UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE IN COMPUTER ENGINEERING

ARTIFICIAL INTELLIGENCE AND ROBOTICS

Creating a 3D scene using a ToF camera and VR tracker

Supervisor: Prof. Carli Ruggero

Graduand: Castagna Elisa

2020377

ACADEMIC YEAR 2023 - 2024

Graduation date: 16th April 2024

Thesis developed in EuclidLabs.

Euclid Labs
3D Vision and Robotics

"To the stars who listen,

and the dreams that are answered".

Abstract

A robotic arm can perform many useful movements during its use, but attention must be paid to possible obstacles in the working environment. The aim of this thesis is to develop an application capable of mapping the robot's work area through the use of a time-of-flight camera and a tracking sensor for virtual reality.

The time-of-flight camera provides both images in two dimensions and point clouds representing the images in three-dimensional space. The collected images are used to build the panorama, through the image stitching process, using algorithms that currently represent the state of the art in the field of computer vision. Once the panorama has been reconstructed, the transformations undergone by the 2D images are estimated and applied to the point clouds to obtain the same result in the 3D world.

When merging the images, it is necessary that there are overlapping areas in common between them, to allow the stitching algorithms to construct the panorama. To increase the chance of success, the images are split based on location, which is provided by a virtual reality tracker. The virtual reality tracker is also intended to make the application portable. The camera, in fact, is mounted as an end-effector of the robot, but it is difficult to always have a real robotic arm available, unless the latter is already mounted and functioning, especially considering that the study of the robot's workspace it is a process that is performed before the final assembly of the robot and its cell. To overcome this problem, the tracker can be connected to software, which simulates the robotic arm and its movements, allowing developers to understand how to best use the manipulator, without needing the physical counterpart.

Un braccio robotico può eseguire molti movimenti utili durante il suo utilizzo, ma occorre prestare attenzione ai possibili ostacoli presenti nell’ambiente di lavoro. Lo scopo di questa tesi è quello di sviluppare un'applicazione in grado di mappare l'area di lavoro del robot attraverso l'uso di una telecamera a tempo di volo e di un sensore di tracciamento per la realtà virtuale.

La camera a tempo di volo fornisce sia immagini in due dimensioni che nuvole di punti che rappresentano le immagini nello spazio tridimensionale. Le immagini raccolte vengono poi utilizzate per costruire il panorama, attraverso il processo di image stitching, utilizzando algoritmi che rappresentano attualmente lo stato dell'arte nel campo della computer vision. Una volta ricostruito il panorama, le trasformazioni subite dalle immagini 2D vengono stimate e applicate alle nuvole di punti per ottenere lo stesso risultato nel mondo 3D.

Quando si uniscono le immagini, è necessario che ci siano delle zone di sovrapposizione in comune tra loro, per permettere agli algoritmi di stitching di costruire il panorama. Per aumentare le possibilità di successo, le immagini vengono divise in base alla posizione, fornita da un tracker di realtà virtuale. Il tracker per la realtà virtuale ha anche lo scopo di rendere l'applicazione portatile. La camera, infatti, viene montata come end-effector del robot, ma risulta difficile avere sempre a disposizione un braccio robotico vero e proprio, a meno che quest'ultimo non sia già montato e funzionante, soprattutto considerando che lo studio del workspace del robot è un processo che viene eseguito prima dell'assemblaggio finale del robot e della sua cella. Per ovviare a questo problema, il tracker può essere collegato a un software, che simula il braccio robotico e i suoi movimenti, permettendo agli sviluppatori di capire come utilizzare il manipolatore al meglio, senza aver bisogno della controparte fisica.

Introduction

Robotic arms are widely known and used in all sectors, ranging from logistics to surgery. Robots are in fact capable of execute simple task as moving and positioning objects from one place to another, and performing extremely delicate surgical operations, as in the case of the da Vinci surgical system.

The Internation Standard Organization defines an industrial robot as a *"automatically controlled, reprogrammable, multipurpose, manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications"* [1]. According to this definition the first robotic arm was created by Unimation, a company founded by George Devol and Joe Engelberger, in 1956. Together, they created the first industrial robot, called Unimate, based on Devol's patent "Programmed Article Transfer"[1]. Nowadays, the number of industrial robots working in 2022 is 3,903,633 units[2].

A robotic arm is a mechanical structure that tries to simulate a human arm and its versatility. It is composed of several consecutive rigid links, connected to each other by joints, which can be revolute (R) or prismatic (P) and together they compose the kinematic chain of the manipulator. Depending on the combination and number of these joints, you can get different robotic arms, each of which is designed and programmed to perform specific tasks, depending on the working environment, the layout of the picking location and the obstacles in its workspace.

The robotic arm can be subdivided into 3 pieces: the *arm*, which allows the manipulator to reach the desire final position in the space, the *wrist*, that provides dexterity, and the *end-effector*, which is the tool that performs the required tasks, such as grasping an object, or spraying paint. The kinematic chain connects the base of the robot, that is where the arm rests, to the end-effector.

Based on the structure of the manipulator, it is possible to determine the degree of freedom of the kinematic chain, since each joint is characterized by a degree of freedom. The number of degrees of freedom determines how the robotic arm moves in the space.

Robotic arms are mainly classified by the first three joints, that compose the arm of the manipulator. Based on the arm joints, they can be:

- Cartesian arm (PPP), that means the manipulator is composed by three consecutive prismatic joints. It is also called linear robot, since it moves in straight lines on the three

[1] International Federation of Robotics (2012). *History of industrial robots.*
[2] International Federation of Robotics (2023). *Executive Summary of World Robotics 2023 – Industrial Robots.*

different axes of the Cartesian system (x, y, z). It is widely employed in CNC machines, 3D printing, or pick and place operations. In the Figure 1, it is possible to see the kinematic scheme of the Cartesian arm:

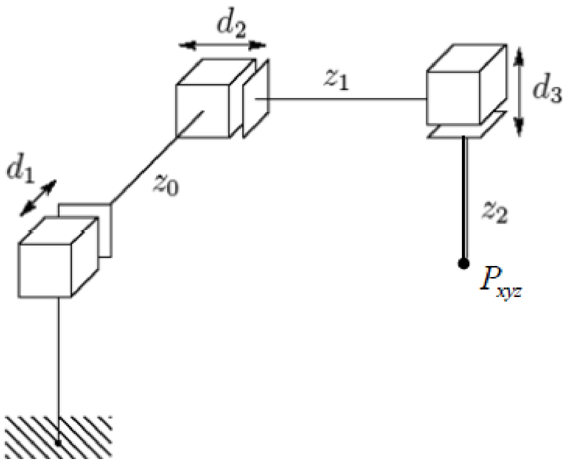


Figure 1: kinematic scheme of a Cartesian arm.[3]

- SCARA manipulator (RRP) is made up by two consecutive revolute joints and a final prismatic joint. SCARA is an acronym and stands for Selective Compliance Assembly Robot Arm. From the extended name, it is possible to understand that this robot is used for assembly and palletizing. One main characteristic of this robot is the fact that all axes of motion are parallel.
- Spherical robot (RRP) has the same geometry as the SCARA, but the axes of motion of each joint are not parallel and their combination allows the robot to obtain a system of polar coordinates.
- Articulated arm (RRR) is composed by three revolute joints, for which the axis of motion of the first joint is orthogonal to the axes of the other two joints, that are parallel to each other. This arm is also called anthropomorphic arm given that it is very similar to a human arm. It is the most used and known robotic arm, thanks to its flexibility.
- Cylindrical arm (RPP) is composed by a revolute joint and two prismatic joints. It is mainly used for assembling, machine tending, or coating applications.

In addition, there are two other types of robotic arms that are used in industry, but do not fall under the previous classification based on the joints that make up the robot arm. These are the *delta robot*, which is a type of parallel robot, and the *collaborative robot*. The delta robot is a

[3] *Image source*: https://commons.wikimedia.org/wiki/File:Cinem%C3%A0tica_robot_cartesi%C3%A0.svg

robot composed of three arms, connected at the base by universal joints. This formation allows the end-effector to always be oriented in the same way. It is a particularly fast robot and suitable for handling light loads, in fact it is mainly used in pick and place operations. The collaborative robot, also called a *corobot*, on the other hand, is a robot that shares the workspace with humans, in stark contrast to the industrial use of the robot itself. This sharing of the workspace entails much more stringent constraints in terms of force and speed of movements in order to avoid harming its user.

Another important aspect to know about a manipulator is its workspace, that is defined as the space that end-effector attached to the robotic arm can reach and is defined by the physical constraints on the joints and the geometry of the robotic arm. This means that for each robotic arm, we can calculate its workspace by knowing which joints compose it and what the mechanical limits of each joint are. The workspace of a robot can be subdivided into two subsets: the *reachable* workspace and the *dexterous* workspace. The *reachable workspace* is the space that the robotic arm can reach, meanwhile the *dexterous workspace* is the space that can be reached by the end-effector without losing degrees of freedom. Everything within the reachable workspace must be correctly evaluated and avoided during the movement of the arm, otherwise the arm may collide and be damaged.

When talking about robotic arms, or more in general about robotics, it easy to understand that there is a strong connection with *computer vision*, because it provides the ability to robots to see and understand their surroundings. Computer vision is a part of artificial intelligence that tries to understand, analyse, and process images. Following the computational power increasing in 1970, computer vision started to improve drastically, thanks to the ability and possibility of using more and more images, so more algorithms were developed, such as the Hough transform [2] in 1972, or the Viola-Jones face detection algorithm [3] in 2004. With the advent of neural networks, a revolution in computer vision can be observed. Deep learning and convolutional neural networks (CNNs) have allowed computers to recognize objects and patterns within images, significantly improving facial and object recognition. Indeed, in 2015, a CNNs was able to outperform a human in the *ImageNet Large Scale Visual Recognition Competition* (ILSVRC), achieving an error-rate of 3.57%, while the human error rate was set to 4% [4]. The future of computer vision is related to *augmented* and *virtual reality*. Meanwhile augmented reality combines together the real world with computer-generated object, virtual reality (VR) completely substitutes the real world with a simulated one. Virtual reality has been developed since the early 1950s, starting with *Sensorama* (patented 1962)[4], developed by Morton Heilig, which was one of the first VR machine that stimulated all senses and provided to users a full cinema experience. Heilig was also the first to invent the VR Head Mounted Display (HDM), creating the *"Telesphere Mask"* (patented 1960)[5], a device described as *"a telescopic television apparatus for individual use... The spectator is given a complete sensation of reality, i.e. moving three dimensional images which may be in colour, with 100% peripheral vision, binaural sound, scents, and air breezes."* from its patent. These days, VR is widely used in different fields and different scopes, such as in video games, 3D cinema or social virtual world for entertainment, or in the medical field, where VR is used for treating anxiety disorders or post-traumatic stress disorders (PTSD) [5].

To access to the virtual reality word, it is necessary to have a virtual reality headset, which is a device that a user can wear, and it is compose by a stereoscopic display, stereo sound, and sensors for tracking the pose of user's head. For interacting with virtual word, users can use motions controllers, that is an input device that use accelerometers, gyroscopes, and cameras to track the position, or optical sensors for motion capture, which provide the position by using special markers that have to be attached to the body or wired gloves. All these controllers are tracked using cameras, in order to not have wires.

The aim of this thesis is to develop an application that can map the robot workspace and create a relative 3D scene. The presence of obstacles inside the workspace must be highlighted to allow the robot controller to find the best trajectory for avoiding all obstacles, so creating a 3D scene of the surrounding of the robot allows the final user to search for obstacles and extract important details such as positions and size of the obstacle.

For doing this, three main tools were used: a IFM time-of-flight camera, a virtual reality tracker, and OpenCV. The IFM ToF camera provided all the information related to the spatial position of what there is in its field view, creating a 2D image and point cloud each time was necessary to collect data. These data were stored, in order to be processed at a later time. The virtual reality tracker provided the position in which the IFM ToF camera was when collecting data, but most important the portability of this application. As a matter of fact, the ToF camera was mounted as a robot end-effector, and using a real robotic arm was not a suitable option, since the application is mainly developed to be used before the installation of the robotic arm. This means that it is necessary to simulate the robotic arm, and this is possible by using the virtual reality

[4] Heilig M. (1962). *US Patent #3,050,870*.
[5] Heilig M. (1960). *US Patent #2,955,156*.

tracker. The tracker provided information to a software that is able to display robotic arms and their movements, based on how the tracker is moved, allowing programmers to understand which motions are feasible and which are not. Using images acquired by the IFM camera, it was required to construct a panorama. For the construction of the panoramic image, OpenCV was used, which is a library that implements state-of-the-art algorithms for real-time computer vision. OpenCV is cross-platform and licensed as free and open-source under Apache License 2.

Study of virtual reality trackers

Virtual reality (VR) is a simulated experience where the real world is substitute with a computer generated one, using pose tracking and stereoscopic display for showing the world and its change. VR is widely used in several applications, such as video games, medical training, virtual meetings, space, and military industries.

For having the possibility of access and interact in a virtual reality world, it is necessary to have a proper setup. This setup requires a virtual reality headset, composed by a stereoscopic display, stereo sound, and sensors for tracking the pose, typically two controllers (one for each hand), and one or more stereo cameras, as shown in Figure 2.

Figure 2: the Valve Index VR Kit, consisting of a headset, two controllers, and two cameras.

A stereoscopic display is a particular display able to create the illusion of depth by stereopsis for binocular vision, which means that two 2D images are provided to the user, and, when these two images are perceived by the human brain as a single 3D image, providing the illusion of depth. For the sound part, the stereo sound method is implemented, which is a method that can give the illusion of listening sounds from various directions. For tracking the pose of the headset and changing what users can see according to their movements, it is used accelerometers and gyroscopes, which provides information about changing in direction, orientation, and speed. The controllers can be various types, such as wired gloves, motion controllers, or optical tracking sensors, and they are chosen based on the use of virtual reality. Stereo cameras, meanwhile, are used for achieving wired free controlling for localization and navigation.

For this thesis, a virtual reality tracker was used. The tracker had to provide the current position of the IFM camera, to which it was attached, position that was stored together with the image and the point cloud for following processing. The tracker also provided information about the motion of the robotic arm and for understanding how the manipulator was moving according to the motion of the virtual reality tracker, it was used a software called MARVIN [6], developed by EuclidLabs. This software can simulate how a robotic arm has to move to obtain the same trajectory of the virtual reality tracker.

Thanks to the increasing interest in virtual and augmented reality, different companies developed their own tracking products, having different shapes, costs, and possible functions. The first step, when trying to create an application based on data received from a tracker, is to determine which tracker is more suitable for the contest. In this specific case, it was decided to analyse how coherent are the data received by trackers with respect to what a robotic arm has done, and it was decided to test 3 different sensors: Tundra Tracker, Valve Index Controller, and HTC Vive Tracker. All these sensors worked with the Valve Index Base Station [7] for tracking the position and orientation changes during their use, this means that all sensors support the SteamVR Version 2.0 Tracking.

For each sensor, three different tests were executed, trying to determine the drift of the system, the noise error, and the ability of the sensor to track a linear motion performed by the robotic arm. From previous experiments done in EuclidLabs, it was discovered that the system suffers of a drift error and, statistically, it required at least 30 minutes to reach a stable point. Since these previous tests were not performed using these three sensors, the first step was to determine how long it took to the system, using these new trackers, to reach a stable point and provide coherent position. When the required time was discovered, it was necessary to determine how noisy these trackers are. To accurately collect this information, it was necessary to minimize interference, so the data provided were transmitted to the computer using wires. Lastly, it was measured how good a dynamic motion was registered, and it was decided to perform the simplest dynamic motion possible, a linear movement. It was measured by placing the tracking sensor on the flange of the robotic arm and performing a linear motion between two points with the manipulator. For this test, it was always used the same robotic arm, in order to avoid any possible change due to using a different manipulator.

Since this virtual reality system uses camera to track sensors, it was natural to change the number of cameras tracking a sensor and their positions, in order to obtain results that can represent different situation, where this system is used.

Sensors: Tundra Tracker, Valve Index Controller, HTC Vive Tracker 3.0

As previously mentioned, the sensors used were Tundra Tracker, Valve Index Controller, and HTC Vive Tracker 3.0, in combination with Valve Index Base Station for tracking their position and orientation. All these trackers work using the SteamVR tracking system [8].

