sensor reading arrives. This significantly relaxes the load on network and simplifies the master side local model.
Inference 7: In a static environment, the local model does not diverge over time. The error of the model stays constant unless a new sensor reading provides more accurate results. This means the master side model can be corrected much less frequently.

With these results, we reiterate our claim that MMT effectively solves TI for static environments, with a few small caveats. Wrong sensor readings and weak models can cause errors in haptic feedback. However the effect of network delay is completely eliminated which enables TI over long distances without reducing quality of experience.

5.4. Dynamic Model Divergence

In this chapter we explore model divergence in MMT for dynamic environments, where at least some objects are movable. It is claimed in this thesis that dynamic objects are the crux of the problem with MMT which makes this the most important area of exploration. Even if all parameters of the model converge to their true values, some parameters of the dynamic environment change over time by nature. This causes inevitable divergence of the local model. To put this hypothesis to test, the a dynamic environment with a movable cube is used. The cube's unchanging parameters such as mass, friction and starting position are assumed to be perfectly known by the master side model of MMT. We examine model divergence both with and without model corrections. Looking at MMT without model corrections allows us to observe the divergence uninterrupted and compare it to the static environment. Then, by looking at MMT with periodic corrections we see the efficacy of our method of correction.

Before starting to experiment with MMT with dynamic objects, the dynamic setup was tested with the traditional TI implementation to first see if there are any additional force error introduced to the system by movable objects when no local model is involved. This sanity check showed expected results where the average magnitude of force error was $0.017\,\mathrm{N}$ with $0\,\mathrm{ms}$ delay, and $0.009\,\mathrm{N}$ with $100\,\mathrm{ms}$ delay (when adjusted for delay). As previously discussed this is the baseline error caused by our nondeterministic testbed. This sanity check ensures that whether or not an object is dynamic has no effect on the correctness of the haptic feedback for traditional TI systems.

Model divergence is best illustrated by the positional error of the dynamic cube which consistently increases over the course of the experiment from the beginning of the first contact with the proxy pointer. Figure 5.5 (b) shows the distance between the origin of the box at the master side and at the controlled side. As seen in the plot, not only does the divergence increase over time when the dynamic interaction is occurring, but also unlike the feedback force, this divergence is permanent. The distance between the two cubes are fairly small in this experiment but the fact that the error increases over time makes it obvious that even if every parameter of the model converged to their ground truth values, changing parameters of a dynamic environment such as position and velocity need to be consistently updated with their real values.

Inference 8: Interacting with dynamic objects causes the local model to diverge from the environment.

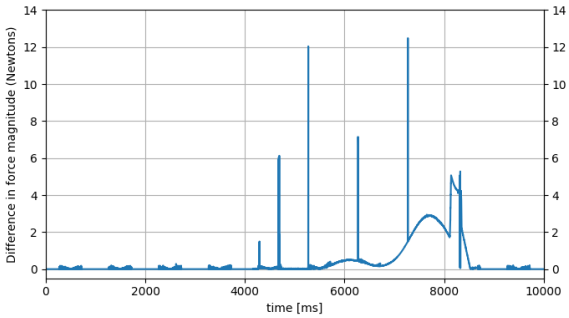
In the same graph, the benefits of correcting the model is obvious: the error gets reduced. While error is still present when interacting with the dynamic object, it is much lower than the case without model corrections. More importantly, the error does not increase over time and does not persist after the interaction ends. Instead, the error completely disappears after enough time has passed.

Inference 9: Periodically correcting the model with correct parameter values reduces model divergence.

