Real-time Module Tracking for Reconfigurable Manufacturing Systems

Bachelor Thesis

Authors

Kasper S. Laursen
BSc. Student - Mechatronics
University of Southern Denmark
Exam Number: 439962

Rasmus K. Henriksen
BSc. Student - Mechatronics
University of Southern Denmark
Exam Number: 438412

Supervisors

Frederik Gottlieb
Special consultant - TEK Innovation
University of Southern Denmark

Christian P. Nielsen
Ph.d. Student - Innovation & Business
University of Southern Denmark

June 2, 2019

Abstract

This project enhances a Reconfigurable Manufacturing System by tracking the modules in real-time. This allows the factory to increase their efficiency by simulating a one-to-one digital version of their current configuration. It also supports other processes, such as self-driving modules, that rely on positional data.

The system utilizes HTC Vive trackers to track the position of modules in the system. The application protocol interface (API) and the graphical user interface (GUI) are programmed in Python. Methods for Siemens Tecnomatix have also been created to integrate it with the program. The application uses an SQLite database and provides a basic server that has access to positional data.

Keywords Reconfigurable Manufacturing Systems · RMS · Tracking · HTC Vive · Simulation

Acknowledgements

We would like to thank our supervisors, Frederik Gottlieb and Christian P. Nielsen, as well David G. Hansen for their guidance and support during the project. The Innovation Lab at University of Southern Denmark Sønderborg for allowing us to use their equipment and facilities. We would also like to thank Emily P. Theriot for creating illustrations for this report, as well as proofreading. Additionally, thanks to Lisa Frenzel for helping us test and demonstrate the program. Finally, thanks to Mikkel D. Olesen for his continuous feedback and bug reporting.

[illegible] [illegible]

Rasmus K. Henriksen	Kasper S. Laursen

Contents

1 Project Background

Reconfigurable Manufacturing Systems allow factories to adapt to a rapidly changing demand in the market. Traditional factory setups are optimized for high production of a small number of specific parts. Since they consist of a production line with various machines in set places, changing a production line is costly and requires a large amount of downtime. Reconfigurable Manufacturing Systems avoid these downsides. They are adaptable and can quickly be configured for the current demand.

Researchers at SDU Sønderborg have begun designing a mini reconfigurable manufacturing system (RMS), consisting of modular tables with various machines that can be linked together. The module positions will depend on the most optimal setup for the specific production, as calculated by the digital twin setup created in Siemens Tecnomatix. The digital twin is a one to one virtual replica of the factory setup, including specific machinery and processes, which allows for analysis of efficiency. This setup also allows for rapid reconfiguration of the factory, reducing the time necessary to set up new production lines.

At the time of writing, the developed prototype system has the ability to simulate the most efficient module setup and is able to move the virtual modules by the use of small objects on a glass plate. In order to expand upon this system, one could create a similar interface to work with the modular table system, which can update the current position of the modules in Tecnomatix. This project aims to fulfill this purpose, and create an easy to use and affordable real-time tracking system for these modules. Existing tracking systems will be considered, and evaluated on their precision, repeatability and easy of use. The system will be referred to RMTRMS (Real-Time Module Tracking for Reconfigurable Manufacturing Systems) from this point on.

2 Problem Formulation

The current prototype exists as a small scale proof-of-concept. A miniature model is created of the factory, and the user can move around small objects representing each machine. This is not ideal, as it makes it difficult for the user to maintain a sense of scale, and allows for unrealistic placement of machines. To mitigate these issues, one could use the actual machines and track their positions directly, as illustrated in Figure 1. For this to work, a real-time tracking system for the modules need to be developed, which leads to the question:

"How can we add real-time tracking of modules to this system?"

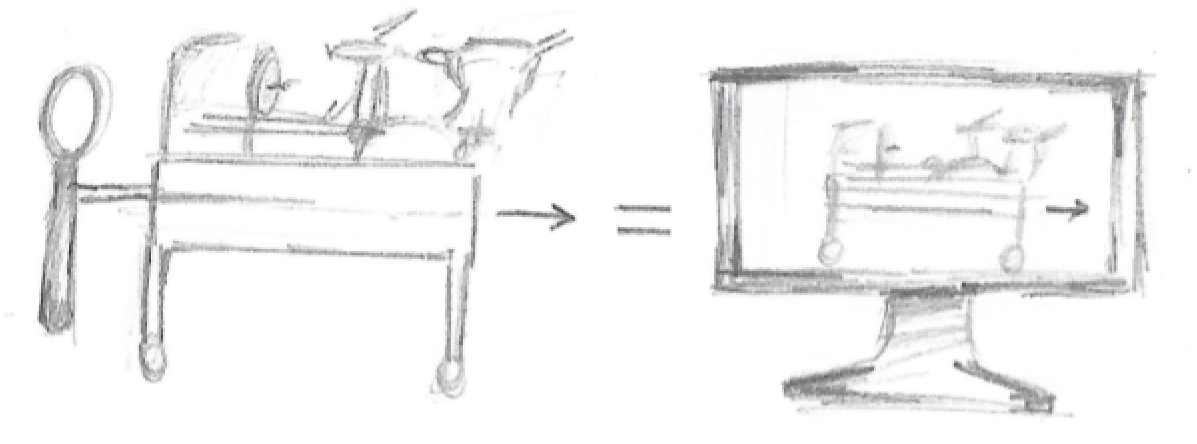


Figure 1: Problem sketch

2.1 Requirements

For the tracking interface to work efficiently with the virtual factory setup, the interface needs to fulfill a set of requirements:

- Retrieve position and rotation of module table with a deviation of max two centimeter.
- Update the X and Y position of the modules.
- Update the rotation along the yaw axis of the modules.
- Update module position minimum five times per second.
- Tecnomatix should update the virtual module positions five times per second.
- Table modules should be traceable minimum 95 % of their uptime.

2.2 Desirables

If all requirements are met before the deadline, the interface should be enhanced to work more efficiently:

- Retrieve position and rotation of module table with a deviation of max one cm.
- Update module positioning minimum 10 times per second.
- Tecnomatix should update the virtual module positions 10 times per second.
- Table modules should be traceable minimum 99% of their uptime.

In addition, the project should be expanded to include extra features:

- Intuitive graphical user interface (GUI).
- Update the rotation along the pitch and roll axis of the modules.
- Update the Z position of the modules.

2.3 Delimitations

It is important to define and limit the scope of the project due to the limited amount of time available. The project is being worked on while other courses are being attended simultaneously over the span of 4 months. This limits the amount of work that can realistically be done in the given time period for the project. The delimitations are described below:

- **Tracking System**
  The project will use an existing hardware solution for tracking the modules. This will allow the members to focus on the software solution, rather than developing hardware for tracking.
- **Module Movement**
  While the system will be able to work in conjunction with a movement control system, the interface will only transfer coordinates to and from the physical modules.

2.4 Budget

As with other Bachelor project at the University of Southern Denmark in Sønderborg, this project will have 1000 DKK allocated from SDU. The HTC Vive trackers and Lighthouses are supplied by Innovation Lab, and the Tecnomatix Plant Simulation license is from the SDU license server. The software used to run the VR tracking system is OpenVR which is open source and free of charge.

A full Bill of Materials can be found under Appendix A.1.

3 Methodology

3.1 Software Tools

The various software tools used in this project are described below:

- **Git**
  A powerful and popular version control software. It allows several users to collaborate on a centralized repository, tracking the changes each contributor makes.
- **Tecnomatix**
  A plant simulation program created by Siemens. It can simulate various processes and production lines in a factory setting. This program is currently used by the Innovation Lab at the University of Southern Denmark Sønderborg to model their reconfigurable manufacturing system.

3.2 System Languages

The interface will be created using primarily two languages:

- **Python**
  Python is main language for the interface. It is an easy to use, intuitive, and well-documented language. Since it is an interpreted language, it allows for quick prototyping as it does not need to compile between runs.
- **Simtalk**
  Simtalk 2 is a programming language developed by Siemens specifically for Tecnomatix.

3.3 The Waterfall Model

The project has been completed using mainly the Waterfall model, a model commonly used in software development[1]. It follows a series of sequential steps, described in Figure 2. The Waterfall model is advantageous in projects where requirements are well-defined and unlikely to change. However, it is limited by its rigid structure and inability to revisit previous steps.

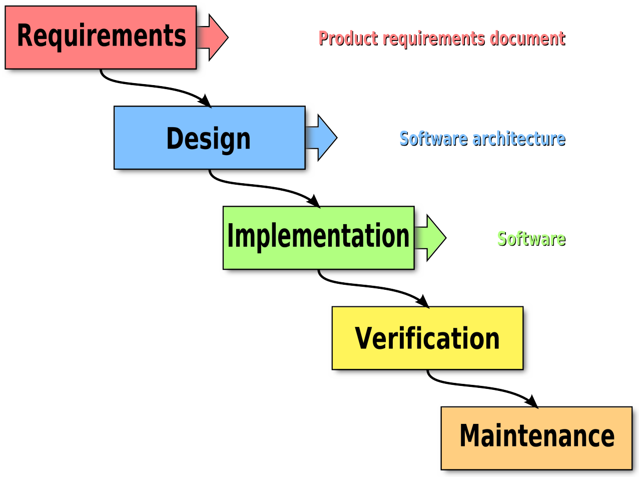


Figure 2: The Waterfall model[1]

[1]Peter Kemp / Paul Smith (https://commons.wikimedia.org/wiki/File:Waterfall_model.svg), „Waterfall model", https://creativecommons.org/licenses/by/3.0/legalcode

3.3.1 V-Shape Model

The V-shape model, described in Figure 3, is an extension to the Waterfall model which includes revisiting previous steps. This allows the model to be more adaptive, yet still maintain the strong structure of the waterfall.

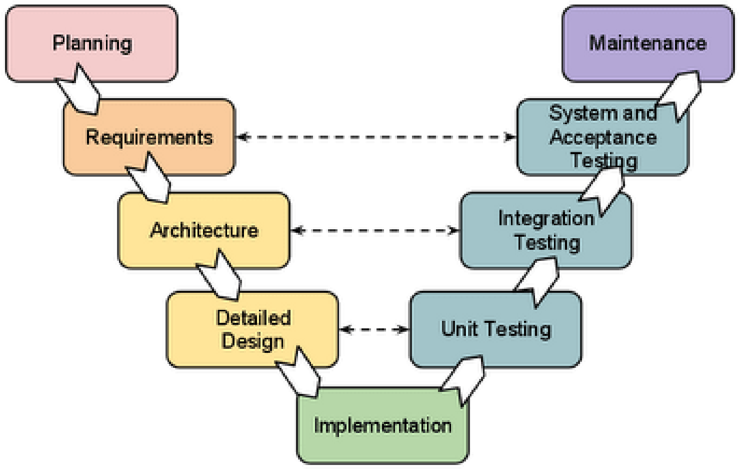


Figure 3: The V-Shape Model[2]

The requirements were determined in collaboration with the researchers working on the manufacturing system. After the requirements were set, all the necessary functions were decided and an overall software architecture was designed. Using the V-model allowed the developers to get feedback on the current iteration of the project and use this to redesign parts.

3.3.2 Milestones

The overall goal with the project has been split up into smaller milestones which should be reached throughout the project:

- Get tracking data in correct format (X, Y, Rotation)
- Integrate with Tecnomatix
- Create database system
- Create PCB with pogo pins for trackers
- User Testing

3.3.3 Timeline

To create a timeline, the milestones are all assigned due dates. The assigned dates are dependent on the importance of the task and which tasks are dependent on other tasks. The proposed timeline can be seen in Figure 4.