The *Tundra Tracker* [9] is developed by Tundra Labs, a US based company, founded in 2018. The Tundra tracker was released in November 2021, and it is a small and light device designed to be worn more easily, but it can be also mounted on objects in order to track them. It allows both cabled, using a type-C cable, and wireless connection. The wireless connection is achieved using the Super Wireless Dongle, created by Tundra Labs, but it can work also with Dongle for VIVE Tracker, Dongle inside headset of HTC VIVE series and Valve Index, and eteeDongle for SteamVR. Its dimensions are 50.2 x 50.2 x 38.5 mm, with a weight of approximately 50 g, and with a battery of 850 mAh the Tundra tracker can work for up to 9 hours. It is equipped with 18 sensors, shown in Figure 3, which cannot be covered.

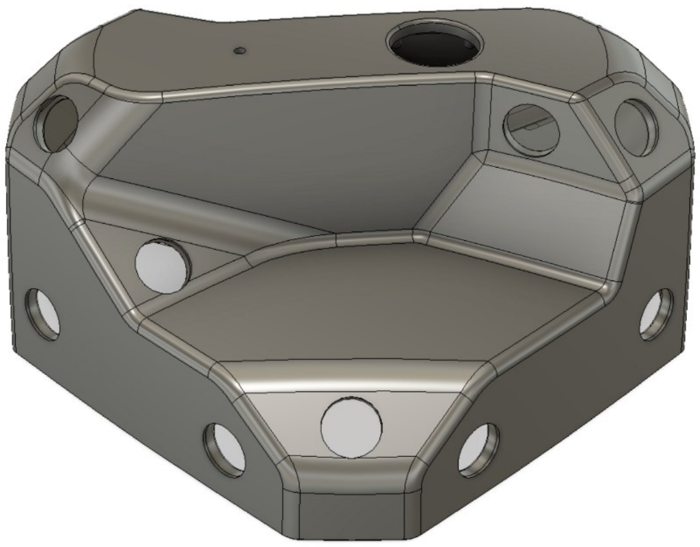


Figure 3: sensors placed on Tundra Tracker[6]

The *Valve Index Controller* [10] is developed by Valve Corporation, founded in 1996 by Gabe Newell and Mike Harrington. It was released in June 2019, with the relative headset Valve Index, which it was intended to be used with, but it can be used with the majority of the headset on the market. The controller, shown in Figure 4, is usually provided in a couple, since it was

[6] *Image source:* https://docs.tundra-labs.com/tundra-tracker/tracker-hardware-specifications

created to simulate all hands behaviours and interact with the virtual world the same way it is possible to interact in the real one using hands. In addition to providing location using Steam VR's tracking system, each controller is equipped with several buttons, specifically an A button, a B button, thumbstick, touchpad, a menu button, a trigger, and, using an array of 87 sensors, it is possible to track motion and pressure of the hand, creating a precisely representation of the user's hand. Using all of these sensors, the controller can tell when an object is being thrown, squeezed, or crushed. The Valve Index Controller provides both types of connection: USB-C or 2.4GHz Wireless. With a battery of 1100 mAh and a weight of 196g, it is quite large, with respect to the Tundra Tracker.

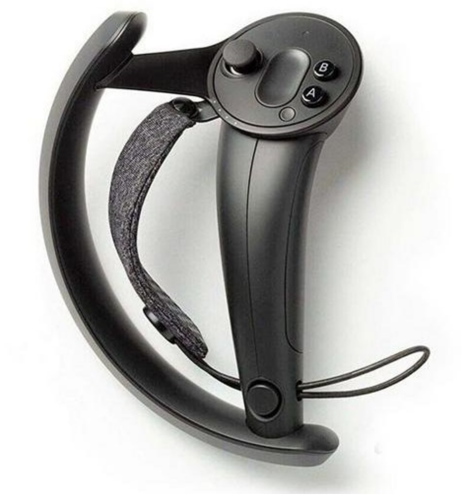


Figure 4: the Valve Index Controller[7]

The last used sensor was the *HTC Vive Tracker 3.0* [11], shown in Figure 5, developed by HTC, a Taiwanese company founded by Peter Chou and Cher Wang in 1997. It was released in March 2021, and it is a device created for a full-body tracking experience. It is 15% lighter and 33% smaller than its previous version, released in 2018, indeed it weights 75g and its dimensions are 70.9 × 79.0 × 44.1mm. It is equipped with a battery that can last up to seven hours, which is up to 75% more than the 2018 version. It provides both cable and cableless connectivity, being compatible with Vive Tracker Dongle and Tundra Labs Super Wireless Dongle.

[7] *Image source:* https://www.gamestop.com/pc-gaming/pc-gaming-controllers/products/valve-index-controllers-left-only/11206126-11205170.html

Figure 5: the HTC Vive Tracker 3.0[8]

In conclusions, these sensors are the latest version, and they are all compatible with the Steam VR Tracking system. The main difference between these sensors is that HTC Vive Tracker 3.0 and Tundra Tracker are both designed and developed to be worn or mounted on an object, while Valve Index Controller is designed to reconstruct the experience of using the hands in the virtual world, going to build the ability to throw or crush objects simply by decreasing or increasing pressure on the controller. In a gaming environment, the difference between controllers that only need to be worn and those that allow interactions must be carefully evaluated to choose the best tracker.

For this thesis, however, it was necessary to determine which tracking sensor was the most precise in providing the position during movements, recreating the same trajectory performed by the tracker, even in the virtual world. According to this aim, choosing a wearable or an interactive tracker was indifferent since it was required to choose the most precise.

Steam VR Tracking System

All these trackers used the Steam VR Tracking system [8], which is the software and hardware technology that has the assignment to provide the 6 DoF pose of the sensor. The Steam VR Tracking technology uses the outside-in method. In this method, cameras are placed in a fixed position among the location, and they track the position of sensors placed on each tracker. Increasing the number of cameras provides a better reading of the position, as overlaps will be

[8] *Image source:* https://www.vive.com/media/filer_public/fed-assets/tracker3/images/lighter-992-tracker.webp

created which allow the final position of the tracker to be better triangulated. There are some disadvantages to using the outside-in system. First of all, the fact that it is very easy for cameras to lose the tracked sensor due to occlusion. If during a motion a sensor is shaded to one or more cameras, then it can jump to a new position when recorded again by the cameras, creating a bad experience. The other main limitation of this system is the limited space in which sensors are tracked. The working zone of a camera is limited due to the fixed position of the base station, which means that changing environment is not possible without shutting down the system. Another problem for outside-in tracking systems is reflective lights, due to the interferences created by this light to cameras. If the system is working near a window or a reflective surface, it is recommended to cover the surface for a better experience.

The Steam VR Tracking is a hardware and software technology that provides actual pose of an object by tracking the sensors placed on it. It uses 3 main components: base station, sensors, and host. The base station is a 120° multi-axis laser emitter for the HTC Vive Base Station 1.0 [12] and a 160º x 115º multi-axis laser emitter for the Valve Index Base Station 2.0 [7]. The sensors placed on trackers are ASIC sensors, lightweight and low power, in particular they are photodiodes. They are placed on the object such that more sensors can be seen when moving the object, as it is possible to see in Figure 6, where ASIC chips are placed along the Vive Controller's ring, which is the part of the controller that cannot be covered.

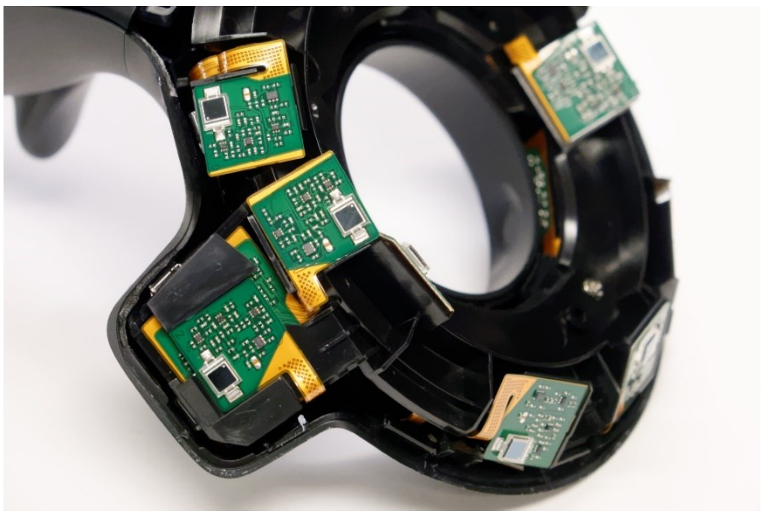


Figure 6: portion of the Vive's controller ring.[9]

[9] *Image source:* https://pcper.com/wp-content/uploads/2016/04/0932-160404-122447.jpg

Other than photodiodes, there is an inertial measurement unit (IMU) with an update of 1000Hz to provide information about acceleration with a low latency. The host is the computer that provides the Steam VR API interface for precise timing, synchronization, and prediction of position.

For obtaining the position of a tracker, each base station emits different lasers, one vertical and one horizontal, and synch pulses scanning the room. Every time the synch pulse is detected by the sensors on the tracker, these ones start to count, since they know that the laser beam is scanning the room. Based on how much time is passed between the synch pulse and the laser signal, the system is able to calculate, using simple trigonometry, where the sensor is placed in the two-dimensional plane. This is possible because the Steam VR software contains a template for each headset, controller, or tracker, which specifies where the photosensors needed for tracking are placed. By having more points, the system can add the third coordinate, angle, and rotation of the object. By using the IMU, the Steam VR Tracking technology is able to understand linear and angular velocity of the tracked object, with an update frequency of 1000Hz.

With the new base station, the Valve Index Base Station 2.0, it is incremented the field of view of the cameras. It is usually recommended to use at least two cameras, but if there are obstructions, it is possible to add a base station in order to get rid of them, and, by adding a fourth one, it is possible to obtain a room of 10x10m. The laser used in this base station fires 100 times per second and it is individually coded, such that the base station can coexist with other infrared devices, avoiding interferences. Up to know, the new base station is compatible with Valve Index e HTC Vive Pro [13].

Executed tests

The sensor study was composed of three different tests, each of which was performed for each sensor. First, the drift of the system was studied. From previous tests carried out, it was discovered that the system suffered from a drift when it was turned on, which resulted in a continuous movement of the position received from Steam VR, independent of the movement of the tracker. It was necessary to understand if the time needed at the system to stabilize was the same for each sensor, or if each sensor required a different stabilization time. The second test carried out was to determine how noisy each tracker is, to understand how to counterbalance this error and how much it affected the received positions. The third test, however, was related to the execution of a simple movement by the tracking sensor and the ability of the tracking system to correctly perceive and track this movement. This test was aimed at understanding how large the cylinder was that contained the final movement perceived by the sensor.

When using the Steam VR Tracking system, the positioning of base stations and the presence of reflective surfaces can significantly affect the data obtained from trackers. In fact, in the guide to the use of base stations and their positioning, it is suggested to cover reflective surfaces and position the stereo cameras so that they are almost never obstructed. Another factor of fundamental importance is the number of cameras that are used for tracking. The greater the number of base stations, the better the precision of the tracked position. During the execution of these tests, the behaviour of the system was also studied, changing the position and number of cameras available for position detection, in order to understand with how many base stations it was possible to obtain a good result and what the best positioning was. For each measurement, the sensor being tested was secured to the flange of the robot, a Kuka KR QUANTEC nano [14]. It was chosen to place the sensor on the robot to prevent it from suffering vibrations deriving from the surrounding environment. Furthermore, the robot was maneuvered and positioned, even during the dynamic test, so that the stereo cameras were never obstructed. Only in configuration 2, the sensor was not positioned on the robot, but on a raised support, as the room used was a closed and protected environment.

Different configurations were set up for the positioning of the cameras, each of which was maintained throughout the entire test run for all trackers studied. The configurations chosen were:

- *CONFIGURATION 1*: 3 cameras were used, placing the cameras on the left, right and top, all on the same wall. The robotic arm that housed the sensor under test was located in the center of the chambers.
- *CONFIGURATION 2*: 2 cameras were used, positioned on the opposite side of the room and at different heights. In this configuration the room was completely dark, to allow studying in a situation where there was no reflective surface. The sensor was positioned in the center of the two cameras, resting on a support.
- *CONFIGURATION 3*: 2 cameras were used, positioned on the left and right on the same wall, at two different heights, in particular 160cm for the left camera, and 180cm for the right camera. The robotic arm was positioned in the center of the two cameras.
- *CONFIGURATION 4*: 2 cameras were used, positioned left and right on the same wall, at the same height of 257 cm. In this setup, two different room sizes were set during the Steam VR setup, the large room, and the small room, to see if the room size setting was

causing problems for the system. Also in this case, the robotic arm was positioned in the center.

For each configuration, you could easily change the number of cameras available, simply by turning the cameras on or off, until the desired number was obtained. All recordings were performed using the MARVIN software, which through the API provided by Steam VR [15] is able to receive the position from the tracker and display it in its 3D world.

Drift experiments

System drift was studied by placing the sensor in the center of the base stations' work area and recording data for approximately one hour. During the recording, the sensor was not moved and there was no obstruction of the cameras. The location was received every second. The drift present in the Steam VR tracking system was studied both when the system was turned on, to calculate the time needed for the system itself to stabilize, and when the system was already turned on and stabilized, to ensure that the drift was a problem solely derived from the turning on the system.

To determine the validity or otherwise of the system, once the positions collected during the test were received, the error on these positions was calculated for each coordinate X, Y, Z, A, B, and C in accordance with the Equation 1:

$$positionError = recordedValue - averageValue$$

Equation 1: computation of the position error for each coordinate.

Once the error for each coordinate was calculated, the absolute error was computed according to Equation 2:

$$error = \sqrt{positionErrorX^2 + positionErrorY^2 + positionErrorZ^2}$$

Equation 2: absolute error

This error was then brought into a graph, in which the abscissa line represents the time, expressed in seconds, while the ordinate represents the calculated error, expressed in millimetres, i.e. how much the position deviates from the average value.

The expected output graphs had to be of two types, in one case having the system just turned on and in another case the system already stabilized. In the case of the newly powered system, the output graph was expected to have an initial fluctuation in position, due to system drift, followed by a complete stabilization after approximately 30 minutes. In the case, however, of the system already switched on and stabilized, the expected graph consisted of a fluctuation of the position error around zero.

Tundra Tracker

To study drift with the Tundra tracker, it was decided to record data using configurations 1 and 2, varying the number of cameras used in each different recording.

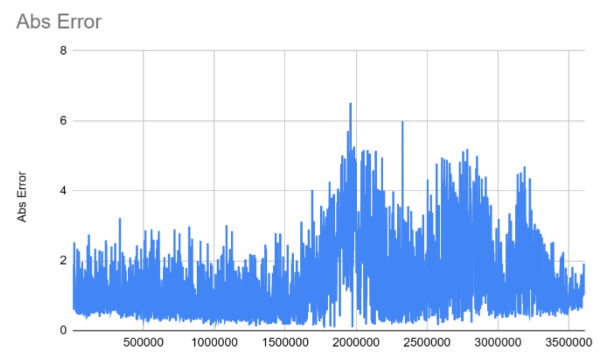
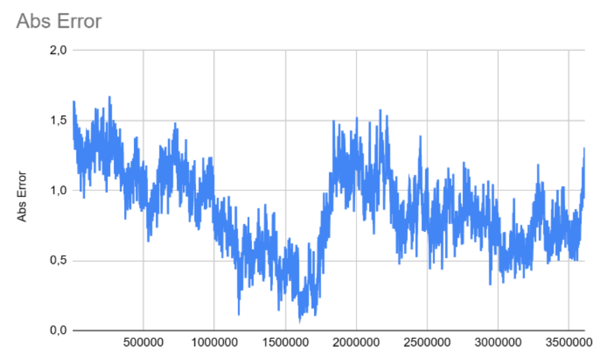
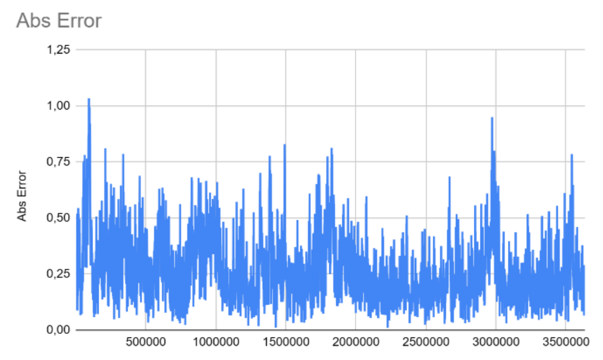
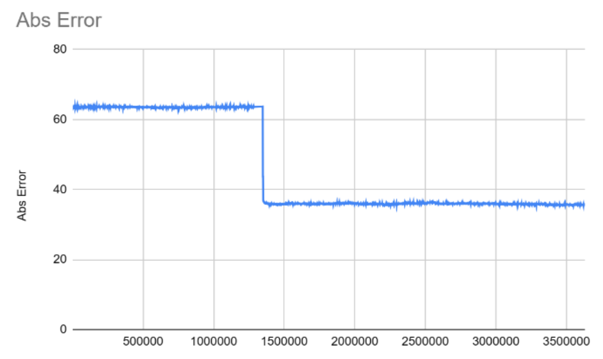
For *configuration 1*, data was collected 4 times, including once with the system just turned on.