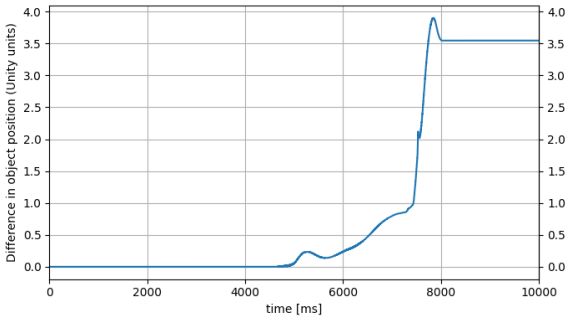
One might ask how can a perfect model diverge. While there are no perfect models in real life, using a virtual environment in our experiments ensures that we can have the identical model at the master and controlled sides. If the models are identical, why do they diverge? In our belief, this divergence is caused by the nondeterministic behavior of our experiments where the nodes can run out of sync. Applying the force at slightly different timestamps causes the force to be applied in slightly different locations, directions and magnitudes. Over time, these differences accumulate and produce visible errors in object position. This behavior is not unique to virtual environments. On the contrary, the discrete nature of the computer program causes inevitable loss of precision both in time and in 3D space which would create similar effects. Even more importantly, inaccurate sensor readings that estimate the wrong starting position or the wrong mass for the object would cause even faster deterioration of model accuracy.

Inference 10: What causes identical models to diverge from each other is the nondeterministic nature of the experiments. The effect would be amplified for real world environments.

It is possible to see in the object position graph that the correction seems to cause the object to jitter. The main reason for this jitter is likely that only position and rotation are updated but not their derivatives. The physics engine of the local model keeps track of not only positions but also impulses of objects, which remains unchanged when the object's position is updated. This means that an object that had a wrong amount of velocity towards a direction will keep the object sliding in that direction. In this case the model is not fully converged to the controlled side environment as some of its parameters are still different, which then causes the corrected parameters to diverge faster. Since the wrong parameters are the derivatives of the corrected parameters, they directly affect the rate of divergence. Not only does this cause overall higher levels of error, but also causes objects to shake around in the local



(a) Magnitude of force error over time.



(b) Positional error of the movable box over time. For reference, the movable box is a 15x15x15 cube.

Figure 5.5: Divergence of the master side local model in a dynamic environment, if the local model is never updated. As both plots show, the error in feedback force and object position increase over time when there is dynamic interaction. While not huge, the difference in object position is permanent. Thus, model needs to be periodically updated.

model until their momentum dissipates through friction. Interacting with these shaking objects, despite creating less error in numbers, creates an unrealistic experience where the object being touched feels to be vibrating.

Inference 11: Instantly updating the position and rotation of a dynamic object without also updating their derivatives causes the object to jitter.

To better examine the effects of this jitter on quality of experience, the force feedback must be analyzed. Figure INSERT FIG plots the Z component feedback force over time for both with and without model corrections. The Z axis is chosen because the box is mostly pushed towards the positive Z direction in the experiment. Figure INSERT FIG plots the magnitude of the force error calculated in the same manner as before. When the model is never corrected, it can be seen that the force error tends to increase over time as long as the dynamic object is interacted with. The force error is greatly reduced by model corrections but a significant jitter in force is visible which indicates that the jitter in object position also affects the feedback force. Quantitatively the feedback force is more correct, but the model correction scheme introduces effects that are detrimental to the quality of experience. A qualitative survey is needed to fully determine the effect of this jitter on user experience but it is apparent that better model correction schemes are needed to perfect interaction with dynamic objects in an MMT based TI application.

Inference 12: Object jitter caused by the overly simplistic model correction scheme directly translates to jitter in feedback force. Better model correction algorithms are needed.

6

Conclusion

Tactile Internet (TI) applications introduce countless new possibilities by enabling haptic interactions over the internet, but they are extremely limited by timing constraints. Even minimal amounts of network delay can cause poor quality of experience, which limits the applications of TI to short distances. Because of this, the majority of TI research focuses on reducing delay in TI applications. In this thesis we take a different approach and instead we relax the delay requirements. For this purpose we came up with a Model Mediated Teleoperation (MMT) approach which uses a physics engine as the local model of the environment that simulates the physical interactions in that environment. This local simulation provides instant haptic feedback regardless of network delay. With our proposition to use a physics engine as the local model, we open up new possibilities for model mediated TI in complex scenarios.