[2]Herman Bruyninckx (https://commons.wikimedia.org/wiki/File:V-model-en.png), „V-model-en“, https://creativecommons.org/licenses/by-sa/3.0/legalcode

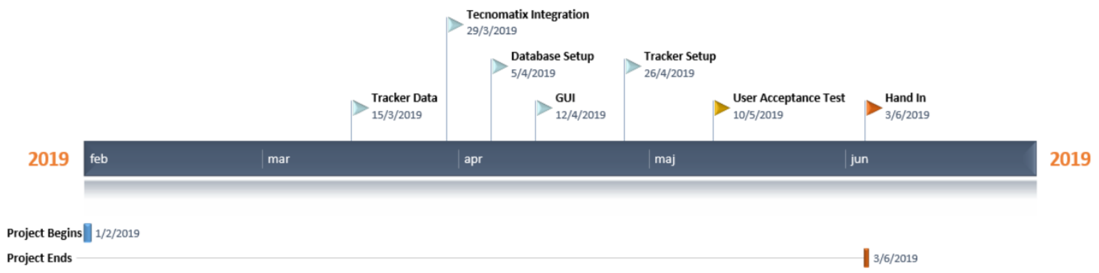


Figure 4: Milestone Timeline

A Gantt chart is used to plan out the various phases of the project. The main phases are subdivided into smaller tasks. These tasks reflect some of the major milestones of the project. the Gantt chart can be found under Appendix A.2.

4 Software Architecture

The system must be expandable and easy to use. It should support several use cases, including simulation, self-configuring modules and error correction. Therefore, it is important that the overall architecture is designed with these elements in mind.

The overall software architecture is outlined in Figure 5.

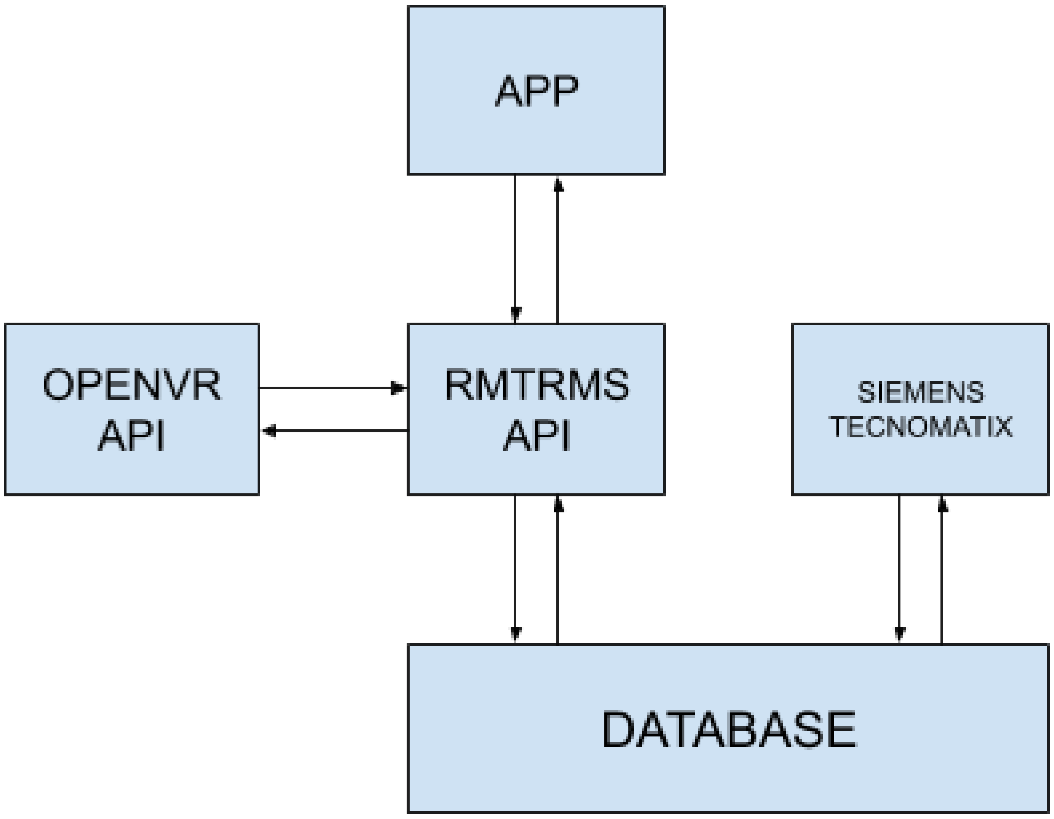


Figure 5: Software Flowchart

The goal is to provide an easy-to-use API that will allow expansion in the future. This means that all the OpenVR and database interactions should be abstracted away. This is achieved by writing an intermediary set of objects and functions that provide the necessary utilities. These are described in Section 7

The source code for all parts of the project can be found on Github at https://github.com/Madmalicius/RMTRMS[3].

5 Tracking System

5.1 State of The Art

There are many technologies in use for tracking today. These include vision/camera based systems, IR sensors and beacons, GPS systems, WLAN, Bluetooth, RFID and a variety of other technologies. A paper by Liu[2] from 2007 provides a quick overview of some of these technologies. The accuracy, precision and scalability of these systems are particularly important for this system. Since the project requires high accuracy, only systems with an accuracy of 50 cm or better will be considered. Out of the technologies discussed in the paper, Ubisense (15 cm), Sapphire Dart (30 cm) and SmartLOCUS (15 cm) fulfill this requirement. Ubisense, Sapphire use UltraWide Band (UWB) technology, while SmartLOCUS uses WLAN and ultrasound technology.

A more recent paper[3] from 2015 discusses some more general technologies and their accuracy. None of the technologies except UWB meet the requirement of better than 50 cm accuracy. According to this paper, the accuracy of these RFID systems is down to a few centimeters.

The aforementioned systems are usually implemented to track inventory or objects within a factory setting. More accurate systems are usually implemented for robotics projects. These usually utilize technologies with a faster response time, such as line of sight systems, such as vision and beacon based tracking systems and dead reckoning[4]. Vision based systems usually use a combination of cameras and special tags or lights to recognize various devices. Beacon systems use Time of Flight calculations to estimate the position. Dead reckoning systems use on-board sensors, such as gyroscopes and accelerometers, and a known position to determine any changes in position.

One area of technology that has advanced rapidly in recent year is virtual reality. The rise in popularity of these systems means that the tracking capabilities has improved dramatically. One of the most popular systems is the HTC Vive, which uses Valve's lighthouse technology. The system consists of one or more base stations (lighthouses) which emit a horizontal and vertical laser, as well as a synchronization signal between each sweep. The tracked objects in this system have up to 32 photodiodes in fixed, known positions relative to each other[5]. This allows the object to derive its orientation and position based on when the light hits each sensor. HTC has also made dedicated trackers, which contain several other sensors, like accelerometers and gyroscopes, in addition to the photodiodes. The position is primarily tracked based on the inertial measurements, and the photodiodes used for error correction. This error correction is performed 120 times per second, much faster than any of the previously discussed methods.

Compared to other similar tracking systems, such as the WorldViz PPT or PhaseSpace, the HTC Vive has a similar amount of jitter and slightly worse error correction if tracking is lost[6]. A slight shift in the XY plane was observed using the Vive system if tracking is lost and regained. This shift was also observed by E. Luckett[7] who performed a similar experiment. This shift was observed in both the Vive system and PhaseSpace system. The price of a Vive tracking system is also much lower than the price of similar systems, at around 5,000 DKK (around $800 USD) at the time of writing. For comparison, the WorldViz PPT system costs around 98,000 DKK (around $15,000 USD) and the PhaseSpace system costs around 131,000 DKK (around $20,000 USD). Despite the limitations, the HTC Vive system provides positional tracking with errors in the millimeter range and with NASA's modified algorithm, it is well suited for robotics tasks[8]. NASA is using the system as a source of ground truth for their Astrobee project. They have developed an improved algorithm that puts less weight on the inertial measurements. Using their algorithm, the Vive tracking system has a standard deviation of 5 mm and a max deviation of 28 mm.

5.2 HTC Vive

The HTC Vive is a virtual reality system designed by HTC and Valve Corporation. It is a consumer product aimed at gaming and virtual reality experiences. It provides a robust tracking system for the headset, controllers and accessories. The HTC Vive system will be used to track the position of the modules. This system currently allows for up to 64 objects to be tracked on a single host[9]. This limit may be increased in the future, as SteamVR is still in active development.

This system was chosen due to the relatively low cost (compared to similar systems), high accuracy and well established user base. The accuracy of the system is tested and the results are found in Section 10.1.

[3]The Github repository and wiki will be accessible after publication

5.3 OpenVR

The HTC Vive system runs on the OpenVR API. Valve has created an implementation of this API, known as SteamVR, which is distributed through their platform, Steam. This API allows access to VR hardware from multiple vendors, including the HTC Vive. Using OpenVR, the user is able to view the position of any headset, controller or tracker connected to the system.

The API is written in C++, however, Python bindings are provided by a Github user, “cmbruns” [10]. These bindings allow Python to interact with the API. On top of this wrapper, Triad Semiconductor have created an enhanced wrapper, making it easier to use[11].

The tracker positions are sorted in a database by the individual tracker’s serial number. This is done since SteamVR will give the trackers different names depending on which is found first, but the serial number is unique to the individual trackers. This way a tracker can be assigned to a module.

6 Database

The backbone of the system is a database that will contain all the relevant information for the system to operate. These include:

- Available trackers and their positions (from the Vive system).
- Available modules and their positions (from Tecnomatix).
- Which tracker is assigned to what module.

6.1 SQLite

The project is designed to run on a single system, therefore the database should be local on this device. This will increase reliability and speed compared to an online solution. Since the database is only used internally there is no need for a server-based solution. Therefore, SQLite has been chosen as the database type, since it uses the same SQL commands as e.g MySQL, but creates and uses a local .db file instead of requiring an online database. The database file is created through the GUI.

SQLite is also directly supported by Tecnomatix, which has a specific module for SQLite (see Section 8.2).

6.2 Database Tables

The database is split up into three tables:

- Modules.
- PositionOut.
- Trackers.

6.2.1 The Modules Table

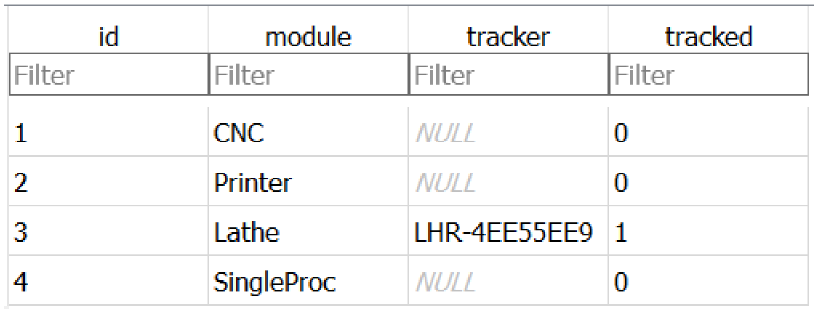
The Modules table has three columns besides the row id. This table is used to determine which tracker is assigned to which module and whether or not that module should be tracked in Tecnomatix.

The first column, Module, contains the names of the modules created in Tecnomatix, and is updated when the Tecnomatix simulation is run.

The second column, tracker, contains the serial of the tracker assigned to the module, chosen through the GUI. If no tracker is assigned, this field’s value is Null.