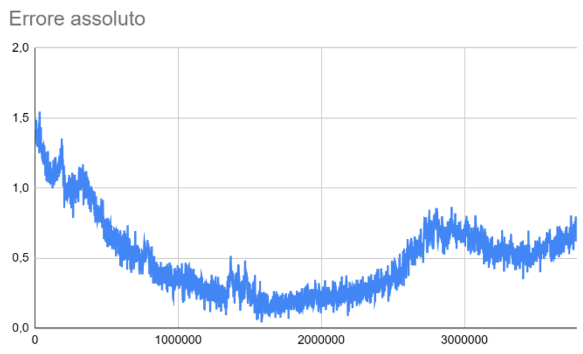
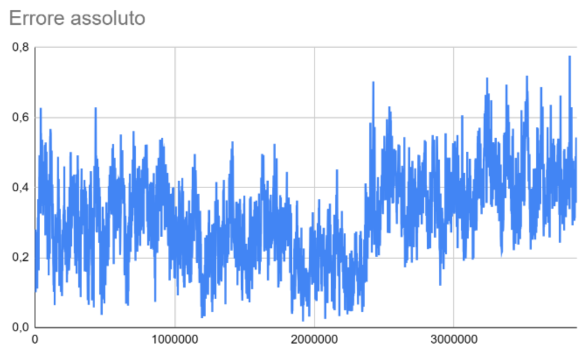
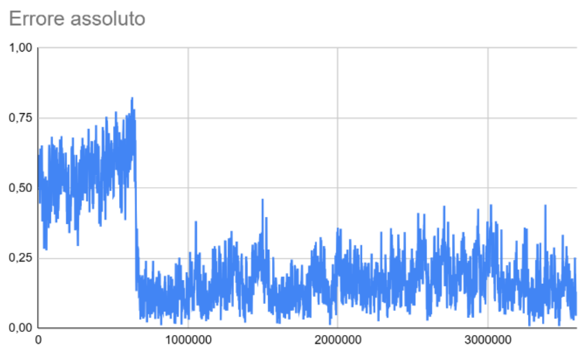
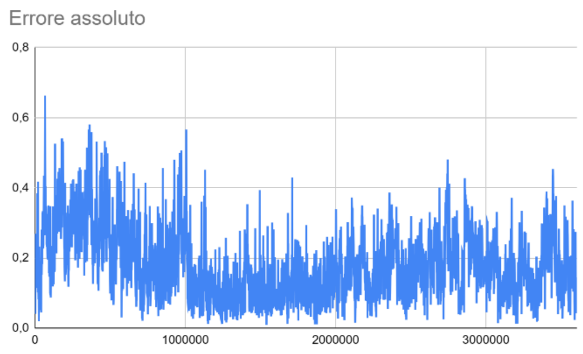
For *configuration 2*, the data was collected 11 times, 2 of which when the system had just been turned on. In this configuration it happened to obstruct the cameras on two separate occasions, which can be seen in the absolute error graphs, which, in fact, show a jump when the cameras resumed free vision.

In the Table 1**Errore. L'origine riferimento non è stata trovata.**, you can see the absolute error graphs obtained from when measuring drift with the Tundra Tracker. It is reported the worst result in case of multiple registration with the same parameters. Furthermore, the coordinate that produced the largest standard deviation was also reported, also specifying the relative value.

Table 1: drift absolute error of Tundra Tracker.

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]

Configuration 1 – registration only with the right camera		Y coordinate: 1,237580705
Configuration 1 – registration with left and right cameras.		X coordinate: 0,6976120504
Configuration 1 – registration with all 3 cameras.		Y coordinate: 0,2025278112
Configuration 1 – registration with all 3 cameras, but the system was just turned on.		C coordinate: 148,9005751

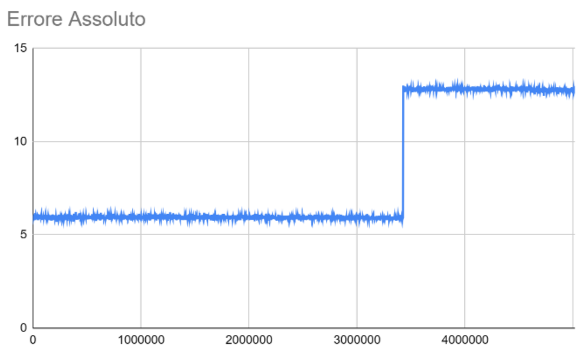
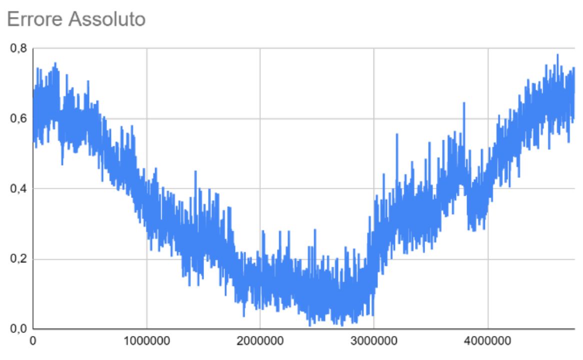
Configuration 2 – system just turned on, worst case.		Y coordinate: 0,4028559344
Configuration 2 – obstruction of a camera.		X coordinate: 0,2078487083
Configuration 2 – obstruction of a camera.		Z coordinate: 0,1802598721
Configuration 2 – registration with both cameras.		Y coordinate: 0,124494943

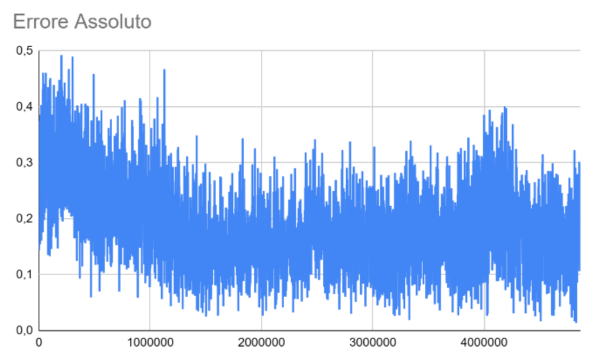
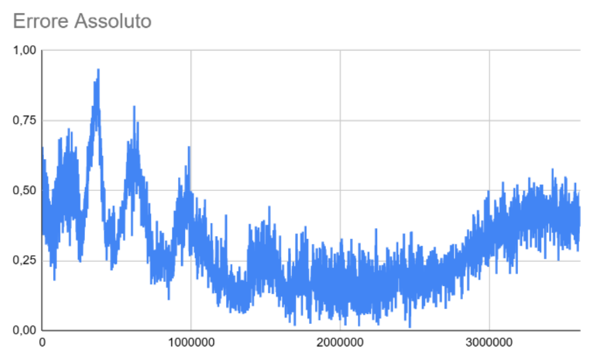
As you can see, the Tundra Tracker behaves as expected, with the exception made in configuration 1 when the system was just turned on. In this case, there was not a drift, but an initial stability of the system at a specific point of the virtual world, followed by a sudden and sudden jump to another random point of the virtual 3D world, followed by the final stability of the system. The same sudden jump that occurred when the system was just turned on also occurred when one of the two cameras in configuration 2 was momentarily obscured. In this case, however, the system did not undergo major variations and the jump remained limited to a variation of a few millimetres, which, however, did not happen in the previous case study, where the jump led to a variation of 20mm in terms of absolute error. As expected from Steam VR, by increasing the number of base stations, the accuracy of the system is increased.

Valve Index Controller

For the Valve Index Controller, drift was only studied in *configuration 1*, for which a total of four recordings were made, one of which was performed when the system was turned on. The results obtained are shown in the Table 2, where you can observe the graph of the absolute error and the coordinate with the greatest standard deviation and the relative value.

Table 2: drift absolute error of Valve Index Controller.

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]
Configuration 1 – registration with the right camera.		Y coordinate: 7,278858865
Configuration 1 – registration with left and right camera.		X coordinate: 0,2601223028

Configuration 1 – registration with 3 cameras.		X coordinate: 0,1445116567
Configuration 1 – registration with 3 cameras and system just turned on.		Z coordinate: 0,2530527207

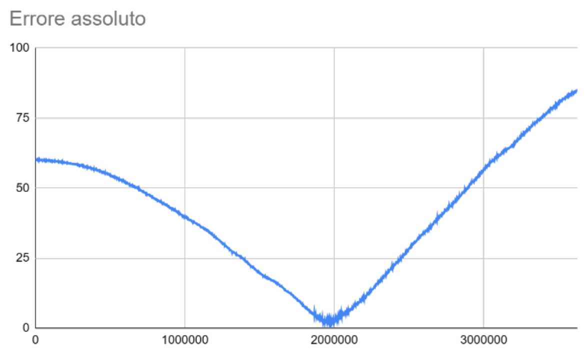
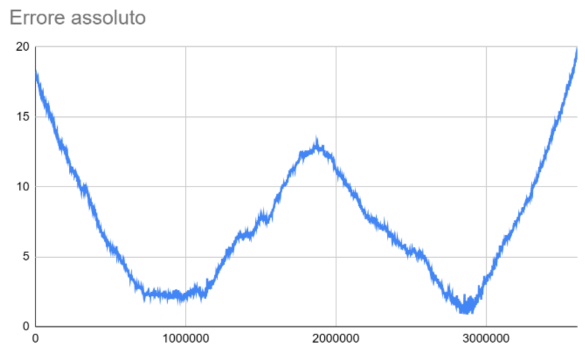
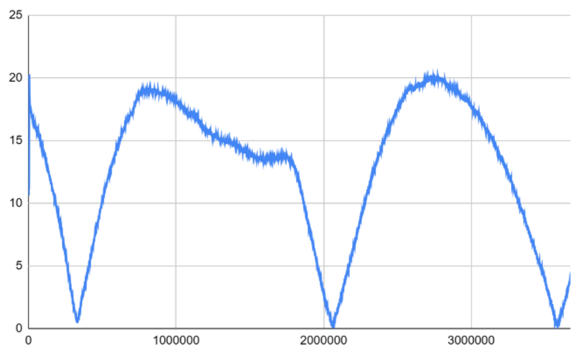
Also in this case, the presence of a jump was noted, which occurred without an obstruction of the cameras. Aside from this anomaly, the Valve Index Controller behaved as expected, decreasing the absolute position error as the number of base stations used increased. Drift was observed when the system was turned on, followed by stabilization.

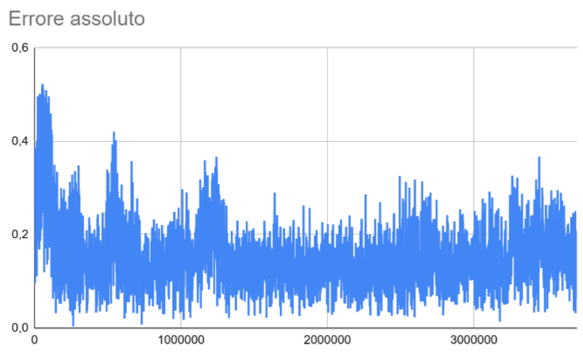
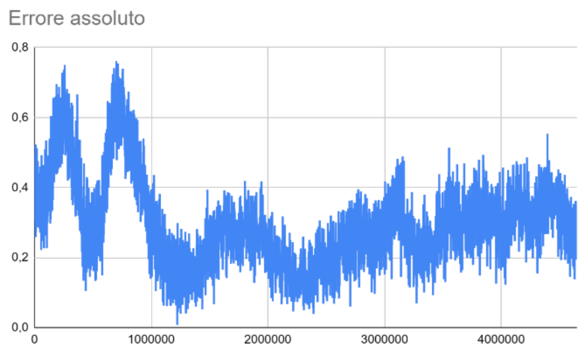
HTC Vive Tracker 3.0

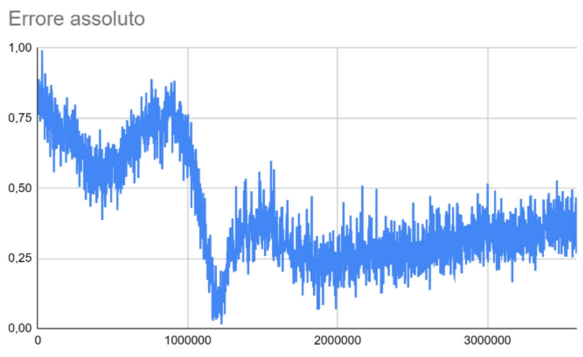
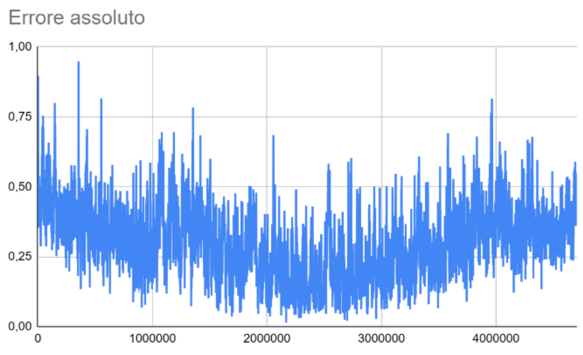
The study of the drift of this tracker turned out to be more problematic than the two already studied. In *configuration 1*, locations were collected 5 times, 2 of which were when the system was turned on. The tracker, in this configuration, worked in the opposite way to what was expected. The best results, in fact, were obtained as soon as the system was turned on, while the number of base stations used was irrelevant for a possible improvement of the system. For *configuration 2*, however, the data was collected 11 times and on 2 occasions the system had just been turned on. In this configuration, the results were consistent with what was expected, reporting very small absolute errors.

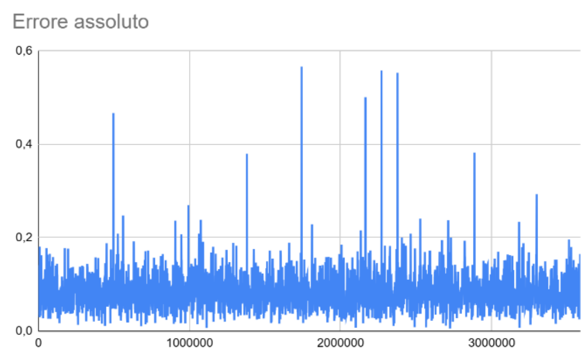
The Table 3 shows the graphs of the absolute error and the coordinate with maximum standard deviation.

Table 3: drift absolute error of HTC Vive Tracker 3.0.

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]
Configuration 1 – registration with the right camera.		X coordinate: 40,58165879
Configuration 1 – registration with right and left camera		X coordinate: 7,399572384
Configuration 1 – registration with 3 cameras		X coordinate: 12,15862595

Configuration 1 – registration with 3 cameras and system just turned on.		X coordinate: 0,1138090438
Configuration 1 – registration with 3 cameras and system just turned on – second attempt.		Z coordinate: 0,2652803129

Configuration 2 – system just turned on (worst case)		Y coordinate: 0,3178720278
Configuration 2 – registration with 2 cameras (worst case).		Z coordinate: 0,2169840993

Configuration 2 – registration with 2 cameras (best case).		X coordinate: 0,05835696222

In Configuration 1, the HTC Vive Tracker 3.0 produced unexpected results. In fact, it was not possible to determine the reason for the tracker's behaviour, despite several attempts having been made, each time excluding a possible cause. Potential reflective surfaces, which did not cause any problems in the other tests, were covered and the measurements were repeated. Any devices that could cause interference were also removed, again without meaningful results. On the contrary, the system was much more stable and with a truly reduced error at startup. In one case, the system did not experience the well-known drift, while in the second attempt the drift was present, but quickly stabilized.

Reflective surfaces being the biggest problem with the Steam VR tracking system, it was decided to further evaluate the system in a room completely devoid of light, i.e. configuration 2. In this configuration, the system behaved as expected and produced excellent results. However, this did not allow us to determine with absolute certainty the origin of the strange behaviours observed in configuration 1.

Noise experiments

The goal of the noise experiment was to determine how noisy each sensor was, in order to provide the most accurate measurement possible. To avoid interference from other devices, it was decided to use a cable to transmit data from the sensor to the PC and not to exploit the wireless connection with which the sensors are equipped. Each sensor was positioned in the center of the cameras' workspace and secured to the robot's flange. The location was recorded for five minutes, with an update rate of one millisecond.