To implement an MMT that utilizes a physics engine, we reverse engineered an existing haptic rendering framework and implemented an enhanced local model for MMT by adding functionality for haptic interactions to an existing physics engine. We developed a multi-purpose TI framework attached to Unity that facilitates creating a variety of TI applications and testbeds, then we added our MMT implementation to this framework for both real and virtual environments. Using our Unity TI framework, we created testbeds to analyze our enhanced MMT implementation and to compare it with traditional TI designs. We performed timing analysis to ensure that the MMT implementation of our TI framework met the $1\,\mathrm{kHz}$ refresh rate requirement. We experimented with our enhanced MMT by interacting with static and dynamic objects in virtual environments. We demonstrated that our MMT approach eliminated the effects of network delay on haptic feedback for stationary objects. Finally, we identified that dynamic objects cause the local model to diverge from the environment, which makes correcting the model divergence with dynamic models the key challenge against making long distance TI in complex environments a reality.

Bibliography

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

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] Daniël Van Den Berg et al. “Challenges in Haptic Communications Over the Tactile Internet”. In: *IEEE Access* 5 (2017), pp. 23502–23518. DOI: 10.1109/ACCESS.2017.2764181.

[9] J.P. Verburg et al. “Setting the Yardstick: A Quantitative Metric for Effectively Measuring Tactile Internet”. In: (2020), pp. 1937–1946. DOI: 10.1109/INFOCOM41043.2020.9155540. URL: https://doi.org/10.1109/INFOCOM41043.2020.9155540.

[10] 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.

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

[12] 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.

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

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

[22] Xiao Xu et al. “Haptic data reduction for time-delayed teleoperation using the time domain passivity approach”. In: *2015 IEEE World Haptics Conference (WHC)*. 2015, pp. 512–518. DOI: 10.1109/WHC.2015.7177763.

[23] F. Conti and O. Khatib. “Spanning large workspaces using small haptic devices”. In: *First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics Conference*. 2005, pp. 183–188. DOI: 10.1109/WHC.2005.118.

[24] Jongeun Cha et al. “An AR System for Haptic Communication”. In: *Proceedings of the 2005 International Conference on Augmented Tele-Existence*. ICAT ’05. Christchurch, New Zealand: Association for Computing Machinery, 2005, pp. 241–242. ISBN: 0473106574. DOI: 10.1145/1152399.1152444. URL: https://doi.org/10.1145/1152399.1152444.

[25] Amit Bhardwaj et al. “A candidate hardware and software reference setup for kinesthetic codec standardization”. In: *2017 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. 2017, pp. 1–6. DOI: 10.1109/HAVE.2017.8240353.

[26] Xiao Xu, Qian Liu, and Eckehard Steinbach. “Toward QoE-driven dynamic control scheme switching for time-delayed teleoperation systems: A dedicated case Study”. In: *2017 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. 2017, pp. 1–6. DOI: 10.1109/HAVE.2017.8240352.

[27] Siwen Liu et al. “QoE-Driven Uplink Scheduling for Haptic Communications Over 5G Enabled Tactile Internet”. In: *2018 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. 2018, pp. 1–5. DOI: 10.1109/HAVE.2018.8547503.

[28] Massimo Condoluci et al. “Soft Resource Reservation for Low-Delayed Teleoperation Over Mobile Networks”. In: *IEEE Access* 5 (2017), pp. 10445–10455. DOI: 10.1109/ACCESS.2017.2707319.

[29] Kurian Polachan et al. “Towards an Open Testbed for Tactile Cyber Physical Systems”. In: *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. 2019, pp. 375–382. DOI: 10.1109/COMSNETS.2019.8711100.

[30] Kurian Polachan et al. “TCPSbed: A Modular Testbed for Tactile Internet-Based Cyber-Physical Systems”. In: *IEEE/ACM Transactions on Networking* 30.2 (2022), pp. 796–811. DOI: 10.1109/TNET.2021.3124767.

[31] Muhammad Zubair Islam et al. “IoTactileSim: A Virtual Testbed for Tactile Industrial Internet of Things Services”. In: *Sensors (Basel)* 21.24 (Dec. 2021).

[32] Vineet Gokhale et al. “TIXT: An Extensible Testbed for Tactile Internet Communication”. In: *IEEE Internet of Things Magazine* 3.1 (2020), pp. 32–37. DOI: 10.1109/IOTM.0001.1900075.

[33] Nima Karbasizadeh et al. “Dynamic Identification of the Novint Falcon Haptic Device”. In: Oct. 2016. DOI: 10.1109/ICRoM.2016.7886795.