The third column, tracked, contains a boolean value which tells whether or not the module on the specific row should be tracked in Tecnomatix or not. If the value is True, the module will be tracked in Tecnomatix.

The Modules table is shown in Figure 6.



id	module	tracker	tracked
Filter	Filter	Filter	Filter
1	CNC	NULL	0
2	Printer	NULL	0
3	Lathe	LHR-4EE55EE9	1
4	SingleProc	NULL	0

Figure 6: Example of the Modules table

6.2.2 The PositionOut Table

The PositionOut table has four columns besides the row id column. This table is used to save the positions of the modules in Tecnomatix. These are used as the desired position of the module for the Self-Driving Modules for Reconfigurable Manufacturing Systems With Digital Twin project by Mikkel Dupont Olesen.

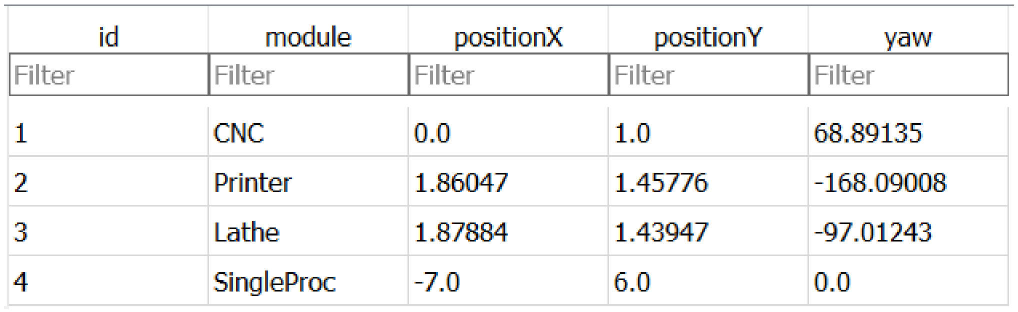
The first column, module, like the column in the Modules table, contains the name of the modules created in Tecnomatix.

The second column, positionX, contains the module's X-axis position from Tecnomatix.

The third column, positionY, contains the module's Y-axis position from Tecnomatix.

The fourth column, yaw, contains the module's yaw rotation from Tecnomatix.

The PositionOut table can be seen in Figure 7.



id	module	positionX	positionY	yaw
Filter	Filter	Filter	Filter	Filter
1	CNC	0.0	1.0	68.89135
2	Printer	1.86047	1.45776	-168.09008
3	Lathe	1.87884	1.43947	-97.01243
4	SingleProc	-7.0	6.0	0.0

Figure 7: Example of the PositionOut table

6.2.3 The Trackers Table

The Trackers table has nine columns besides the row id column. This table is used to store all known trackers' position data, serial, and name given.

The first column, name, contains the name of the tracker given in the GUI. If no name has been given, it will be given a default name "Tracker #" where # is the row id number.

The second column, serial, contains the serial number of the specific tracker.

The third column, active, contains a boolean value used to check if the tracker is currently active. If the tracker is found by SteamVR, it is considered active and the value will be set to True.

The fourth column, positionX, contains the last known X-axis position of the tracker.

The fifth column, positionY, contains the last known Y-axis position of the tracker.

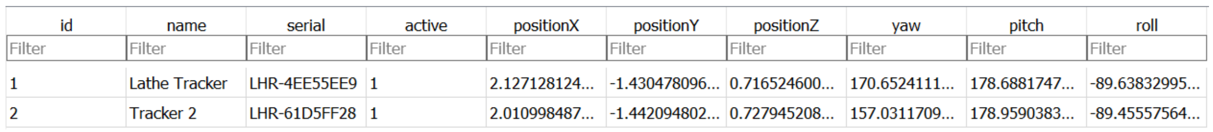
The sixth column, positionZ, contains the last known Z-axis position of the tracker.

The seventh column, yaw, contains the last known yaw angle of the tracker.

The eighth column, pitch, contains the last known pitch angle of the tracker.

The ninth column, roll, contains the last known roll angle of the tracker.

The Tracker table is shown in Figure 8.



id	name	serial	active	positionX	positionY	positionZ	yaw	pitch	roll
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Lathe Tracker	LHR-4EE55EE9	1	2.127128124...	-1.430478096...	0.716524600...	170.6524111...	178.6881747...	-89.63832995...
2	Tracker 2	LHR-61D5FF28	1	2.010998487...	-1.442094802...	0.727945208...	157.0311709...	178.9590383...	-89.45557564...

Figure 8: Example of the tracker table

6.3 Pragmas - Database Settings

When an SQLite database is created, it has a set of default settings to determine how communication flows with the file. These are known as pragma statements, and SQL extension specifically designed for SQLite[12]. They are similar to regular SQL commands, but differ in the way they are called. Some pragma statements are run during the SQL compilation stage, which means they are only run once if a set of SQL commands are lined up to be executed, whereas others are run during the execution stage, so they execute along with every command lined up. The two most important pragmas for this project are Journal Mode and Synchronous.

6.3.1 Journal Mode

The Journal Mode pragma determines how the communication with the database is handled according to read/write procedure. The default mode is *DELETE* which utilizes a rollback journal and deletes it when the changes stored in the journal has been properly saved to the file.

Using this mode is not very efficient since it locks the table the journal writes to. This means other processes cannot read from the same table, which causes problems with the Tecnomatix integration. To ensure a smooth tracking, the python script in charge of updating tracker positions should be able to write to the tracker table while Tecnomatix simultaneously reads from the same table.

Instead of using a rollback journal, the mode can be set to *WAL* to use a write-ahead log. Where the rollback journal creates a temporary copy of the unedited database while the changes are written directly to the file, the write-ahead log creates a log file with the changes and commits these to the database. This allows several other processes to read from the same table in the database targeted by the writing process.

6.3.2 Synchronous

The default synchronous setting is *Full*, which is not needed when the database is using the *write-ahead log* journal mode. Therefore, the synchronous pragma is set to *Normal*, the recommended setting for the WAL journal mode, which synchronizes less frequently, but is as effective with the chosen journal mode. This lets Tecnomatix read from the database more frequently.

7 Python

The project is programmed mainly in Python with some parts made with Methods in Tecnomatix using the built-in program Simtalk 2. The part of the project made with Python is organized into 5 different files:

- RMTRMS.py
- CreateDatabase.py
- Configure.py
- Restore.py
- ModuleTracker.py

7.1 RMTRMS.py

The main Python script is RMTRMS, and utilizes the OpenVR wrapper created by Triad Semiconductor. The script consists of three objects, a Tracker object, a Database object and a Server object.

The Database object creates a connection to a specified database, and provides all the necessary functionality to manage this database. It allows the user to get a list of all known trackers. It also allows the user to get information about a certain tracker or module, such as which tracker or module it is currently assigned to. It provides functionality to create these links, as well as getting the status of the tracker. The object creates an abstraction layer on top of the database, making it easy to interact with.

The Tracker objects represent the HTC Vive trackers. It provides a number of properties, which are:

- Name
- Serial
- X-axis coordinates
- Y-axis coordinates
- Z-axis coordinates
- Yaw rotation
- Pitch rotation
- Roll rotation

This object allows for renaming and updating the position of the tracker if it is active. The object can be created with a connection to the virtual reality system, in which case they are "active". If they are created without a connection, they are "inactive" and have no positional data, but can still be managed. These Tracker objects are linked to a Database object, where by which they update their position and name in the corresponding database. It also allows the user to identify which tracker they are working on, by using the "rumble" output of the HTC Vive tracker. A light or sound emitting device can be connected to this output.

The Server object provides a web server that will serve the current module positions, as well as desired positions. The server will respond to http GET requests in the form of "/modules/MODULE_NAME". If the specified module does not have a tracker associated with it, or it is not active, it will return all zeroes. The server should be started in a separate thread so that it will not block the main program. The host and port can be specified. If they are not specified, the server will start with default values:

Host: 0.0.0.0 (This will listen on all addresses)

Port: 8000

The server can then be accessed on the local network at "ComputerIP:8000/modules/MODULE_NAME", or on the computer itself via "localhost:8000/modules/MODULE_NAME"

7.2 CreateDatabase.py

This script will create a SQL database in the folder it is run. It creates a database with the necessary tables:

- **Trackers**
  A table of known trackers. Has the name, serial, status and positional data.
- **Modules**
  A table of modules inside Tecnomatix.
- **PositionOut**
  A table of the positions of the modules inside Tecnomatix.

7.3 Configure.py

This script will configure SteamVR to run in headless mode (without the display). It will modify the SteamVR settings, and automatically make a backup of the current settings.

7.4 Restore.py

This script will restore the previous SteamVR settings that Configure.py modifies. This will also delete the backup file. Both of these scripts will throw an error if SteamVR is not installed. These functions are also both accessible from the GUI.

7.5 ModuleTracker.py

This is the main program that runs the GUI, as well as updating the position of the trackers. This is described in more detail in Section 9.

8 Tecnomatix

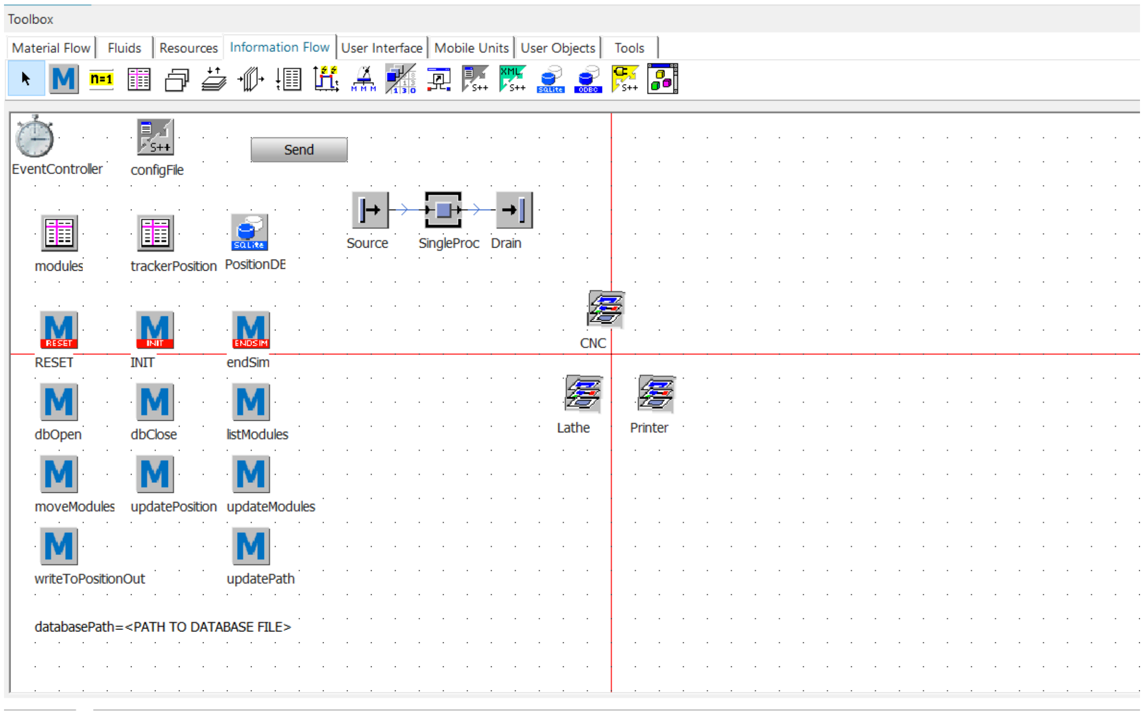
Siemens Tecnomatix is a plant simulation software which "enables the simulation, visualization, analysis and optimization of production systems and logistics processes"[13]. It allows the user to simulate a complete factory, including transport time between machinery, individual machine processing times and failure rates.