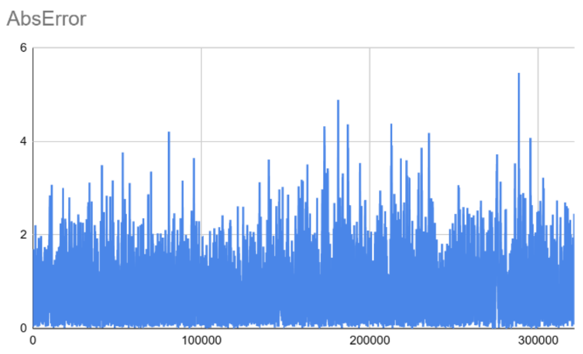
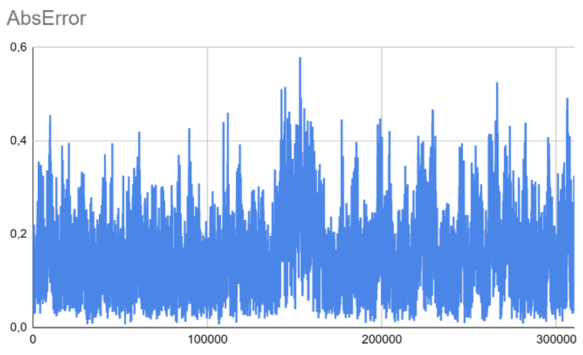
The noise experiment was conducted when the system was fully operational and not when the system had just been turned on, to avoid running into the drift problem. The error relative to each coordinate was calculated according to Equation 1, while the absolute error was calculated according to Equation 2. The expected absolute error trend is characterized by a set of constant values close to zero.

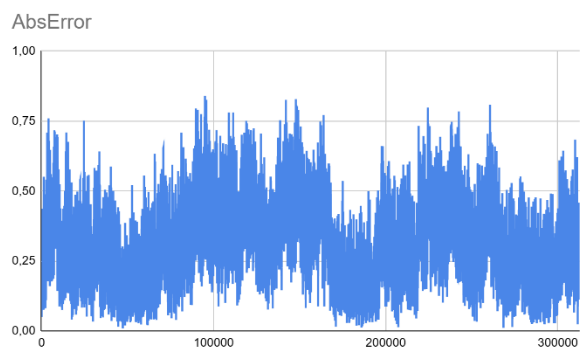
Tundra Tracker

For the Tundra Tracker, the measurements were all performed in *configuration 1*, recording with a different number of base stations, in order to understand how the noise changed in relation to an increase in cameras. Measurements were performed twice for each set of cameras used, for a total of 6 recordings overall.

The Table 4 shows the graphs of the error obtained, always considering the worst case, i.e. the case in which the standard deviation of the absolute error was the largest. The coordinate with the greatest standard deviation is also reported.

Table 4: noise absolute error of Tundra Tracker

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]
Configuration 1 – registration with right camera.		Y coordinate: 0,7913909286
Configuration 1 – registration with left and right cameras.		Y coordinate: 0,1150919967

Configuration 1 – registration with all 3 cameras.		Y coordinate: 0,227001771

From the results, we can see how the transition from one to two base stations contributed to a decrease in the absolute error, going from an average absolute error of 0.8773684662mm for the positions recorded with a single base station to an average error of 0,152208148mm for the positions received using two base stations. By further increasing the number of cameras, no significant decrease in noise was found, in fact the absolute average error increased, obtaining an average absolute error of 0,3050001074mm, always considering the worst case between the two recording.

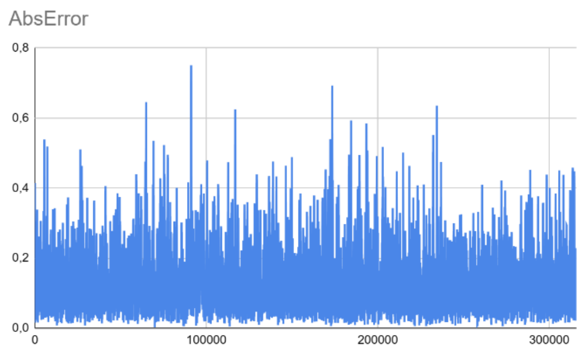
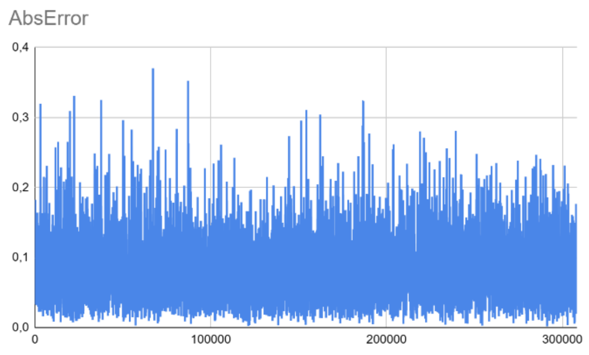
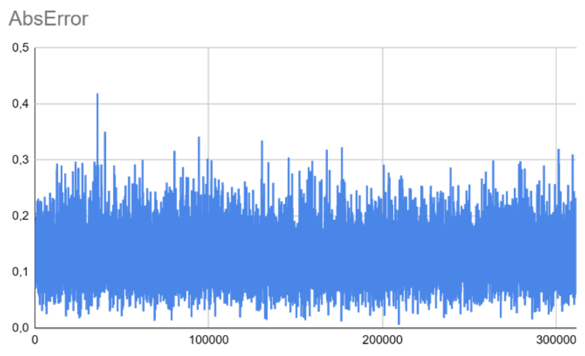
Valve Index Controller

Also for the Valve Index Controller, the noise experiment was repeated 6 times, twice for each number of working cameras, always in *configuration 1*.

The Table 5 shows the graph representing the absolute error and the coordinate having the greatest standard deviation.

Table 5: noise absolute error of Valve Index Controller.

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]

Configuration 1 – registration done with the camera on the right.		Y coordinate: 0,1136903042
Configuration 1 – registration done with left and right cameras.		X coordinate: 0,06116572452
Configuration 1 – registration with all cameras.		X coordinate: 0,09650580398

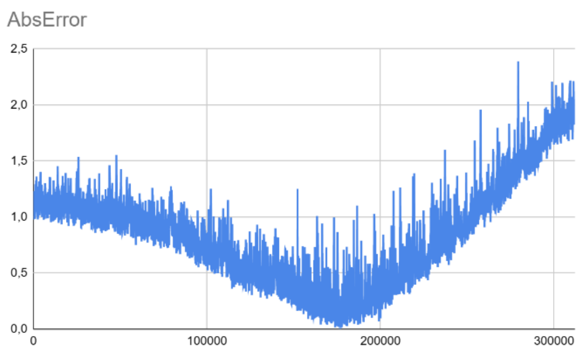
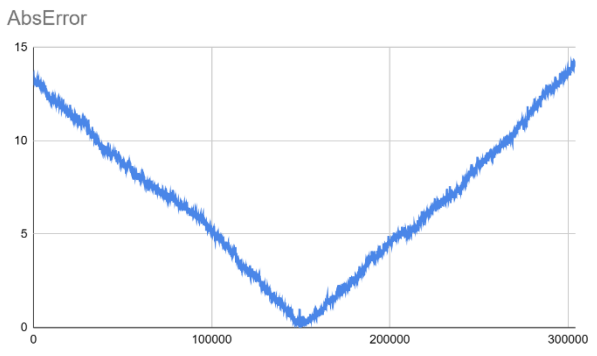
The Valve Index Controller was shown to be quite indifferent to the number of base stations used for position tracking. In fact, the best result is obtained using two base stations, with an average error of 0.08112156402mm, while with one and three cameras, the average absolute error is around 0.13mm.

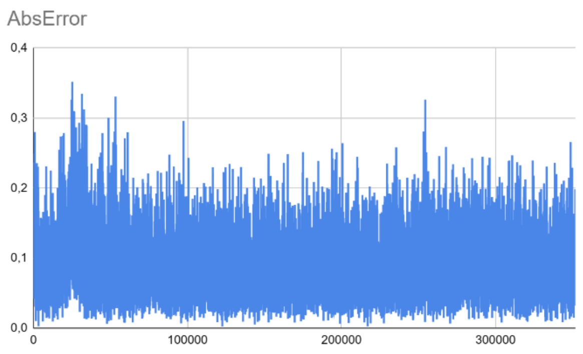
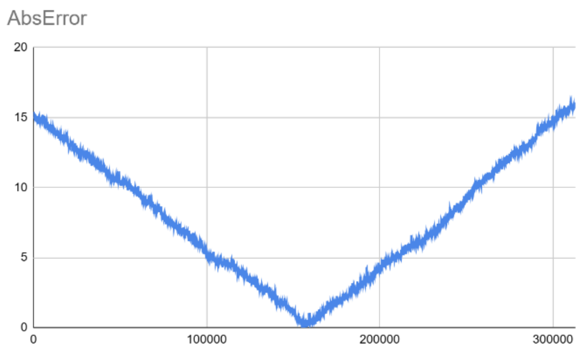
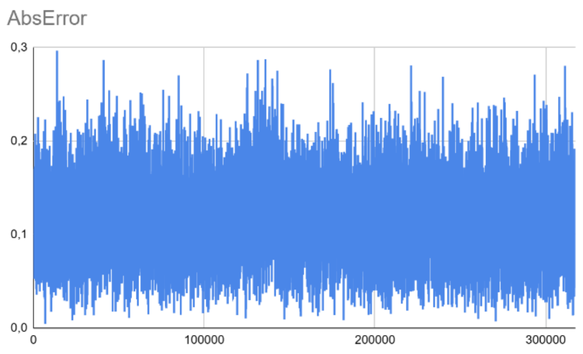
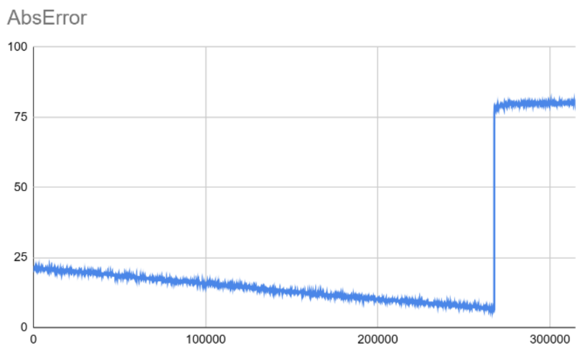
HTC Vive Tracker 3.0

Several recordings were made for the HTC VIVE Tracker 3.0, as behaviour was noted that differed from what was expected. It was in fact possible to observe on MARVIN the sensor which, although it was completely still and anchored to the robot, moved, or jumped in complete autonomy in the virtual world. A first hypothesis as to the reason for this behaviour of the tracker was the interference of the IFM time-of-flight camera, used during nearby recording, but subsequent tests showed that the IFM did not cause any interference, since good results, even while the camera was working and was deliberately pointed at both the base stations and the tracker.

In total, the data was collected 13 times, all in *configuration 1*, and the graphs of the trend of the absolute error in the worst and best cases, based on the maximum and minimum standard deviation, are provided in the Table 6 together with the coordinate that reached the maximum standard deviation.

Table 6: noise absolute error of HTC Vive Tracker 3.0.

CONFIGURATION	ABSOLUTE ERROR [mm]	MAXIMUM STANDARD DEVIATION [mm]
Configuration 1 – registration done with the camera on the right – best case.		Y coordinate: 0,7953497516
Configuration 1 – registration done with the camera on the right – worst case.		Y coordinate: 7,12638882

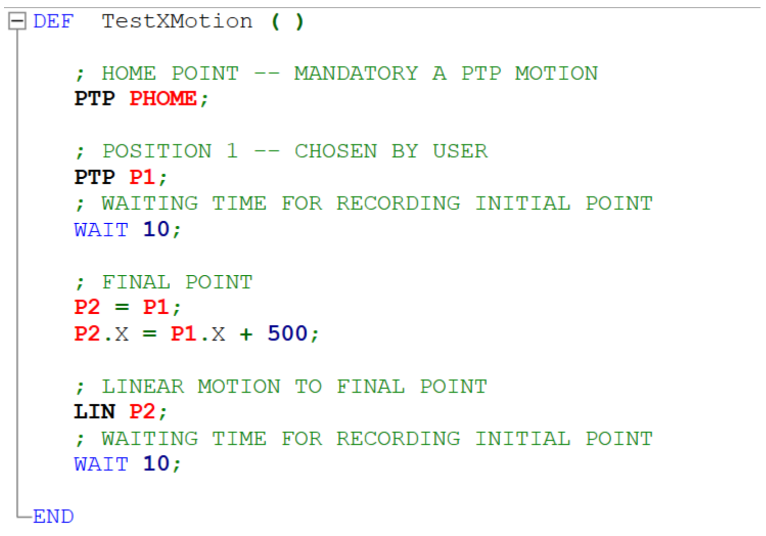
Configuration 1 – registration done with the right and left camera – best case.		X coordinate: 0,07084194781
Configuration 1 – registration done with the right and left camera – worst case.		Y coordinate: 8,2794865
Configuration 1 – registration done with all cameras – best case.		X coordinate: 0,07565882987
Configuration 1 – registration done with all cameras – worst case.		X coordinate: 29,33615729

The performance of the HTC Vive Tracker 3.0 led to inconclusive results. In fact, in some recordings, as can be seen from the graphs in which the absolute error is not an accumulation of points close to zero, but the values form a v, the sensor moved autonomously in the virtual world, as if something was moving it moving. Unfortunately, despite various attempts, it has not been possible to determine the causes of this shift. In fact, reflective surfaces, interference due to other devices, such as the IFM used during recordings, and an insufficient number of base stations were excluded. When this visible movement on MARVIN did not occur, the HTC Vive Tracker 3.0 performed noteworthy. In fact, with two and three cameras, the average absolute error was 0.09149734072mm and 0.1166392708mm respectively.

Dynamic experiments

The last aspect studied about these three sensors was their ability to record linear movement. The main purpose of the dynamic test, in fact, was to determine the ability of the sensors to faithfully reproduce the linear movement performed by the robot and to calculate the radius of the cylinder containing the movement. Ideally, the radius of this cylinder should be zero, as the sensor should identically reproduce what it experienced in the real world in the virtual one. Each sensor was placed on the flange of the robotic arm, which performed three linear movements, one for each Cartesian axis. It is important to underline that the movement was linear with regards to the robot's reference system, which is not guaranteed by the sensor. In fact, the reference system of the robot and the one that the sensor has in the world generated by MARVIN are completely independent of each other. Another feature studied in these dynamic tests was the sensor's ability to correctly record the distance travelled. If, in fact, the robotic arm had performed a linear movement of one meter, the distance between the initial and final position of the sensor should be at a distance in space of one meter, because that is the distance travelled in the real world.

The tests were performed by repeating the same loop, changing only the axis of movement of the robotic arm. Initially, the sensor remained stationary at the starting point for approximately 5 to 10 seconds, linear movement was performed, followed by another stop of 5 to 10 seconds. In this way, it was possible to recover the initial and final point of the straight line travelled in the virtual world, calculating the average of the first and last 200 points recorded. The code used to make the robot move is shown in the Figure 7.



```
DEF  TestXMotion ( )

   ; HOME POINT -- MANDATORY A PTP MOTION
   PTP PHOME;

   ; POSITION 1 -- CHOSEN BY USER
   PTP P1;
   ; WAITING TIME FOR RECORDING INITIAL POINT
   WAIT 10;

   ; FINAL POINT
   P2 = P1;
   P2.X = P1.X + 500;

   ; LINEAR MOTION TO FINAL POINT
   LIN P2;
   ; WAITING TIME FOR RECORDING INITIAL POINT
   WAIT 10;

END
```

Figure 7: code for X motion.

In the evaluation of the dynamic tests, two errors were evaluated: the distance of each individual position with respect to the straight line followed in the linear movement and the length of this straight line. Remembering that there is no correlation between the frame the robot used, and the frame used by the sensor in the virtual world, all calculations were performed in three dimensions. As regards the distance $d(\cdot,\cdot)$ between the straight line r and each individual position P_i, it was calculated according to Equation 3, where $P_1 = (x_1, y_1, z_1)$ is any point on the line r and it was decided to use the final point, $\boldsymbol{n}$ is defined as $\boldsymbol{n} = \boldsymbol{m} \times (\boldsymbol{m} \times \overrightarrow{P_iP_1})$, and $\boldsymbol{m}$ is defined as $\boldsymbol{m} = \langle a, b, c\rangle$, where a, b, c are the coefficient of the linear equation of line r.

$$d(P_i, r) = \frac{\left|\overrightarrow{P_iP_1} \times \boldsymbol{n}\right|}{\|\boldsymbol{n}\|}$$

Equation 3: distance between a point and a line in space.