The programs uses a block-based, drag-and-drop system with each block split up into categories depending on their functionality. The setup used for the module tracking simulation is composed of several different block types listed in table 1:

Block type	Amount
Method	11
Frame	3
TableFile	2
Source	1
Drain	1
SingleProc	1
FileInterface	1
SQLite	1
Button	1
Variable	1

Table 1: Tecnomatix components

An example of how the setup can look can be seen in Figure 9. A more detailed guide for the simulation setup can be found on the Github Wiki. The wiki can be accessed at https://github.com/Madmalicius/RMTRMS/wiki/.



databasePath=<PATH TO DATABASE FILE>

Figure 9: Example setup

8.1 Methods - Tecnomatix Coding Blocks

The way programming works in Tecnomatix is through Method blocks. These blocks work like small scripts which can be called through other objects, and are programmed in Tecnomatix's own language, Simtalk 2. As with any module, these can be renamed, however Method blocks have special keyword names:

- **INIT**
  Keyword to make the Method block run when the simulation is initialized.
- **RESET**
  Keyword to make the Method block run when the simulation is reset.
- **ENDSIM**
  Keyword to make the Method block run when the simulation has ended.

All three keywords have been utilized in the simulation setup. The *INIT* Method is used open the database and update the modules created in Tecnomatix, where the *RESET* and *ENDSIM* Method blocks are used to safely close the database connection.

Besides the initial execution from the *INIT* block, a separate method block labeled *moveModules* is continuously executed throughout the course of the simulation. This is done by setting up a Source->SingleProc->Drain sequence and triggering *moveModules* every time the drain receives a resource. *moveModules* then updates the tableFile blocks which stores the database tables *trackers* and *modules* (see Section 6.2) by calling the *updatwModules* and *updatePosition* methods.

Regardless of whether the simulation is running or not, the user should be able to write the module positions to the database. This is done by calling a Method block labeled *writeToPositionOut* with a button block. *writeToPositionOut* is also called once when the simulation is initialized.

8.2 SQLite Integration

In order to write to and read from the database, the built-in SQLite module is used. It creates a copy of the database in memory and continuously updates this via the *moveModules* method block. If the modules have a tracker associated with them in the database, and they are labelled as "tracked", then they will update their position to match that of the tracker.

The path to the database file is loaded through the fileInterface block labeled *configFile* which gets the information from the config file generated by the GUI (see Section 9.3) and stores it in a variable block.

Tecnomatix also lists all modules, which are SingleProc and Frame blocks, and saves them to the actual database file. It does this as shown in Figure 10 when the simulation is started. The same procedure happens when *writeToPositionOut* is called.

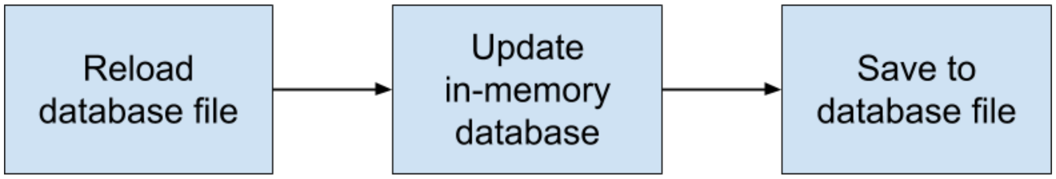


Figure 10: Tecnomatix database update flowchart

9 Graphical User Interface

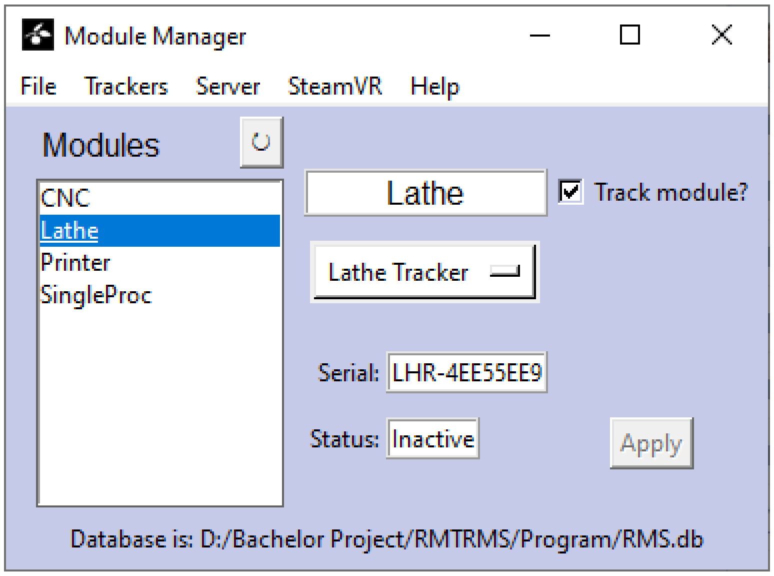
The graphical user interface is designed to be a separate, intuitive, easy-to-use manager for the modules and tracker data stored in the database. Its intended use is an easy way to assign a tracker to any given module. Furthermore, the GUI offers a way to rename, organize and delete trackers from the database. Besides the management of trackers, the GUI executable also updates the position of the active trackers, connected through SteamVR, in the database. This is done through a separate thread in the GUI process. This means, in order to update the modules in Tecnomatix, the GUI must be running.

The interface is created using the Python graphical User Interface package, Tkinter.

9.1 Main Window

The GUI's main window, as seen in Figure 11, consists of a list of known modules and information about their corresponding trackers' serial and activity status. Furthermore, it provides an easy way of assigning a new tracker to a module via the dropdown menu of all known trackers. A checkbox is also provided for letting Tecnomatix know whether or not the module should be tracked.

On top of the module list, a refresh button is also provided to update both the list of modules and the list of trackers.



Serial: LHR-4EE55EE9

Status: Inactive

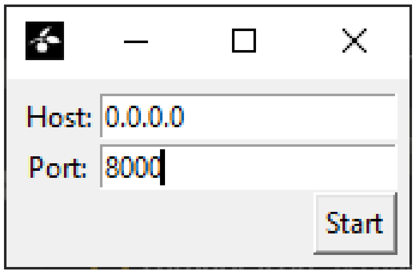
Database is: D:/Bachelor Project/RMTRMS/Program/RMS.db

Figure 11: Main window layout

9.1.1 Topmenu

The main window also includes a topmenu with five tabs, four of which have separate submenus:

- **File**
  Submenu for creating and opening database files.
- **Trackers**
  Opens the Tracker Management window. See Section 9.2.
- **Server**
  Submenu for starting and stopping a local server. The *Start Server* subtab opens a new window with *Host IP* and *Port* entries (see Figure 12).
- **SteamVR**
  Submenu for configuring SteamVR to run headless and restoring SteamVR to default.
- **Help**
  Submenu for opening documentation, reporting errors with the program and information about the authors. The documentation can be found in Appendix D



Host: 0.0.0.0

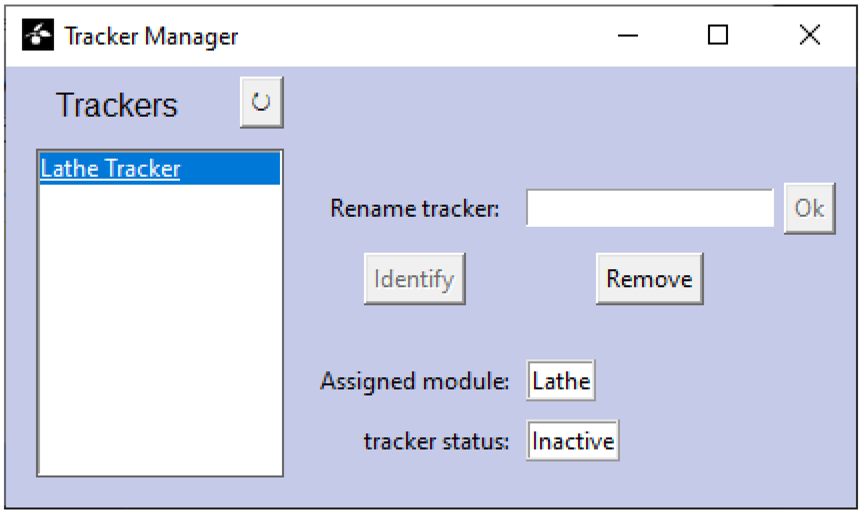
Port: 8000

Figure 12: Server window layout

9.2 Tracker Management Window

The Tracker Management window, seen in Figure 13, is a separate window for renaming and removing trackers stored in the database. It is opened via the main window's topmenu. It also provides status indication for the trackers along with the name of the module it is currently assigned to. The list of trackers can be refreshed by the button above the list, which reloads all tracker data from the database.

The tracker manager also provides a way of physically indicating and identifying the chosen tracker given it is active. Using the physical interface of the tracker[14], a light or sound emitting device can be triggered. This is done via a button labeled *Identify* which sends a signal to the tracker telling it to pull its digital output on pin 1 to high. Along with the GND connection on the tracker's pin 2, this can be used to trigger an LED or similar. This is shown in Figures 14 and 15.



Lathe Tracker

Rename tracker:

Assigned module: Lathe

tracker status: Inactive

Figure 13: Tracker window layout

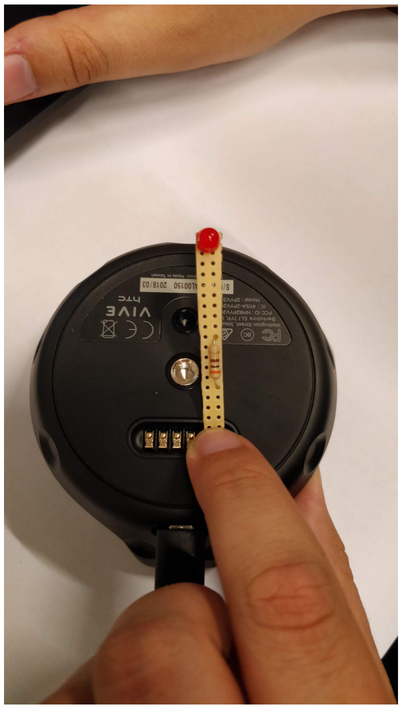


Figure 14: Default state

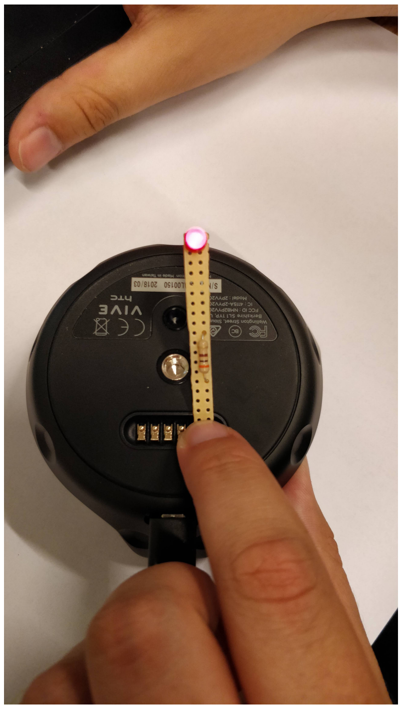
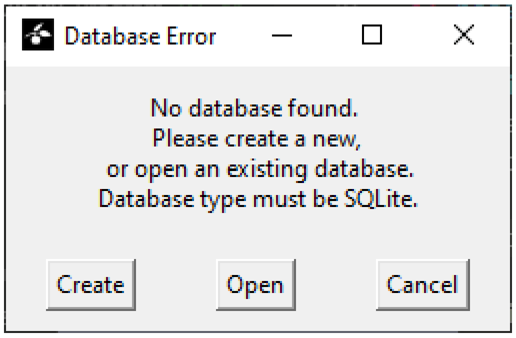


Figure 15: Identify State

9.3 First Time Startup

When the program is initially launched an "error" will occur due to a lacking config file. It will then ask for an SQLite database to be chosen or created with which it will generate a new config file containing the path to the database file. This is intentionally implemented as an error in case the database or config file have been moved or deleted, which would trigger the window on a new launch of the program. The error window can be seen on Figure 16.