The distance $d(\cdot,\cdot)$ between the initial measured point $A = (x_A, y_B, z_B)$ and final measured points $B = (x_B, y_B, z_B)$ was calculated according to Equation 4.

$$d(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

Equation 4: distance between point A and point B in a 3-dimensional space.

The expected graph representing the distance of each individual position with respect to the line travelled should have had a bell shape, in fact during the first 5/10 seconds of the test the robot was stationary, followed by a linear movement towards the final point, where it would have restated for another 5/10 seconds. The recordings made when the robotic arm was stationary should have produced an almost zero distance from the line. The bell shape, however, should have reached its peak towards the middle of the line, because the robotic arm performed an acceleration up to the halfway point, followed by a deceleration to reach the final point and stop. Since the sensors are equipped with IMU and the position calculated by Steam VR also based on the data received from the IMU, it was expected that the acceleration and deceleration would affect the recorded position, creating the bell-shaped graph.

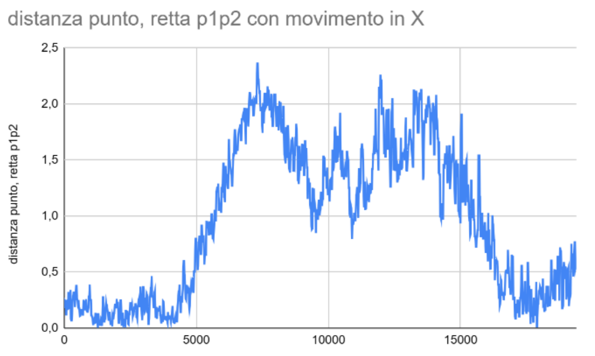
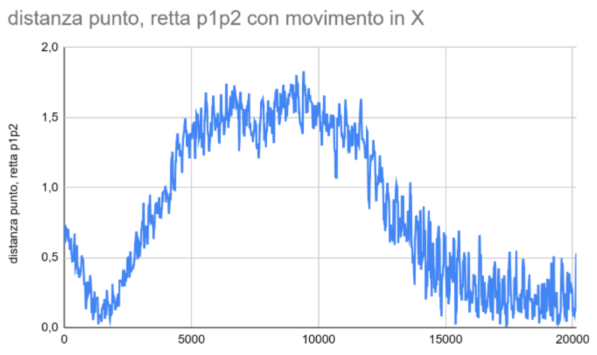
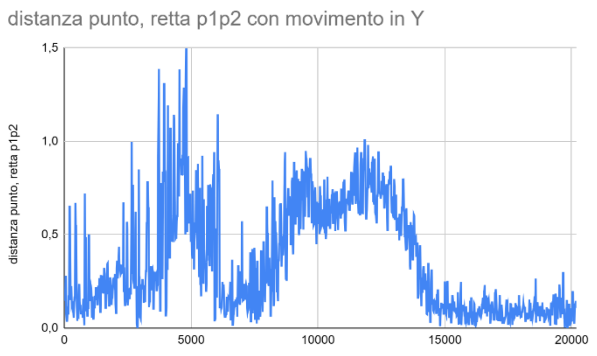
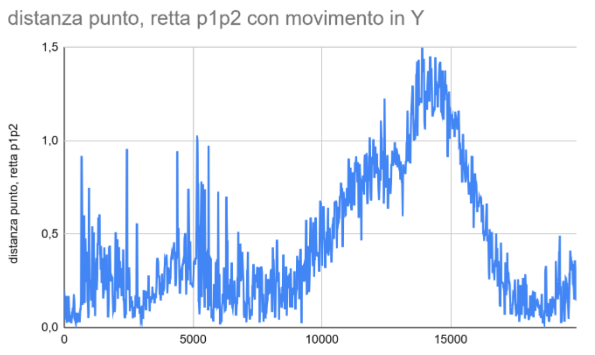
Tundra Tracker

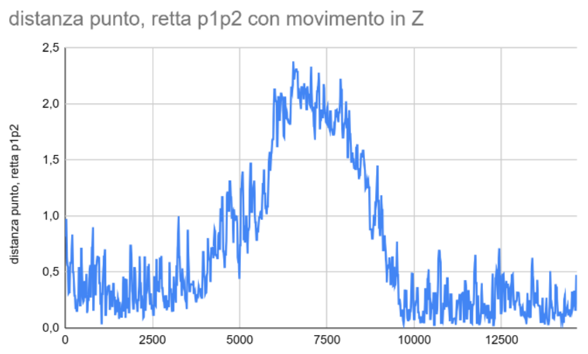
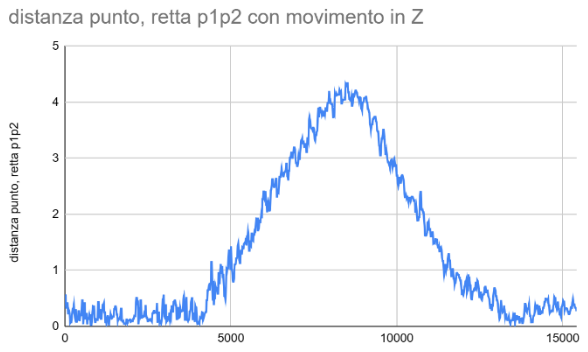
For the dynamic tests, the movements performed by the robotic arm consisted of a repeated sequence of linear movements, performed one for each axis of the Cartesian reference system. In particular, for *configuration 1*, the movement along the X-axis and Y-axis was 1000mm, while for the Z-axis it was 500mm, in order to avoid going outside the working range of the base stations. In *configurations 3* and *configuration 4*, however, the movements along the three Cartesian axes were all 500mm long. With the aim of recording movement as precisely as possible, it was decided to record trajectories with at least two cameras switched on at all times. In accordance with this, one recording was made with two cameras, using the left and the right camera, and two recordings with all three base stations for each movement performed in configuration 1. In configurations 3 and 4, however, each movement was recorded six times.

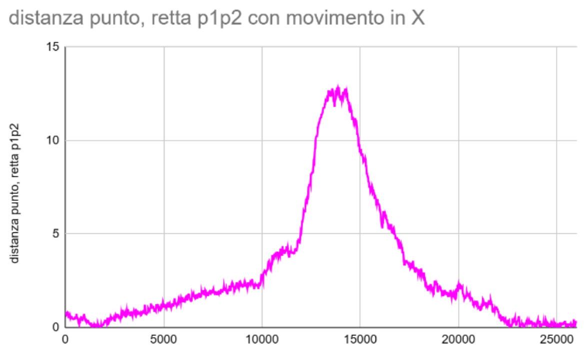
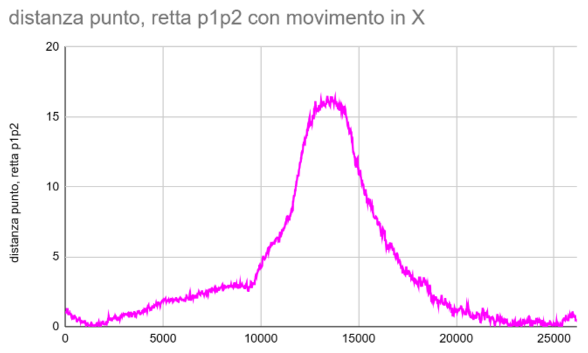
Graphs showing the distance of each recorded position to the line calculated from the start point to the end point and the distance travelled as measured by the tracker are shown in the Table 7. To determine the best and worst case, the standard deviation of the mean of the distance between each individual point and the line travelled was calculated.

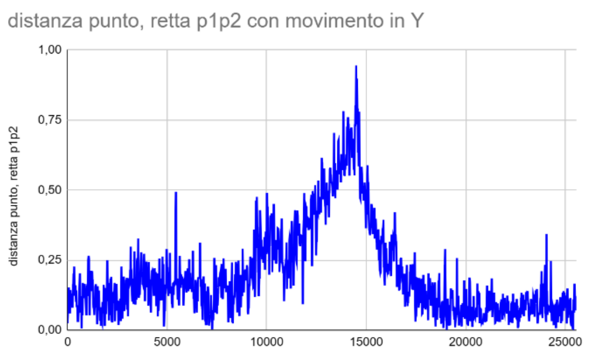
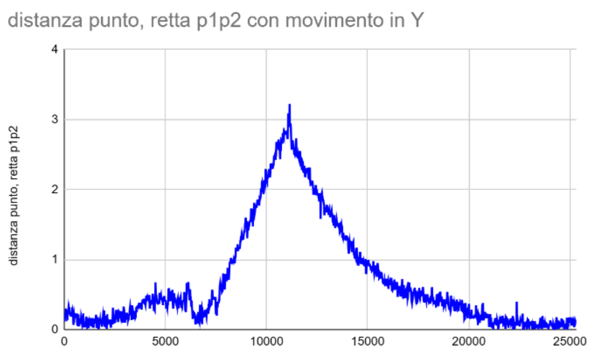
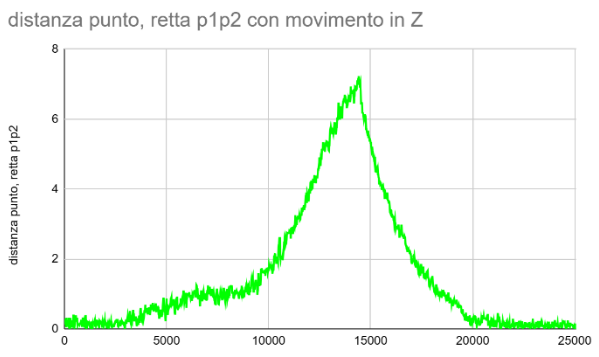
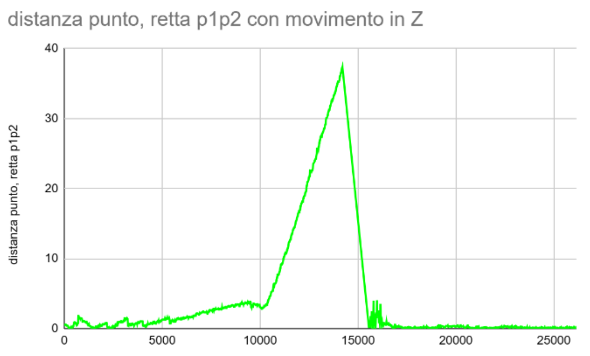
Table 7: point-line and measured distance for Tundra Tracker.

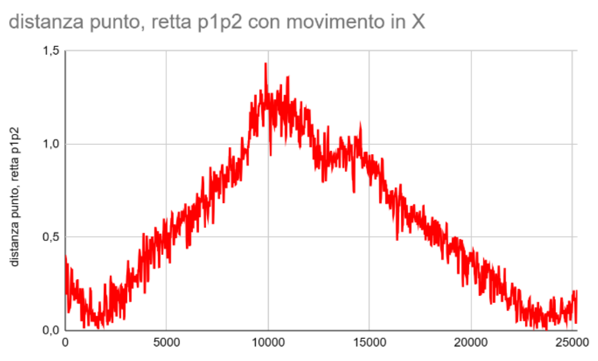
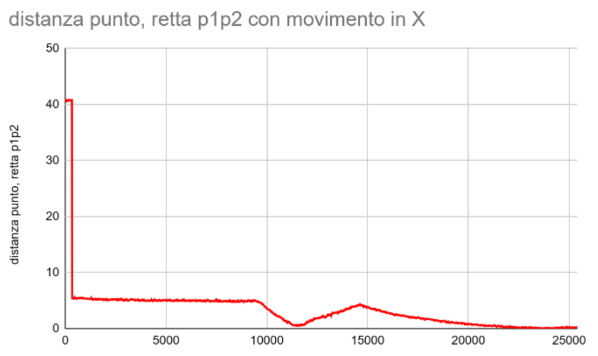
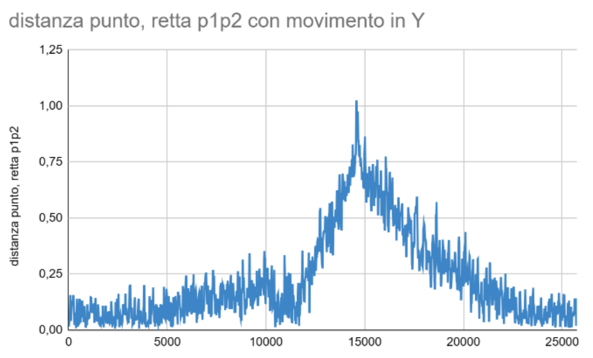
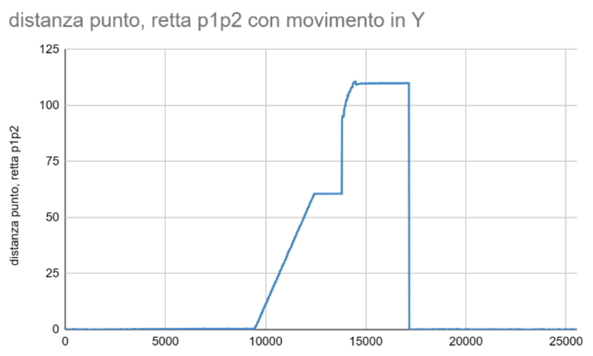
CONFIGURATION	DISTANCE POINTS-LINE [mm]	MEASURED DISTANCE [mm]

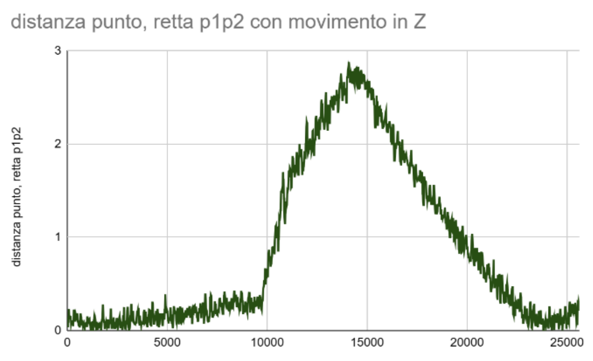
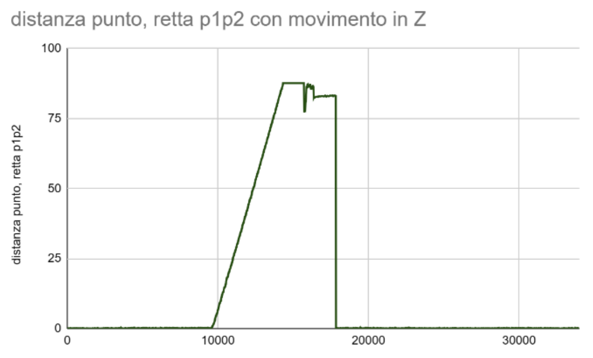
X MOTION	Configuration 1 – registration with left and right cameras.		1009,525669
	Configuration 1 – registration with all cameras – worst case.		1009,204628
Y MOTION	Configuration 1 – registration with left and right cameras.		1004,600502
	Configuration 1 – registration with all cameras – worst case.		999,3449921

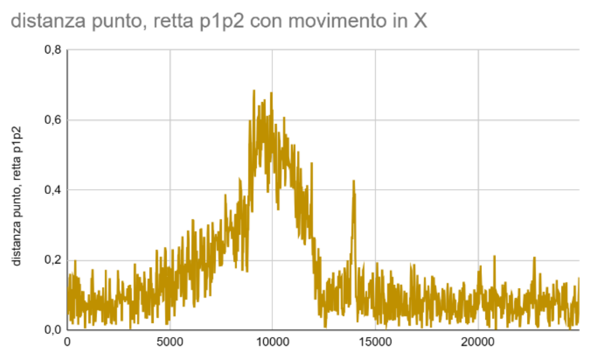
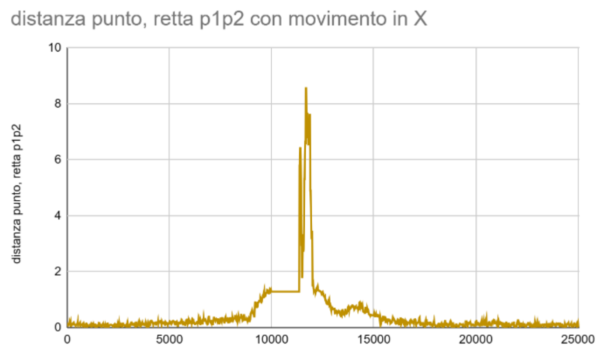
Z MOTION	Configuration 1 – registration with left and right cameras.		504,213404
	Configuration 1 – registration with all cameras – worst case.		504,6515467