Database Error

No database found.
Please create a new,
or open an existing database.
Database type must be SQLite.

Figure 16: No database error window

9.4 Error Handling

The program is designed to handle errors gracefully. If SteamVR is not installed the program will not launch, but instead open an error window asking for SteamVR to be installed. Other actions that results in errors include:

- Starting a server twice
- Trying to stop a non-existing server
- Open a file that is not a database
- Opening the program after moving or deleting the database or config file. See Section 9.3

10 Verification

10.1 Tracker Data

To verify the accuracy and precision of the Vive trackers, a series of tests were performed. All tests were performed in the same environment, using the same base station configuration and computer.

10.1.1 Stationary position

To test the accuracy and precision when the tracker is stationary over a period of time, the tracker was placed on a table in an empty room. The position was logged during a five minute period, and the change in position and orientation was graphed.

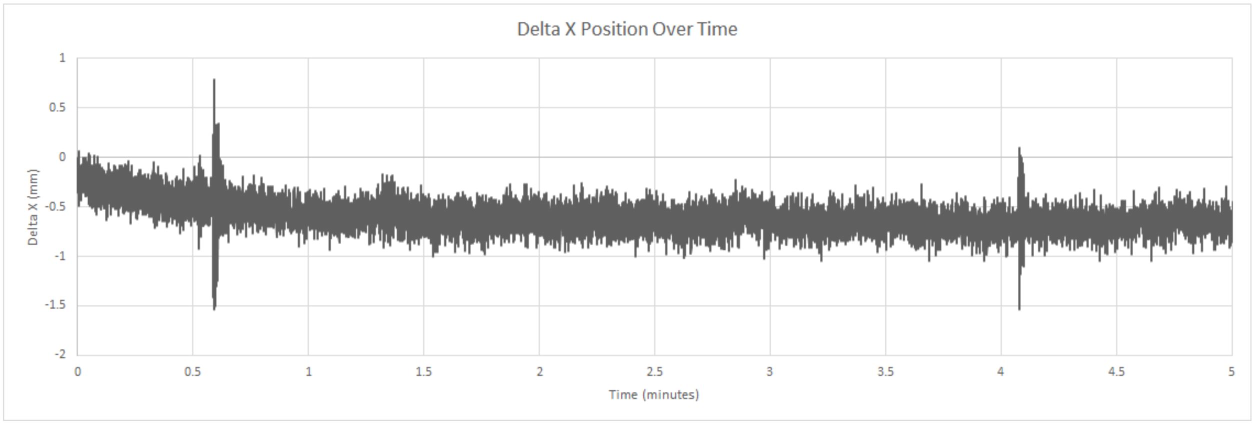


Figure 17: Graph of Change in X Position over Time

As seen in Figure 17, the X position varied by about half a millimeter on average. The spikes in movement are likely a result of the tracker being disturbed by someone walking by the room or closing a door nearby.

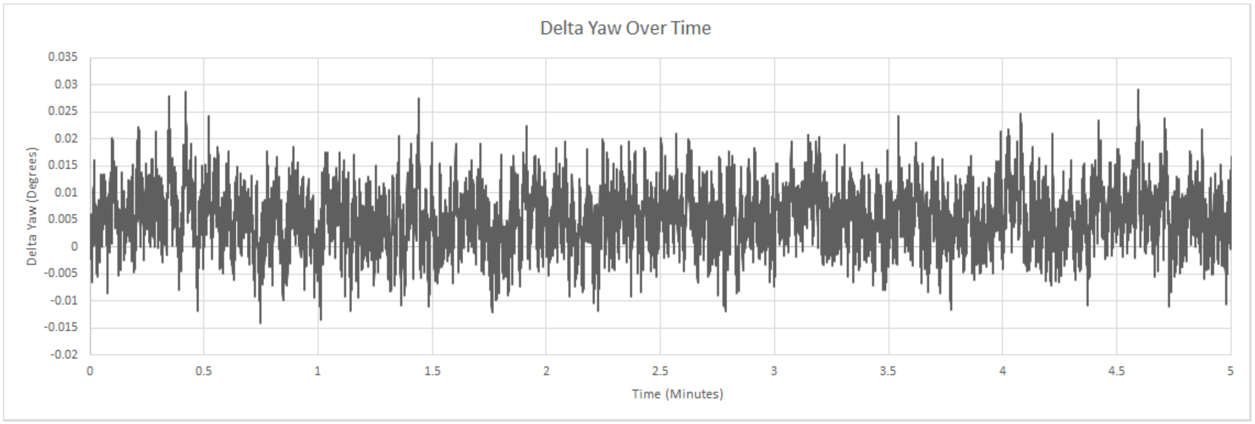


Figure 18: Graph of Change in Yaw over Time

Figure 18 shows the graph of the change in orientation along the yaw axis. The rotation deviates about 0.03 degrees at most over the five minutes.

Both of these results, as well as the rest of the data (see Appendix C.1) are well within the desired deviation of two centimeters.

10.1.2 Non-stationary position

To test the precision and reliability of the position when the tracker is moving, the tracker was placed on a turntable spinning at a constant speed and the location of the tracker was logged. The location was recorded for 1000 rotations and the results and setup are pictured below. The test setup can be seen in Figure 19.

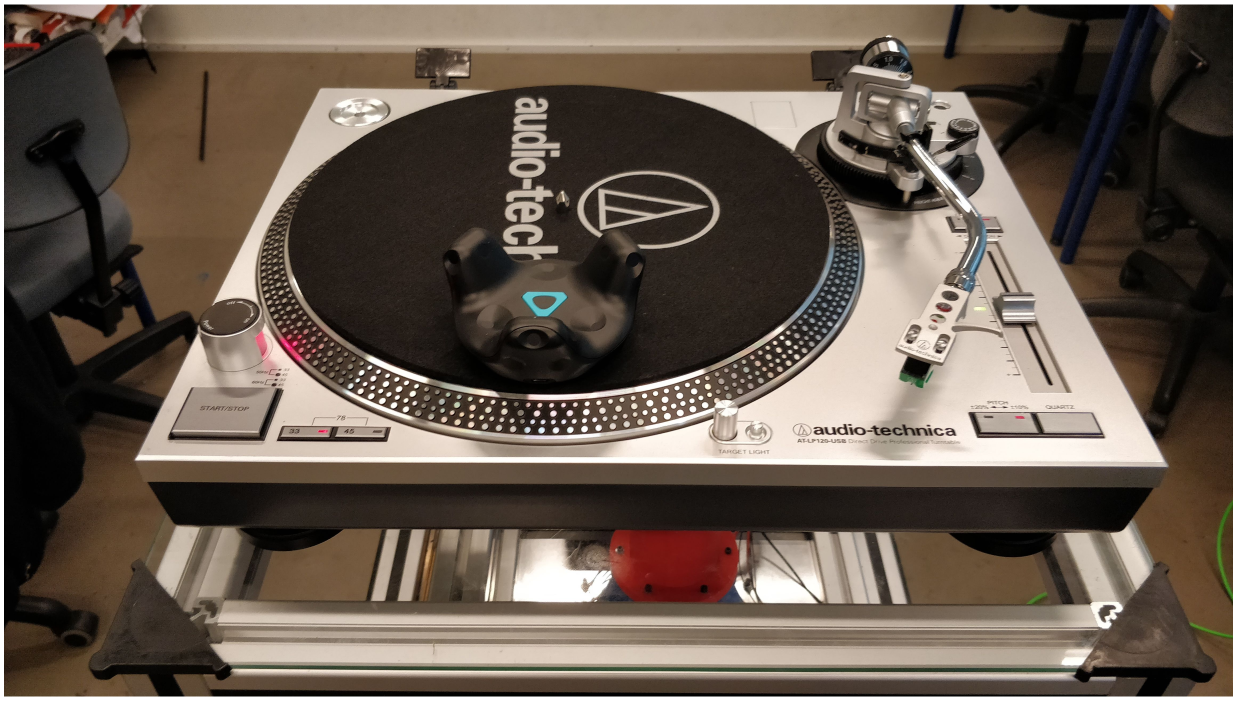
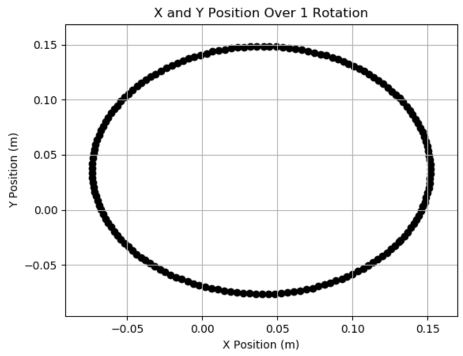


Figure 19: Test setup



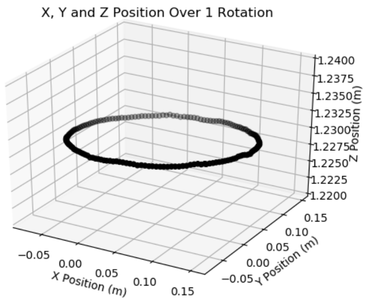


Figure 20: X and Y Position Over One Rotation

Figure 21: X, Y and Z Position Over One Rotation

The first rotation gives an indication of the trajectory the tracker should ideally repeat if it is precise. Figure 21 shows that the turntable is at a slight angle (for a side view of all the graphs in this section, see Appendix C.2). Figure 20 shows that the X and Y positions make a perfect circle, with little deviation from the expected trajectory.

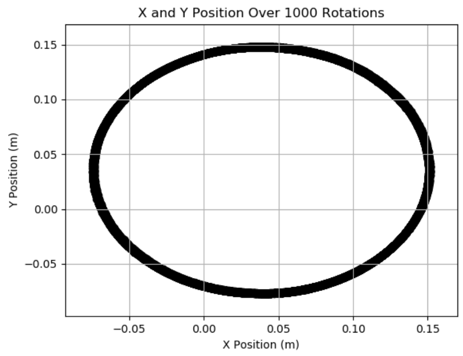


Figure 22: X and Y Position Over 1000 Rotations

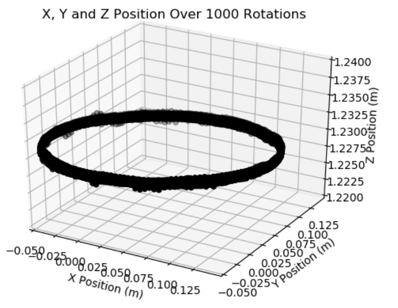


Figure 23: X, Y and Z Position Over 1000 Rotations

After half an hour and 1000 rotations, Figures 22 and 23 show that the tracker maintained the same trajectory, with little deviation. The tracking system consistently performs well.