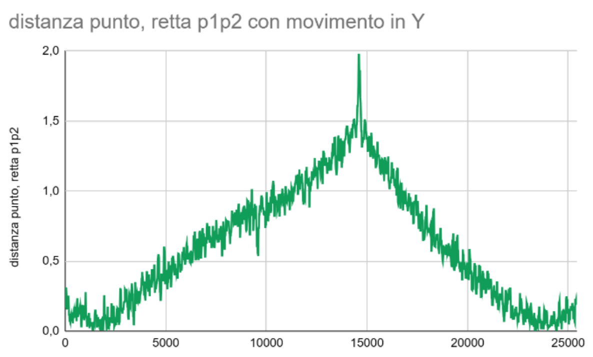
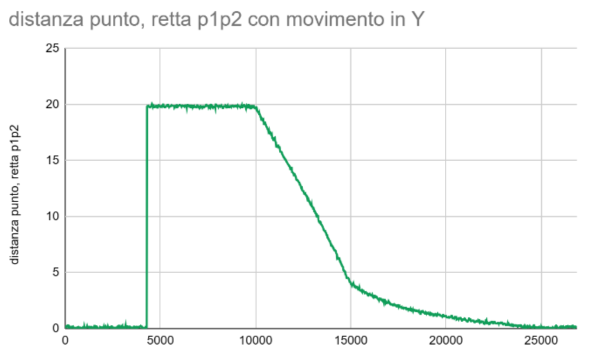
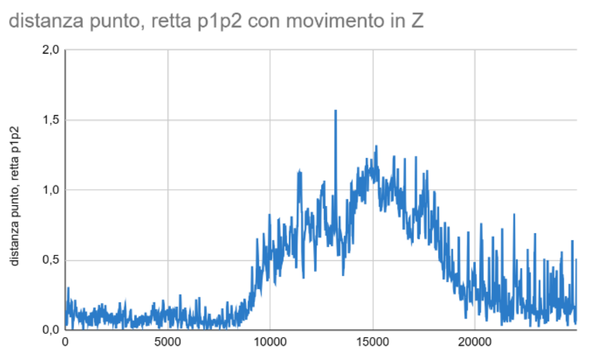
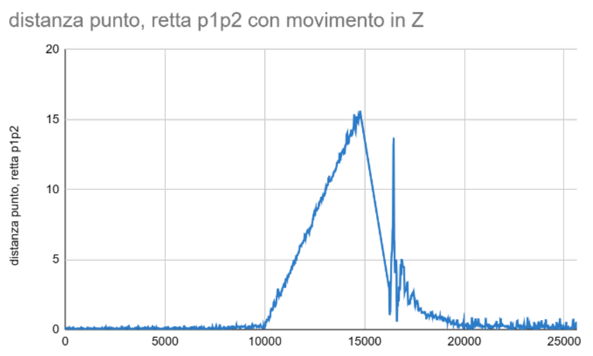
X motion	Configuration 3 – best case.		506,7414017
	Configuration 3 – worst case.		506,8251189

Y motion	Configuration 3 – best case.		510,6966075
	Configuration 3 – worst case.		507,710826
Z motion	Configuration 3 – best case.		505,9614791
	Configuration 3 – worst case.		521,3983842

X motion	Configuration 4 (big room) – best case.		501,6321646
	Configuration 4 (big room) – worst case.		503,3852616
Y motion	Configuration 4 (big room) – best case.		501,0487939
	Configuration 4 (big room) – worst case.		618,2568119

Z motion	Configuration 4 (big room) – best case.		496,4305413
	Configuration 4 (big room) – worst case.		538,5562775

X motion	Configuration 4 (small room) – best case.		501,7408512
	Configuration 4 (small room) – worst case.		501,601504

Y motion	Configuration 4 (small room) – best case.		501,953283
	Configuration 4 (small room) – worst case.		530,8506427
Z motion	Configuration 4 (small room) – best case.		500,1816479
	Configuration 4 (small room) – worst case.		501,2992505

During the execution of the dynamic tests, it was noticed, as already happened for HTC Vive Tracker 3.0 during the noise tests, some jumps and movements that did not agree with the movement performed by the robotic arm. As can be seen from some graphs, there are sudden peaks, which place the sensor very far from the straight line travelled in the virtual world of

MARVIN, leading to the execution of a non-linear movement and an incorrect measurement of the overall distance travelled. The jumps made by the sensor are not connected to the axis of the movement performed, as they are present in all three Cartesian axes. Furthermore, they are not even connected to the different configuration of the Steam VR room, as the jumps are present both in the room configured as small and in the one configured as large. When these jumps are not present, the movement performed on the X-axis is the one that achieved better performance, managing to contain the line travelled inside a cylinder with a maximum radius of 0,6862903364 mm.

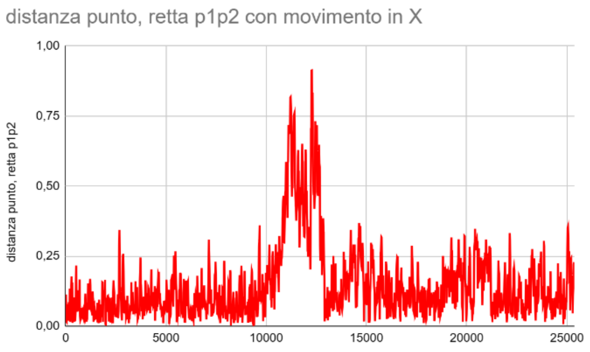
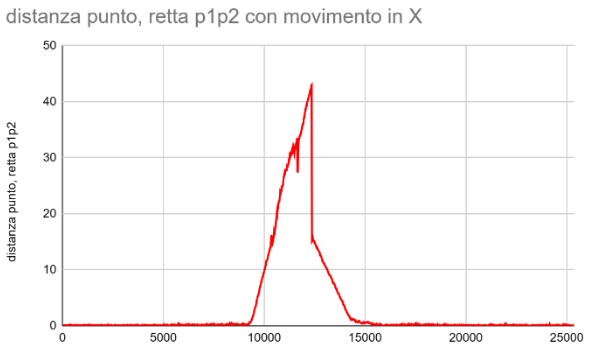
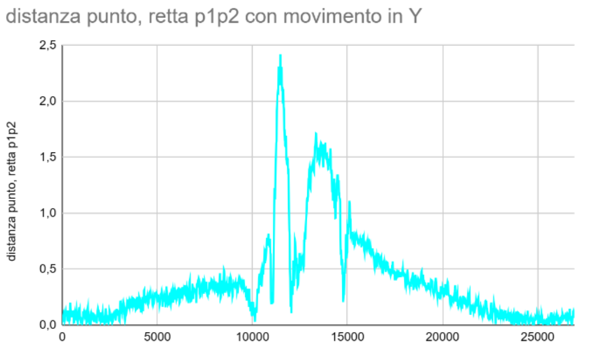
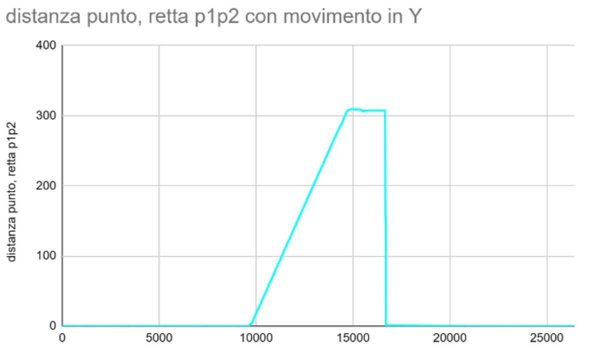
High jumps, i.e. where you can see from the graph, a notable increase to the point of observing all the other distances measured as a single line, also significantly affect the distance measured. Generally, the Tundra Tracker was able to correctly measure the distance travelled to within a few millimetres.

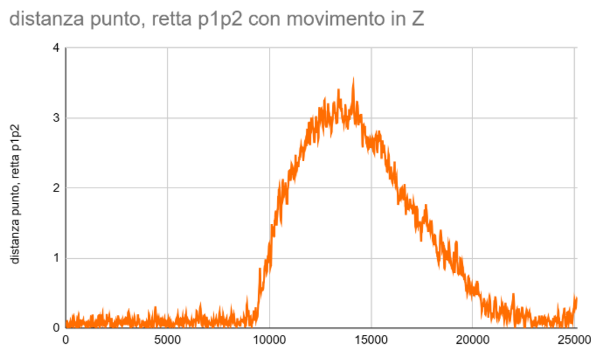
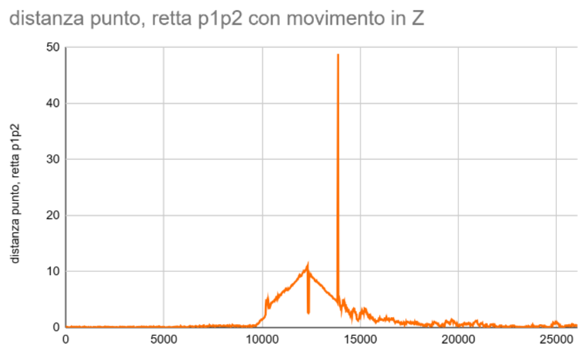
Double Tundra Tracker

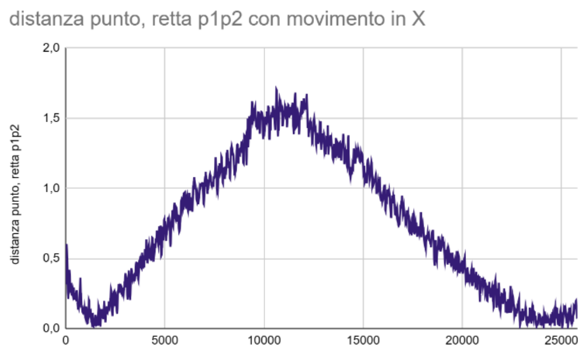
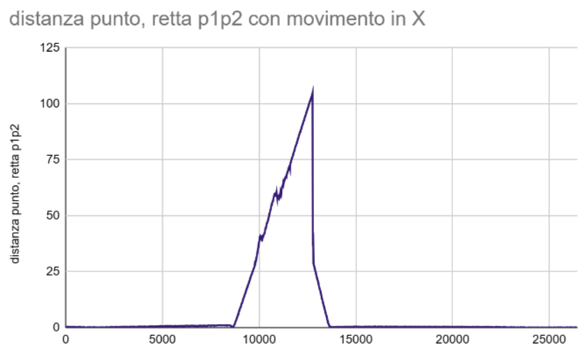
During tests relating to the Tundra Tracker sensor, jumps and peaks were noted which led the sensor to provide incorrect information, both relating to the distance travelled and the distance of each individual position with respect to the trajectory performed. To try to limit this problem, the dynamic tests performed for the Tundra Tracker were repeated using two Tundra Trackers simultaneously and averaging the two positions received, to limit any possible incorrect reading by one of the two sensors. The two sensors were placed on a bar attached to the flange, at a distance of 15cm from each other. Since the Tundra Trackers are placed far apart from each other, the recorded positions of the sensors do not coincide. It was, therefore, necessary to manipulate the data in order to obtain a single relative position for both sensors, simulating the measurement of the same point by both. The two sensors were placed at a distance of 15cm to reduce the possible number of photodiodes covered by the presence of the other sensor.

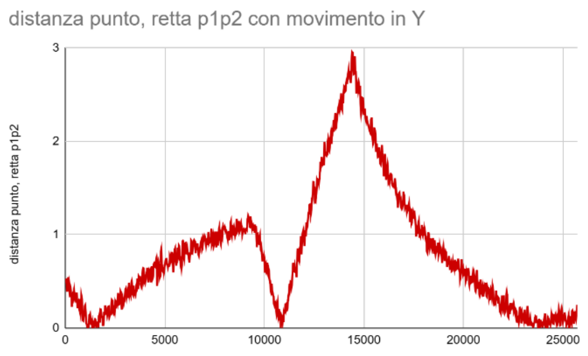
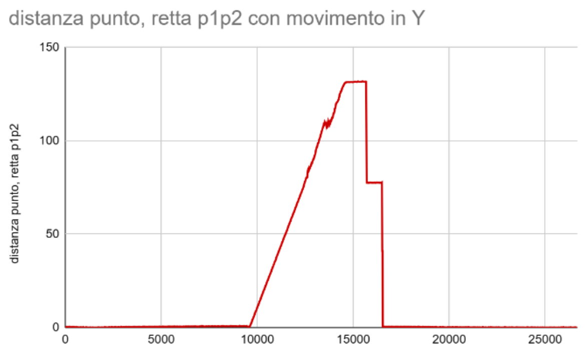
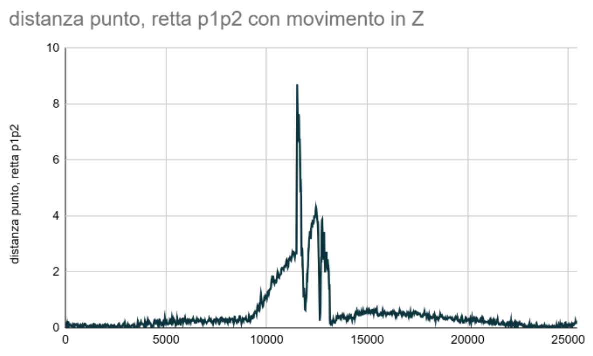
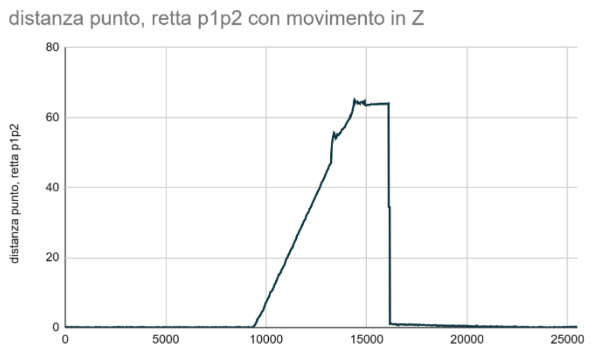
The tests were carried out in *configuration 4*, recording 6 movements for each axis, covering a distance of 500mm. The Table 8 shows the graphs of the distance of each point with respect to the line travelled and the distance measured, relative to the average of the data received from both sensors.

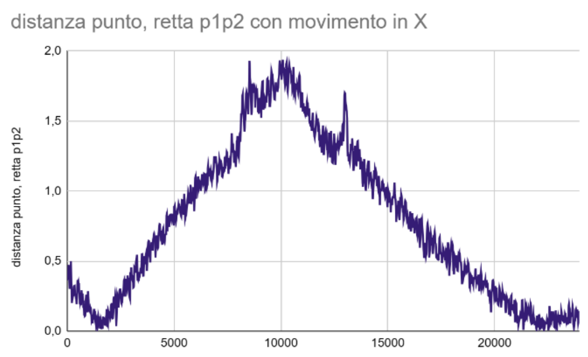
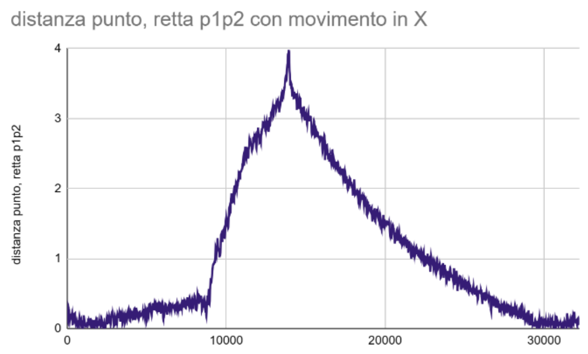
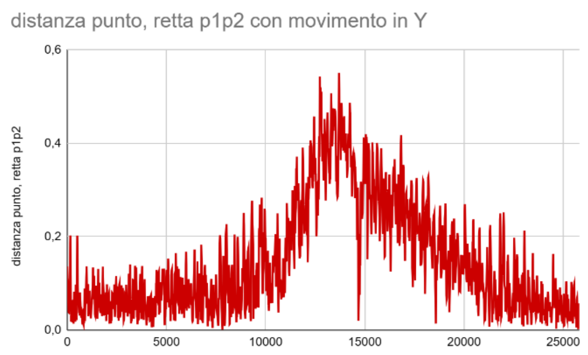
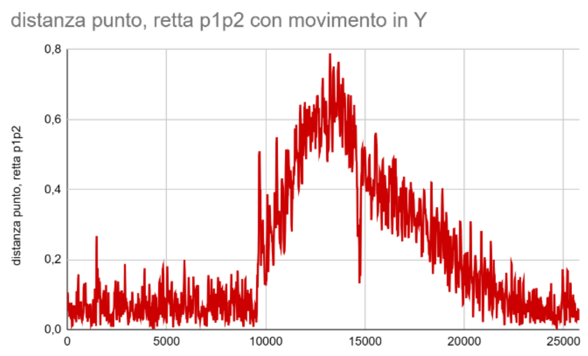
Table 8: point-line and measured distance for double Tundra Tracker.