10.1.3 Update Rate - Python

The update rate of the trackers position was measured using the time.perf_counter function in Python. This is a high performance timer meant for benchmarking software[15]. It provides very accurate timing functionality. This was used to determine the time between executions of the update function, as well as the time to execute. On average, the time between executions is 2.309 milliseconds and the time for execution is 2.23 milliseconds. This means all trackers are updated about every 4.539 milliseconds, or an effective update rate of 220 times per second.

10.1.4 Update Rate - Tecnomatix

A test was made in Tecnomatix to see how fast it updates its internal database and executes the method blocks. This is done by creating a set of variables saving the time in milliseconds of the simulation at the beginning and end of the triggered method block. By comparing the values, the time it takes for the method to execute and the time between executions can be found.

According to the test results, the execution of the method block is done immediately while the time between executions was 60 milliseconds. This means Tecnomatix updates the position of the modules around 16 times per second.

10.2 User Test

A qualitative user test was done to determine the intuitiveness of the GUI. One of the supervisors opened the program and went through his initial thoughts. His first impulse was to look for the help guide, which was found within the first five seconds after the program was launched. This indicates that the documentation button is placed in an intuitive position. The guide also helped explain the use of some features, but lacked a concrete workflow on how to use the program efficiently and should be added to the documentation of the program.

11 Conclusion

The system should be evaluated against the specified requirements and desirables, as well as the intended purpose. The system is able to retrieve the position of a module with an accuracy of one millimeter. This is far below the required two centimeters. It is able to update the X, Y, Z positions, as well as the rotation along all axes. This fulfills both the required and desirable characteristics. The system updates the position of the trackers at a rate of around 220 times per second, far above the desired 10 times per second. Tecnomatix updates at a rate of 16 times per second, above the desired 10 times per second.

The requirements state that modules should be traceable 95 % of their uptime. The HTC Vive trackers will be constantly available, as long as they are within sight of a base station. With SteamVR 1.0, there is a limit of two base stations, which means the modules can operate within a maximum area of 5 by 5 meters. As long as they are within these limits, they will always be traceable. SteamVR 2.0 introduces the new base stations, which allow more than two to be connected together. This means a virtually unlimited tracking area. This will allow larger setups to also fulfill the traceability requirement.

The project stayed well within budget. No money was spent due to the generosity of the Innovation Lab at SDU Sønderborg, who allowed the team to borrow their virtual reality equipment.

In terms of fulfilling the intended purpose and set requirement, the project was a success. The program integrates a tracking system with Tecnomatix, allowing real-time tracking of the modules. It allows the user to transfer positional data to and from Tecnomatix. Database management is automated through the graphical user interface. SteamVR configuration is also done through the interface.

12 Future Works

The program has been designed with expandability in mind, allowing future projects to utilize all or parts of the program. For example, a self-driving module could access the built-in server for positional data. The source code is all hosted on Github, as well as a contribution guide. This means that anyone can copy the code, implement their idea and submit a pull request to improve the software.

A large improvement to the current workflow would be automated build containers and linting. For example, a Docker container could be used to build the executable and publish a new release of the software every time it is updated. A container could also be used to create a PDF version of the wiki, to bundle with the executable. Linting can be done as a commit-hook, which is executed every time new code is submitted to the repository. This will ensure consistency in the code base.

The overall documentation should also be updated to include a proper workflow guide for the GUI. As the result suggested from the user test (see Section 10.2) this part was the main lacking information.

References

[1] Mohamed Sami. Software Development Life Cycle Models and Methodologies - Mohamed Sami. `https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/`, 2012. Accessed: 2019-05-28.

[2] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems, 2007.

[3] Davide Dardari, Pau Closas, and Petar M. Djuric. Indoor tracking: Theory, methods, and technologies. *IEEE Transactions on Vehicular Technology*, 64(4):1263–1278, 2015.

[4] Dragan Stojanović and Natalija Stojanović. Indoor Localization and Tracking: Methods, Technologies and Research Challenges. *Facta Universitatis, Series: Automatic Control and Robotics*, 13(1):57–72, 2014.

[5] SteamVR Tracking - Triad Semiconductor. `https://www.triadsemi.com/steamvr-tracking/`. Accessed: 2019-02-28.

[6] Diederick C. Niehorster, Li Li, and Markus Lappe. The accuracy and precision of position and orientation tracking in the HTC vive virtual reality system for scientific research. *i-Perception*, 8(3):1–23, 2017.

[7] Ethan Luckett. A Quantitative Evaluation of the HTC Vive for Virtual Reality Research. (May), 2018.

[8] Miguel Borges, Andrew Symington, Brian Coltin, Trey Smith, and Rodrigo Ventura. HTC Vive : Analysis and Accuracy Improvement. *IEEE International Conference on Robot and Systems*, pages 2610–2615, 2018.

[9] OpenVR SDK. `https://github.com/ValveSoftware/openvr/blob/02bc73b9bcfe9cc2d2802bd5fdc91f724de8ef10/headers/openvr.h#?L184`, 2019.

[10] Christopher Bruns. PyOpenVr. `https://github.com/cmbruns/pyopenvr`, 2019.

[11] Triad Openvr. `https://github.com/TriadSemi/triad{_}openvr`, 2019.

[12] Pragma statements supported by SQLite. `https://www.sqlite.org/pragma.html`. Accessed: 2019-05-29.

[13] Siemens Tecnomatix Plant Simulation | Engineering USA. `https://www.engusa.com/en/product/siemens-tecnomatix-plant-simulation`. Accessed: 2019-05-31.

[14] HTC VIVE Tracker (2018) Developer Guidelines Ver. 1.0 Version Control. Technical report, 2018.

[15] time — Time access and conversions — Python 3.7.3 documentation. Accessed: 2019-05-30.

A Appendix - Project Formulation

A.1 Bill of Materials

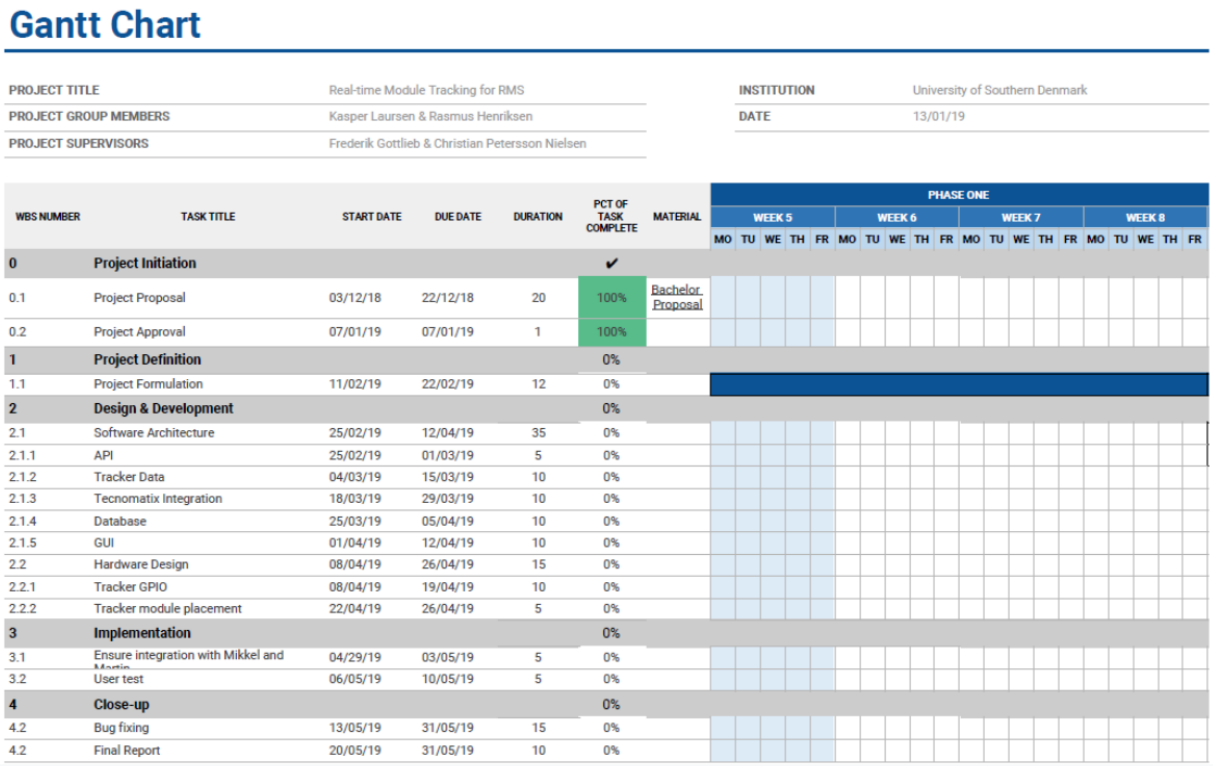
Component list

- HTC Vive VR System - https://www.vive.com/eu/product/
- HTC Vive Tracker - https://www.vive.com/eu/accessory/

Component	**Amount**	**Price** (Eur)
HTC Vive VR system	1	599.99
HTC Vive tracker	5	119.99
total		1199.94

Table 2: Bill of Materials

A.2 Gantt Chart



Gantt Chart

PROJECT TITLE	Real-time Module Tracking for RMS	INSTITUTION	University of Southern Denmark
PROJECT GROUP MEMBERS	Kasper Laursen & Rasmus Henriksen	DATE	13/01/19
PROJECT SUPERVISORS	Frederik Gottlieb & Christian Petersson Nielsen		

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	MATERIAL
0	**Project Initiation**				✔	
0.1	Project Proposal	03/12/18	22/12/18	20	100%	Bachelor Proposal
0.2	Project Approval	07/01/19	07/01/19	1	100%	
1	**Project Definition**				0%	
1.1	Project Formulation	11/02/19	22/02/19	12	0%	
2	**Design & Development**				0%	
2.1	Software Architecture	25/02/19	12/04/19	35	0%	
2.1.1	API	25/02/19	01/03/19	5	0%	
2.1.2	Tracker Data	04/03/19	15/03/19	10	0%	
2.1.3	Tecnomatix Integration	18/03/19	29/03/19	10	0%	
2.1.4	Database	25/03/19	05/04/19	10	0%	
2.1.5	GUI	01/04/19	12/04/19	10	0%	
2.2	Hardware Design	08/04/19	26/04/19	15	0%	
2.2.1	Tracker GPIO	08/04/19	19/04/19	10	0%	
2.2.2	Tracker module placement	22/04/19	26/04/19	5	0%	
3	**Implementation**				0%	
3.1	Ensure integration with Mikkel and Martin	04/29/19	03/05/19	5	0%	
3.2	User test	06/05/19	10/05/19	5	0%	
4	**Close-up**				0%	
4.2	Bug fixing	13/05/19	31/05/19	15	0%	
4.2	Final Report	20/05/19	31/05/19	10	0%	