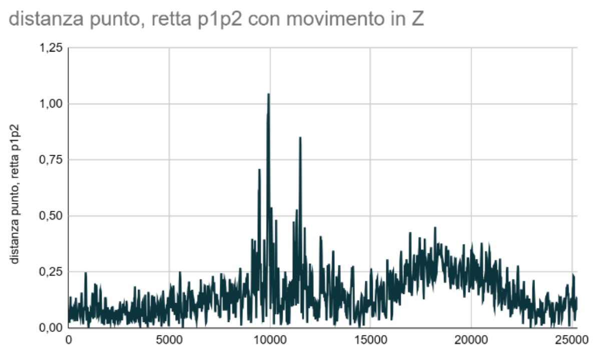
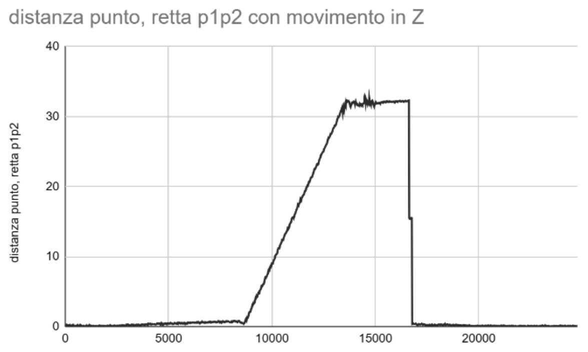
CONFIGURATION		DISTANCE POINTS-LINE [mm]	MEASURED DISTANCE [mm]
X motion	Configuration 3 – best case.		504,0582598
	Configuration 3 – worst case.		558,7241646
Y motion	Configuration 3 – best case.		497,1101181
	Configuration 3 – worst case.		431,9820084

Z motion	Configuration 3 – best case.		503,4948609
	Configuration 3 – worst case.		494,7577204

X motion	Configuration 4 (big room) – best case.		502,5242297
	Configuration 4 (big room) – worst case.		432,4266245

Y motion	Configuration 4 (big room) – best case.		500,3860924
	Configuration 4 (big room) – worst case.		376,8588932
Z motion	Configuration 4 (big room) – best case.		498,2322746
	Configuration 4 (big room) – worst case.		513,7498514

X motion	Configuration 4 (small room) – best case.		501,3652286
	Configuration 4 (small room) – worst case.		500,5601538
Y motion	Configuration 4 (small room) – best case.		501,7838017
	Configuration 4 (small room) – worst case.		502,3532542

Z motion	Configuration 4 (small room) – best case.		501,6296631
	Configuration 4 (small room) – worst case.		491,9248033

As can be seen from the graphs of the distance of each position from the straight line travelled, the presence of two sensors to provide the final position did not serve to improve the final performance of the system. In fact, although during the tests only one sensor underwent movements in the virtual world that did not conform to the movements performed in the real world, the one not affected by these jumps is unable to compensate for the incorrect reading of the position and provide, through the average, a final position consistent with what is performed in the real world.

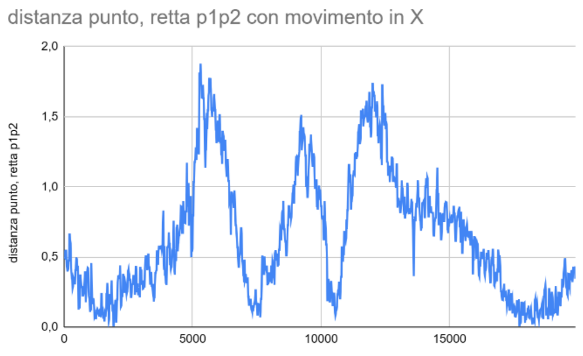
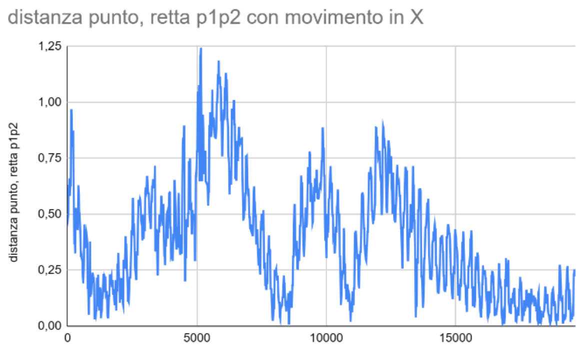
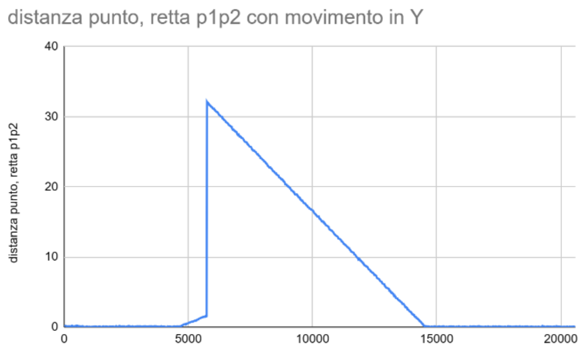
It is also worth underlining that both sensors are affected by these sudden jumps in the space of the virtual world, eliminating, as a trigger, the possible breakage of the two sensors.

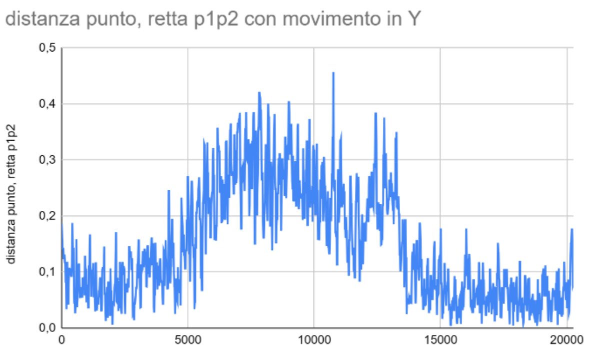
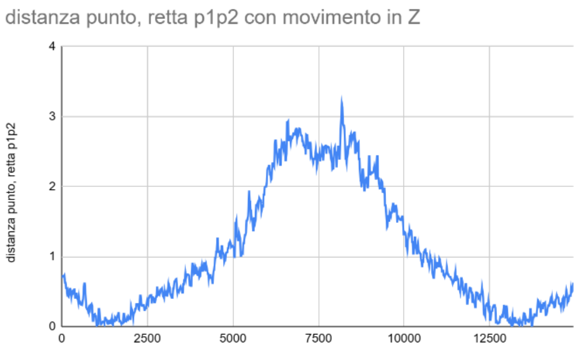
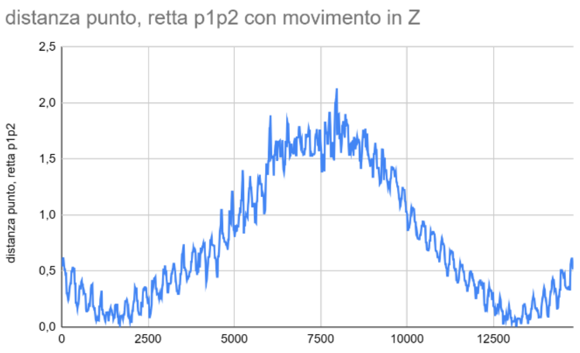
Valve Index Controller

For Valve Index Controller, the dynamic tests were always performed in configuration 1, once for each axis of the Cartesian system using two base stations, in particular the cameras positioned on the right and left were used, while two recordings were performed for each axis with all three cameras present. The distance travelled during each movement along the X and Y axes was 1000 mm, while along the Z axis it was 500 mm.

The Table 9 shows the graphs representing the distance of each position detected with respect to the line travelled and the distance measured from the starting point to the final point.

Table 9: point-line and measured distance for Valve Index Controller.

CONFIGURATION		DISTANCE POINTS-LINE [mm]	MEASURED DISTANCE [mm]
X motion	Configuration 1 – registration with left and right cameras.		995,752264
	Configuration 1 – registration with all cameras – worst case.		996,1899583
Y motion	Configuration 1 – registration with left and right cameras.		1010,856248

	Configuration 1 – registration with all cameras – worst case.		997,8108896
Z motion	Configuration 1 – registration with left and right cameras.		499,0184554
	Configuration 1 – registration with all cameras – worst case.		498,911419

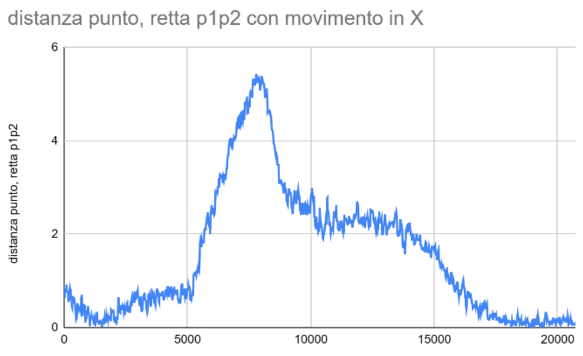
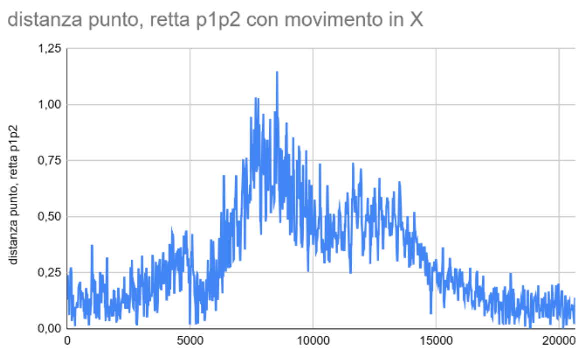
The Valve Index Controller provided the best performance with three cameras working, managing to contain the recorded linear motion within a cylinder of radius 0.457015668mm. Also regarding the ability to measure the distance travelled, the sensor has shown itself to be able to measure the distance correctly, with a deviation of a few millimetres.

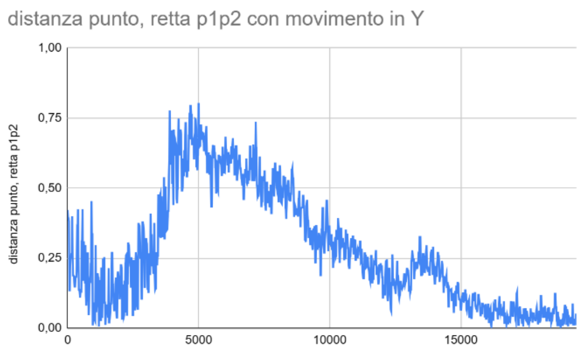
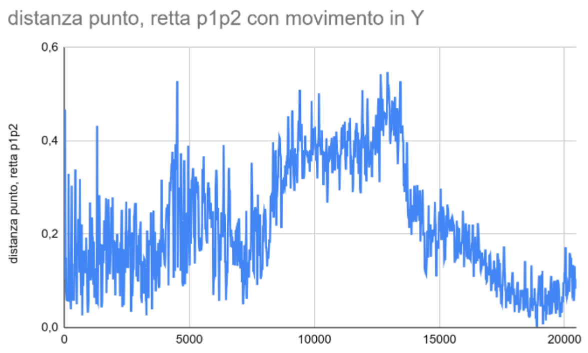
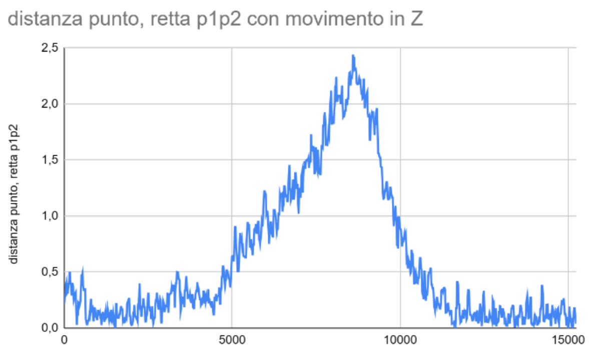
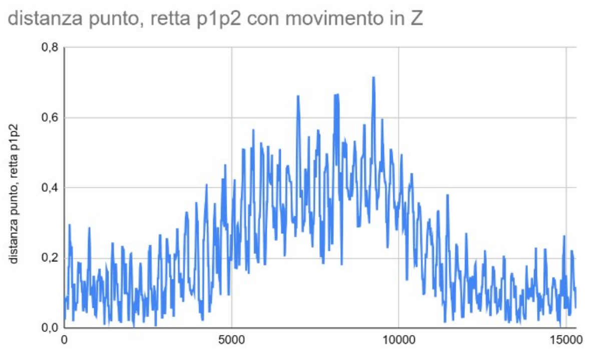
HTC VIVE Tracker 3.0

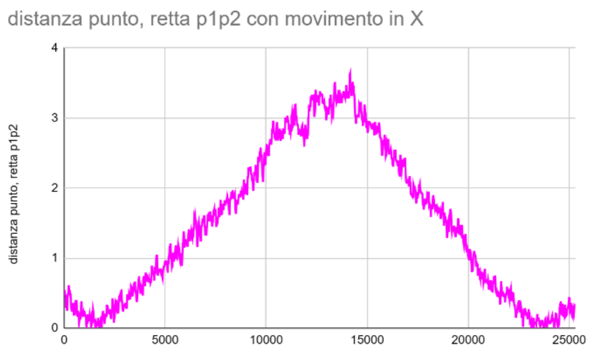
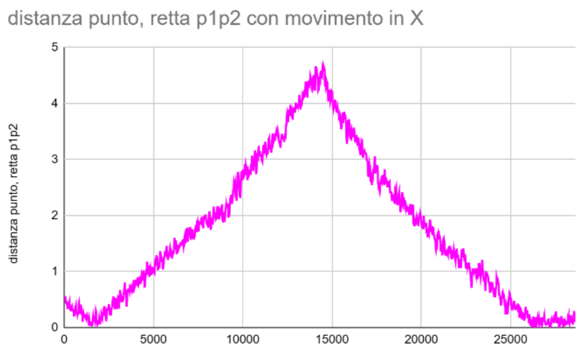
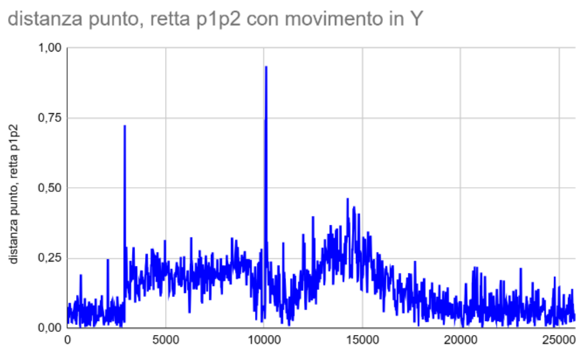
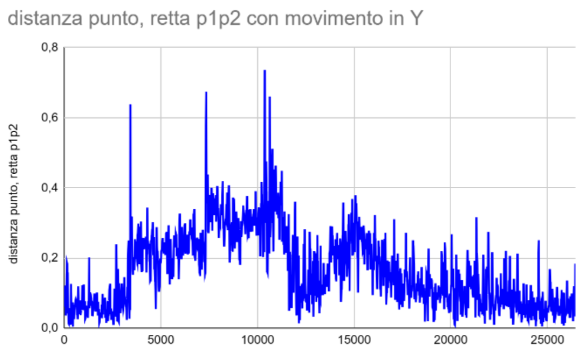
The dynamics of the HTC VIVE Tracker 3.0 were studied in different configurations. In configuration 1, data was collected once for each axis with two cameras, specifically the left and right base stations were turned on, and twice with three cameras. The distance travelled along the X and Y axes was 1000 mm, while along the Z axis it was 500 mm. In configurations 3 and 4, six movements of 500mm each were performed and recorded.