Figure 24: Gantt Chart Part 1

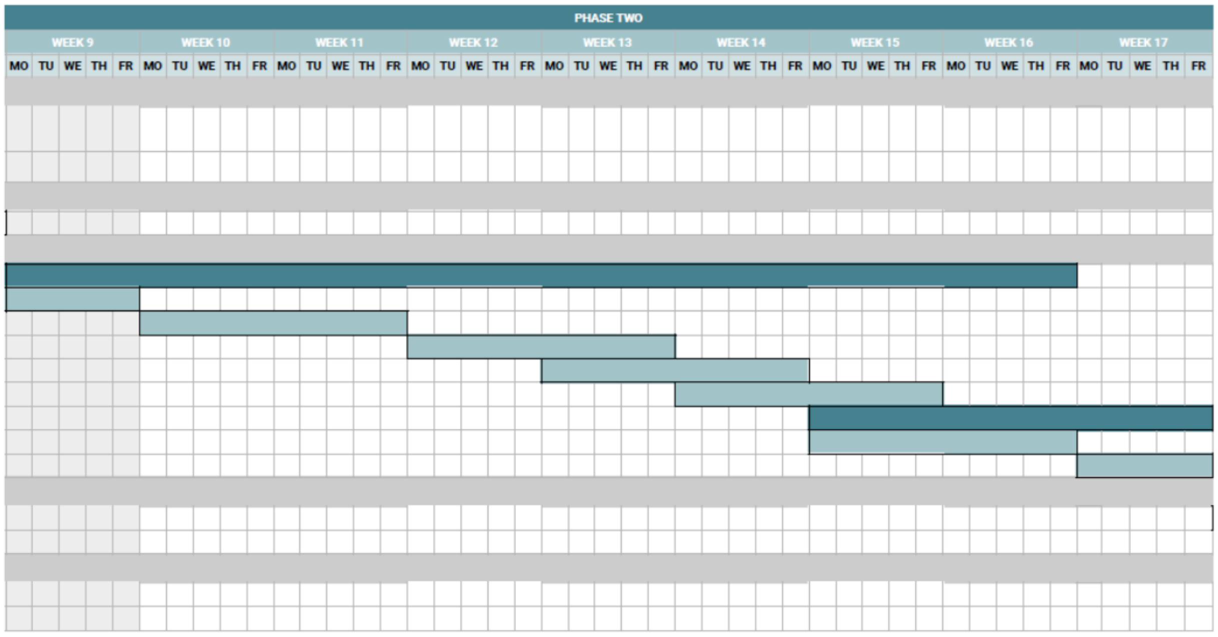


Figure 25: Gantt Chart Part 2

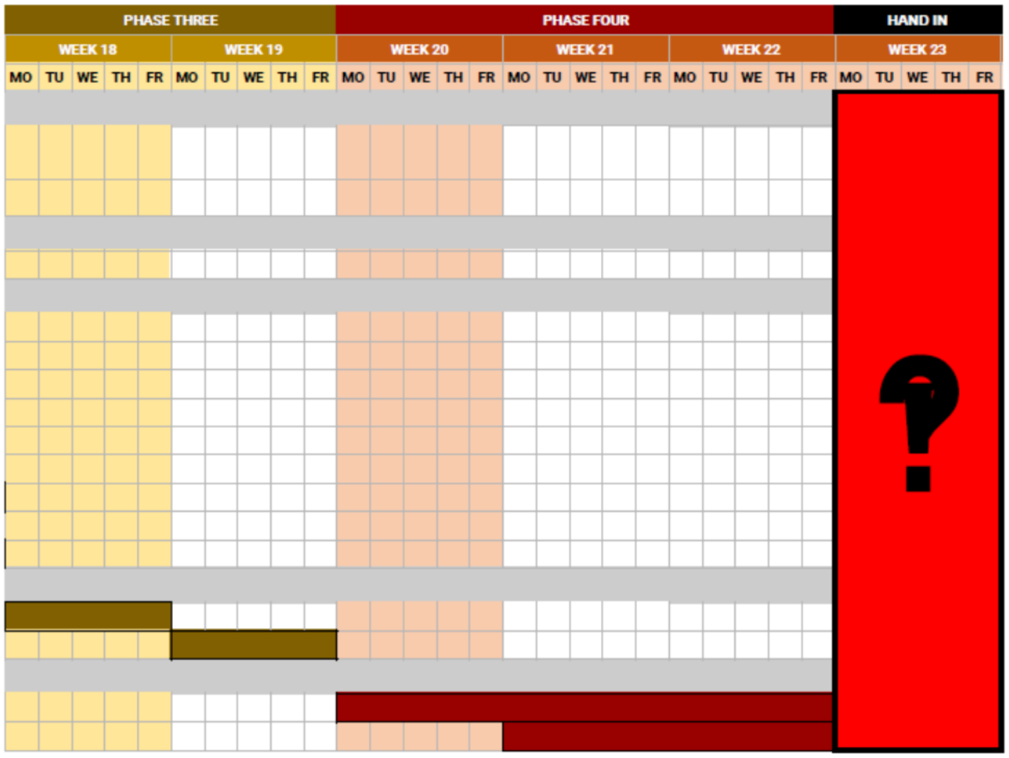


Figure 26: Gantt Chart Part 3

B Appendix - Software

All source code for this project can be found at https://github.com/Madmalicius/RMTRMS. For exam purposes, the files can be found in the attached zip file.

C Appendix - Testing

C.1 Stationary position

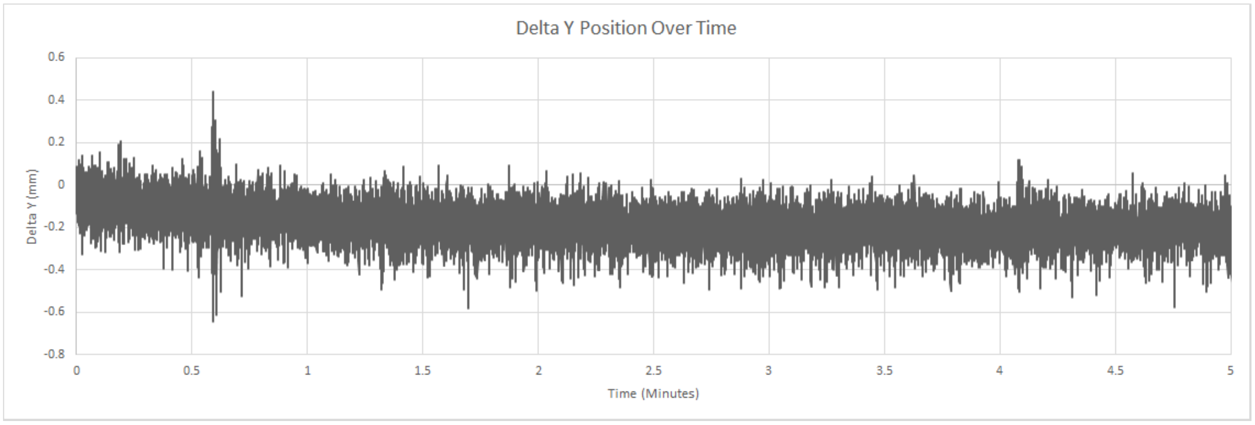


Figure 27: Graph of Change in Y Position over Time

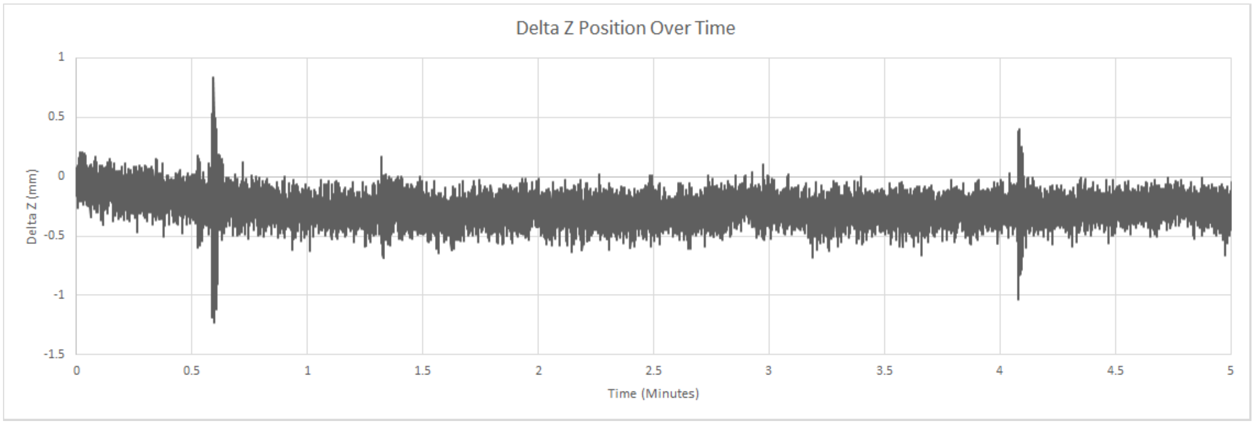


Figure 28: Graph of Change in Z Position over Time

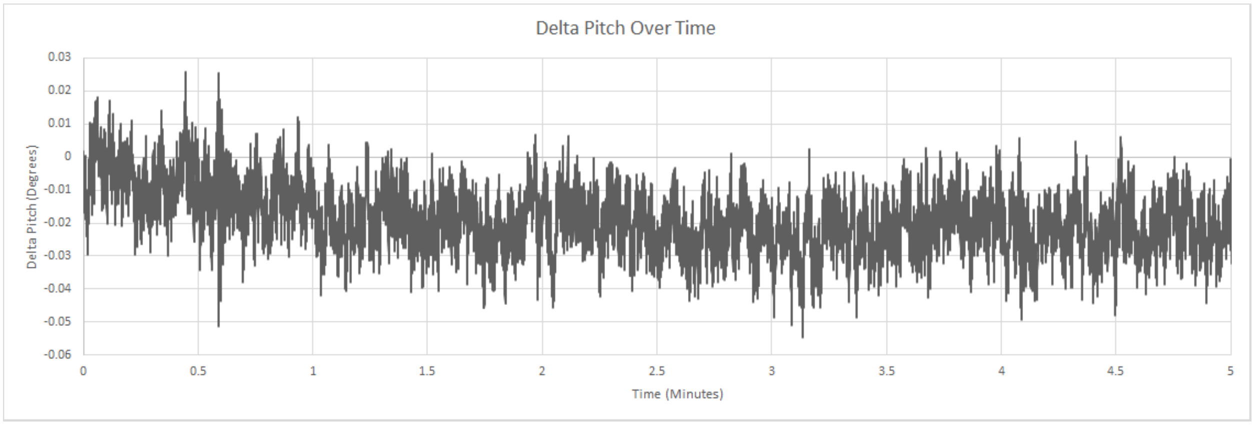


Figure 29: Graph of Change in Pitch over Time

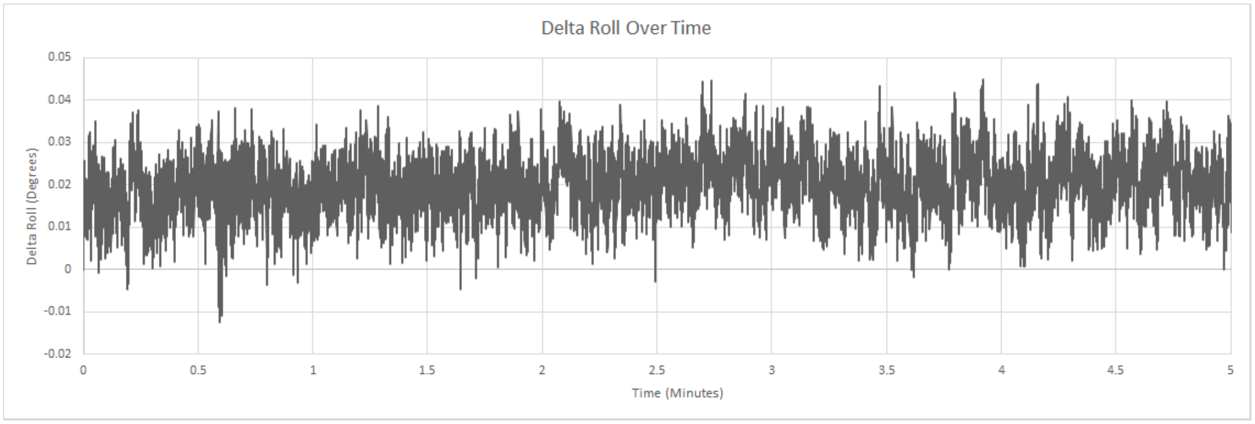


Figure 30: Graph of Change in Roll over Time

C.2 Non-stationary Position

C.2.1 500 Rotations

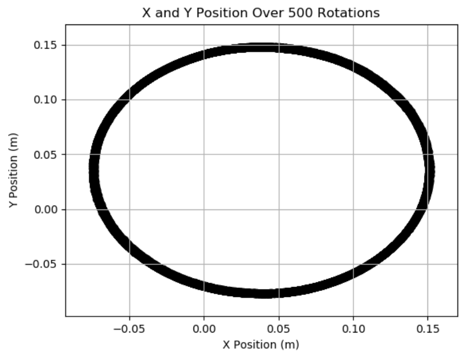


Figure 31: X and Y Position Over 500 Rotations

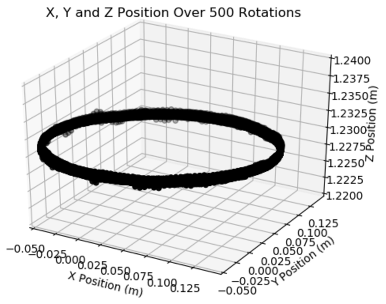


Figure 32: X, Y and Z Position Over 500 Rotations

C.2.2 Side views

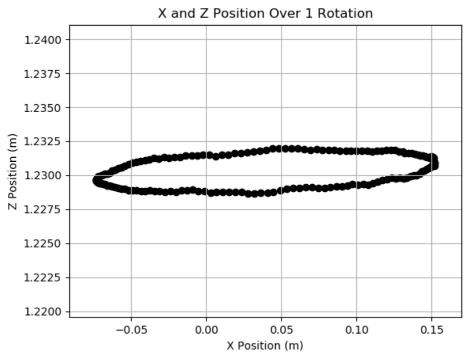


Figure 33: X and Z Position Over 1 Rotation

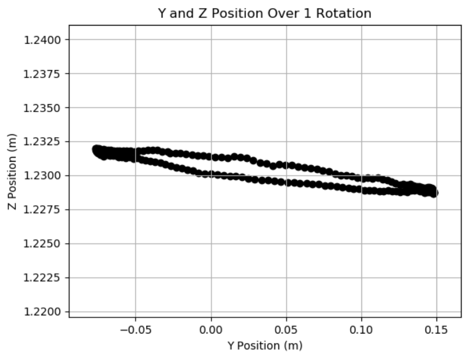


Figure 34: Y and Z Position Over 1 Rotation

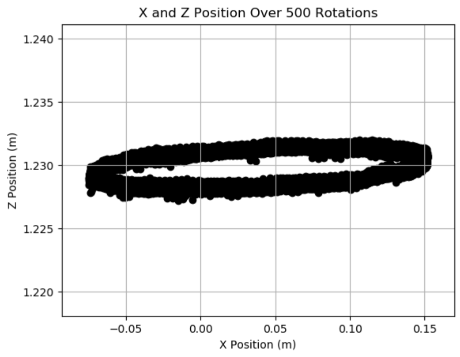


Figure 35: X and Z Position Over 500 Rotations

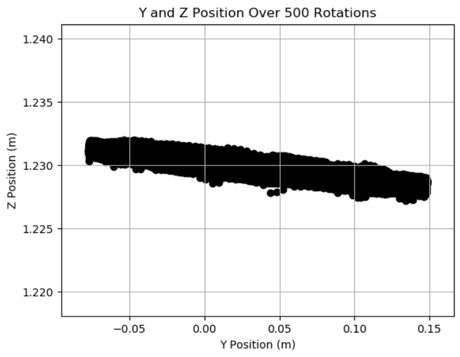


Figure 36: Y and Z Position Over 500 Rotations

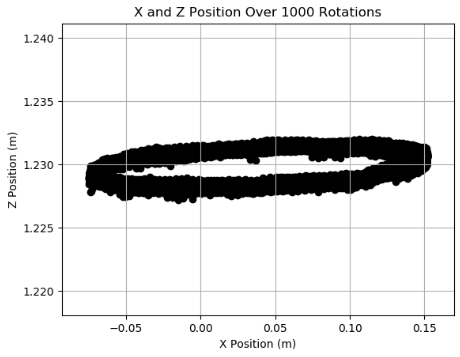


Figure 37: X and Z Position Over 1000 Rotations

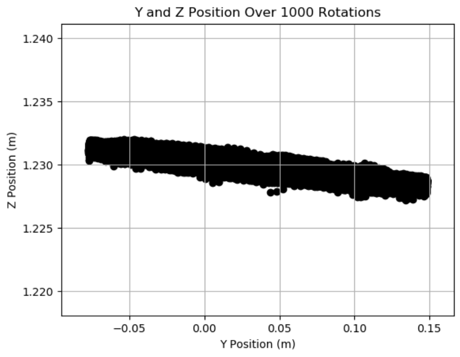


Figure 38: Y and Z Position Over 1000 Rotations

D Appendix - Documentation

Documentation

TABLE OF CONTENTS

For Developers

Setting up your workspace

Software Requirements

- Windows (10 preferably)
- Git
- Python
- SteamVR
- Siemens Tecnomatix

Installation Instructions

Your computer must have both Siemens Tecnomatics and SteamVR installed.

Students can download Siemens Tecnomatics for free from their website.

SteamVR can be found under the **Tools** section of your Steam library.

The newest version of Git for Windows can be found here.

Python should be downloaded and installed from their website. The newest version (or most versions 3.x) will work. For CMD, the path to Python might need to be added manually. A guide can be found here.

Once python is installed, use Pip to install pipenv in a shell of your choice (CMD, Git Bash, PowerShell, etc):\ `python -m pip install pipenv`

In a shell, cd into your work directory and clone this repository using Git:\ `git clone https://github.com/madmalicius/RMTRMS.git`

Navigate to the repository and to the Database folder:\ `cd RMTRMS/Database`

Install the necessary Python packages:\ `python -m pipenv sync`

Activate the virtual environment:\ `python -m pipenv shell`

Create the database by running the script:\ `python createDatabase.py`

Setting up SteamVR to run HMD-less

Triad Semi has a great guide on how to run SteamVR without a headset.

For your convenience, we've created a script to automatically set this up.:

Run SteamVR and set up your playspace as normal. After this is done, close SteamVR.

Run the python script `configure.py` found in the SteamVR folder in this repository. This will copy the settings file to your steam directory (Assuming default Steam install directory).

Alternatively, you can copy and paste the file into your Steam/config directory.

After this, you should be able to unplug the headset and box and run SteamVR using only the tracker and accompanying dongle.

NOTE\ SteamVR may give errors such as "Compositor is not running", "room setup is invalid", but these will not effect the system.

Contributing

Contributing to this project follows the standard git workflow. Create an issue in Github with your proposed feature/bug/etc and tag it appropriately. Fork the repo, make your changes (ideally in a new branch) and make a pull request. The maintainers will merge your changes.

For Users

Installation Instructions

Setting up SteamVR

To set up the SteamVR setting to run using only lighthouses and trackers, the GUI can help. Open the GUI and click *SteamVR>Configure*.

To set the SteamVR settings back to default, click *SteamVR>Restore* in the GUI.

Setting up Siemens Tecnomatix

To setup the Tecnomatix simulation, create a new 2D model with the following components:

```
<Toolbox tab>
- <Component>   : <Component Name>

Material Flow
- Source        : Source
- SingleProc    : SingleProc (Is also counted as a module)
- Drain         : Drain
(Add in more SingleProcs as modules if needed)

User Interface
- Button [ok]   : sendPositions

Information Flow
- Variable      : databasePath
- TableFile     : modules
- TableFile     : trackerPosition
- FileInterface : configFile
- SQLite        : positionDB
- Method        : INIT
- Method        : RESET
- Method        : ENDSIM
- Method        : dbOpen
- Method        : dbClose
- Method        : listModules
- Method        : moveModules
- Method        : updatePath
- Method        : updatePosition
- Method        : updateModules
- Method        : writeToPositionOut

User Objects
- Frame         : <Module Name>
(Add in more frames as modules if needed)
```

If some blocks (such as SQLite) cannot be found in the Toolbox, they can be added through *home>Model>Manage Class Library*.

An example of how the setup could looks:

The content of the method files can be found under MethodFiles.

Right-click the component and click *Rename* to name the component with the *Name* entry. Renaming can also be done under the component's settings by changing the *Name* field.

Add a connector between `Source` and `SingleProc`, and between `SingleProc` and `Drain`. Double-click `Drain` and under *Controls>Entrance* select `moveModules`.

Double-click the variable and set the data type to String under *Value>Data type*.

Right-click `sendPositions` and click open to enter settings for the button. Label is the text displayed on the button, which is recommended to set to *Send*. Under *Attributes>Control* choose *Select Object>writeToPositionOut*.

The FileInterface, `configFile`, should be connected to the config file created by the GUI. Double-click `configFile` and choose the file under *Attributes>File Name*.

Tecnomatix needs access to the computer. This is given by going to *File>Model Settings>general* and uncheck `Prohibit access to the computer`.

A new origin can be assigned by opening the 2D frame and pressing `Ctrl+R`. Then under *Axis Origin* set new X- and Y-coordinates.

Lastly, click the *Home>Navigate>Open 2D/3D* button, accept the default 3D graphics.

Run the simulation by clicking *Home>Event Controller>Start/stop Simulation*.

Using the GUI

Running the GUI gives the opportunity to manage trackers and assign trackers to modules created in Tecnomatix.

Menu Options

The GUI interface has several top menu options.

File: With subtabs *Create Database* and *Open Database*.

- Create Database: creates a new SQLite database file and sets the path to the new file.
- Open Database: load in an existing SQLite database file.

Trackers: With subtabs *Refresh* and *Manage Trackers*.

- Refresh: Reloads the list of known trackers from the database.
- Manage Trackers: Opens a new window for tracker management. See Manage Trackers Window.

SteamVR: With subtabs *Configure* and *Restore*.

- Configure: Sets up SteamVR to run in headless mode so only lighthouses and trackers are needed.
- Restore: Reverts SteamVR back to the default settings.

Help: Opens the help page on Github.

About: Opens the About page on Github.

Main Window Interface

On the main window, the list on the left side shows all modules created in Tecnomatix. This list includes *User Object Frames* and *Single Procs*.

The white label in the middle of the GUI displays the name of the selected module from the list.

Above the label, the path to the currently loaded database file is shown.

Underneath is a dropdown menu of all trackers available to be assigned to the selected module.

The checkbox on the right side of the module label determines whether or not the specific module should be tracked in Tecnomatix.

The Apply button assigns the current chosen tracker to the highlighted module along with changing the tracking status depending on the checkbox status.

Manage Trackers Window

The Manage Trackers window shows a list of all known trackers with the ability to rename or delete them from the database.

The desired name is entered into the textfield and the tracker to be renamed should be highlighted on the list to the left. The _Ok_ button saves the new name for the tracker in the database.