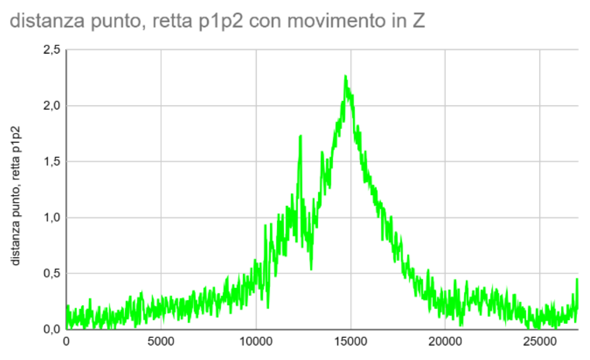
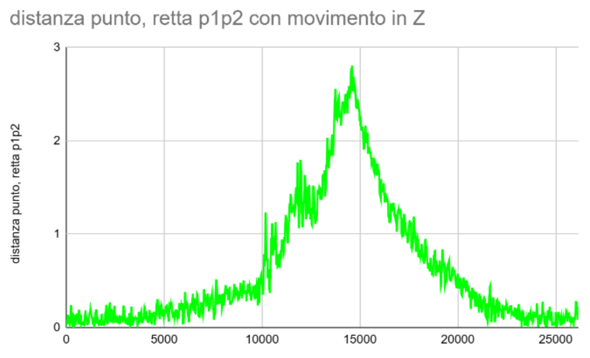
In the Table 10 you can see graphs showing the distance of each position sent by the sensor from the line formed from the start point to the end point. The distance measured by the sensor during movement is also shown. The worst case was chosen by observing the movement that produced the greatest standard deviation compared to the average distance of each single point from the line.

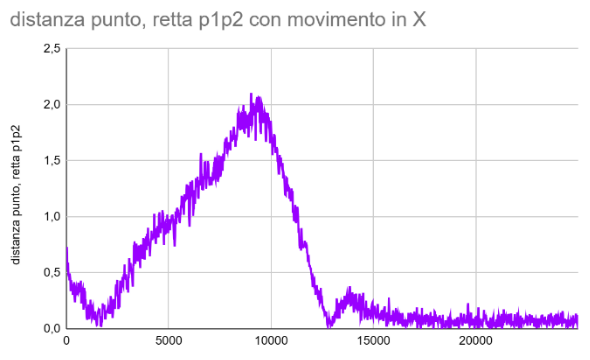
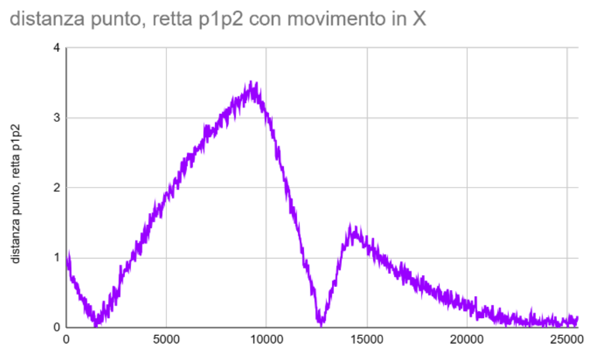
Table 10: point-line and measured distance for HTC Vive Tracker 3.0.

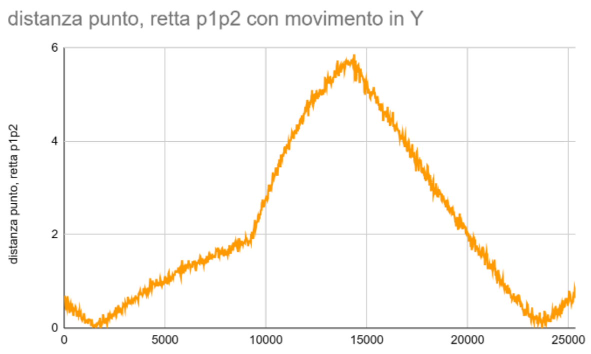
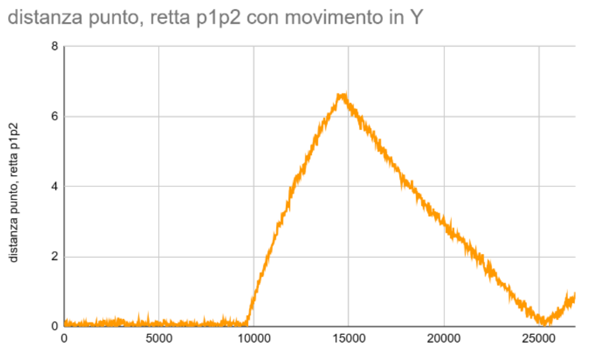
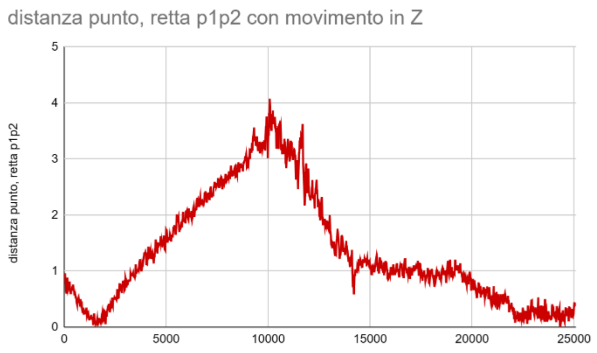
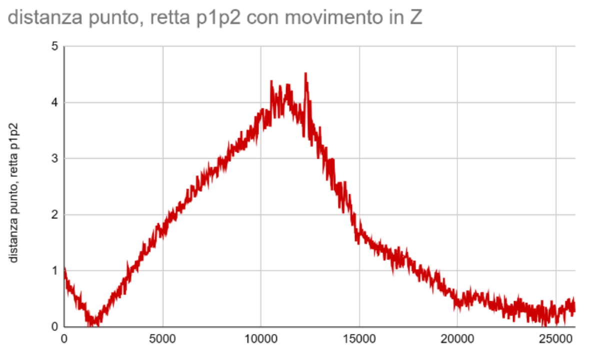
CONFIGURATION		DISTANCE POINTS-LINE [mm]	MEASURED DISTANCE [mm]
X motion	Configuration 1 – registration with left and right cameras.		1000,649107
	Configuration 1 – registration with all cameras – worst case.		1003,18467

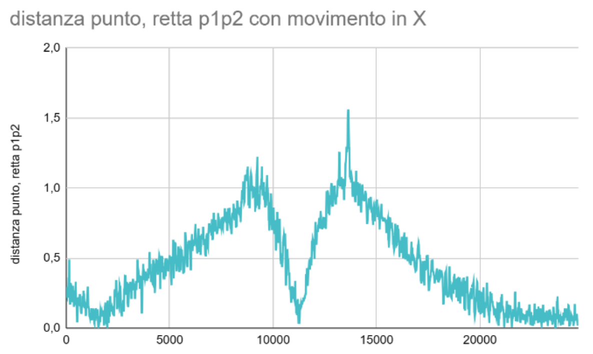
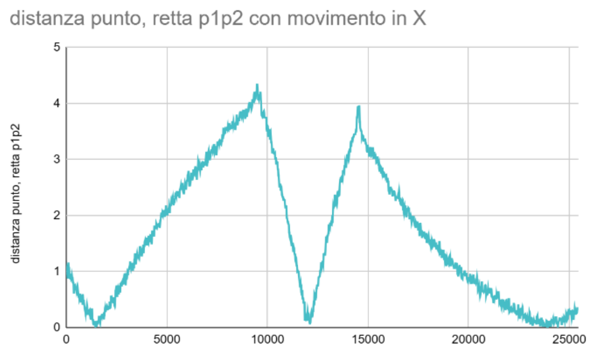
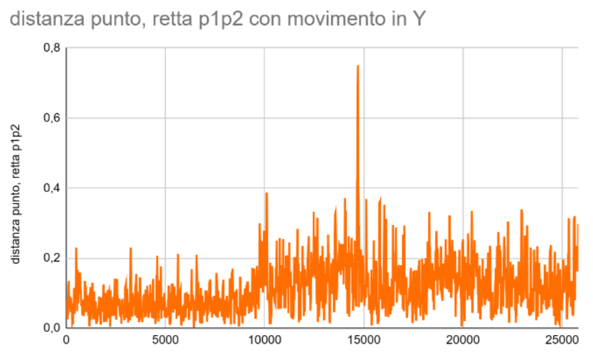
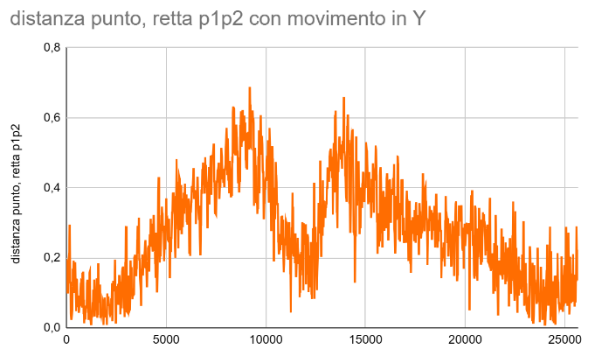
Y motion	Configuration 1 – registration with left and right cameras.		1001,160374
	Configuration 1 – registration with all cameras – worst case.		1003,069011
Z motion	Configuration 1 – registration with left and right cameras.		501,2255416
	Configuration 1 – registration with all cameras – worst case.		501,1005786

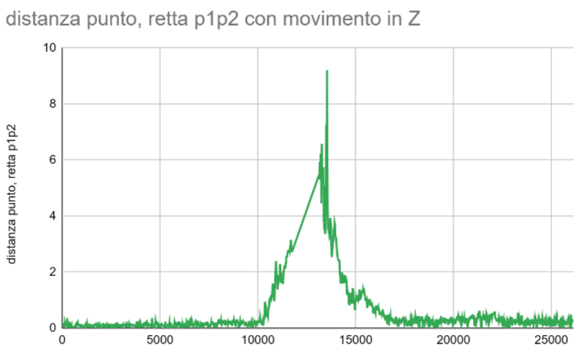
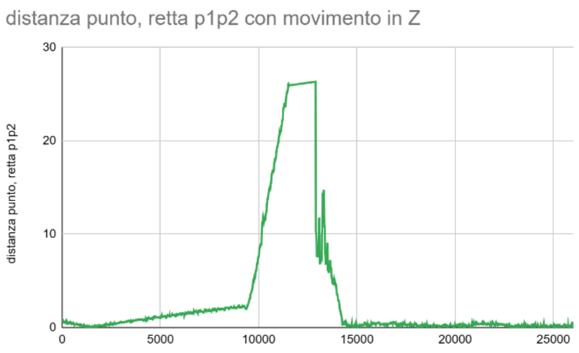
X motion	Configuration 3 – best case.		500,660658
	Configuration 3 – worst case.		500,7138501
Y motion	Configuration 3 – best case.		498,5079719
	Configuration 3 – worst case.		498,3249213

Z motion	Configuration 3 – best case.		500,8744656
	Configuration 3 – worst case.		500,9026684

X motion	Configuration 4 (big room) – best case.		498,5087948
	Configuration 4 (big room) – worst case.		499,3826092

Y motion	Configuration 4 (big room) – best case.		495,529309
	Configuration 4 (big room) – worst case.		491,8911094
Z motion	Configuration 4 (big room) – best case.		498,8271946
	Configuration 4 (big room) – worst case.		498,7279385

X motion	Configuration 4 (small room) – best case.		502,1255223
	Configuration 4 (small room) – worst case.		499,3868284
Y motion	Configuration 4 (small room) – best case.		501,0584043
	Configuration 4 (small room) – worst case.		502,2481032

Z motion	Configuration 4 (small room) – best case.		513,7118094
	Configuration 4 (small room) – worst case.		533,8013837

HTC Vive Tracker 3.0 generated better results when running the dynamic tests than it performed with the noise tests. The sensor, in fact, was more stable and less subject to sudden jumps. Using three base stations, the tracker was able to keep the cylindrical moving along a linear trajectory, always with a radius of less than 1.5mm in the worst case. In configurations with two cameras, however, the cylinder expanded, extending the beam to 3mm or more.

Results

During the execution of required measurements, it was discovered a problem within the system, since it was seen a random jitter, also when the sensors were fixed. This jitter caused problems during the measurements since it did not provide a coherent position for all the time measurement. The cause that generated this event was not precisely discovered, because the possible causes provided by the developer were one by one checked and, eventually, corrected. Another problem that was encountered was the drift present also when the system had been on for a long time. It was known that the system required some time to calibrate and provided the exact position of the sensors, but in some cases, it drifted without following any predictable path. For the drift experiments, it was discovered that 1 camera was not enough for receiving data with an acceptable error, considering the fact that the precisions requested was about 1mm, so the system needed at least 2 cameras.

The Tundra Tracker worked very well with 2 cameras in a dark place and with 3 cameras in a lighter place, but it did not provide any guarantees about its behaviour when the system was just turned on, so it needed that the system will be turned on before starting the usage of the sensor.

The Valve Index Controller worked very well with 3 cameras, but it provided satisfying results also with 2 cameras. When the system was just turned on it fit with the expected behaviour, namely a first continuous oscillation, followed by a stabilization. It stabilized after approximately 30 minutes.

The HTC Vive Tracker 3.0 was the sensor that provided the best results in terms of time; indeed, it stabilized after 20 minutes, but it provided some strange behaviour. The exact cause of these behaviours was not very clear, since the sensor worked in the same conditions as all the others, but it was supposed to be a change in the light of the room, that was not perceptible by humas. In support of this hypothesis, there is the fact that the HTC Vive Tracker 3.0 provided the best results in the dark room, staying within the upper limits of the error, also when the system was just turned on.

The difference in time required for each sensor to stabilise when the system is just switched on is displayed in the Figure 8:

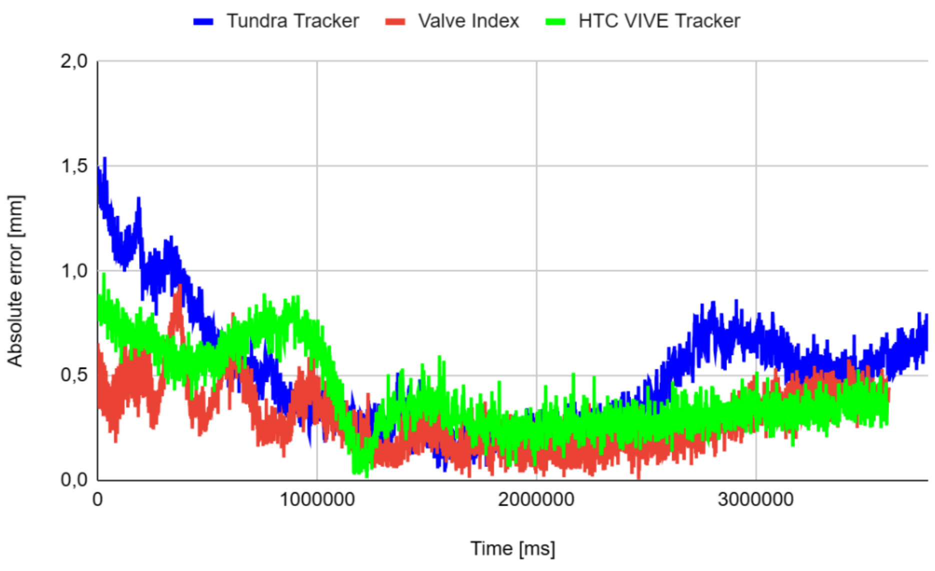


Figure 8: drift results for the three sensors.

In the noise experiment both Valve Index Controller and Tundra Tracker worked very well with two or more cameras, provided the expected output. The HTC Vive Tracker 3.0, on the other hand, behaved unexpectedly, indeed sometimes it behaved as predicted, provided optimal results with two or three cameras, while at other times, the position received by the sensor during recording showed a movement of the sensor, although the sensor had always been stationary in place.

For the dynamic tests, all the sensor worked as predicted, since most of the time, the output graphs had a bell shape.

The Tundra Tracker provided similar results with two or three cameras, both in terms of point distance from the line and measured covered distance. The distance of each received position to the line travelled was most often too great, in fact the sensor moved further away from the line than the required 1 mm, meanwhile the measured distance was always very different from the distance covered by the robotic arm. With the Tundra Tracker, it was decided to use two sensors and mediate the received position to increase the precision of both sensors. In this case, the reliability of the two sensors was very important, but in always all performed motions, at least one sensor did not work correctly, so the average output value was not usable. Nevertheless, the average of the two sensors provided a better result in terms of measuring the travelled distance, in fact the two sensors were able to measure accurately the covered line.

In Figure 9, the graph shows the difference between the distance measured by the single Tundra sensor and the pair of Tundra sensors in configurations 3 and 4. The measurements reported are the same as those reported in Table 7 and Table 8. In these configurations, the sensors travelled a distance of 500mm for each Cartesian axis. The average difference measured for the single Tundra Tracker is 14mm, which translates to the sensor adding, on average, 14mm to the recorded measurement. The Tundra Tracker pair, however, has an average difference of -10mm, resulting in measurements approximately 10mm lower than the 500mm performed.

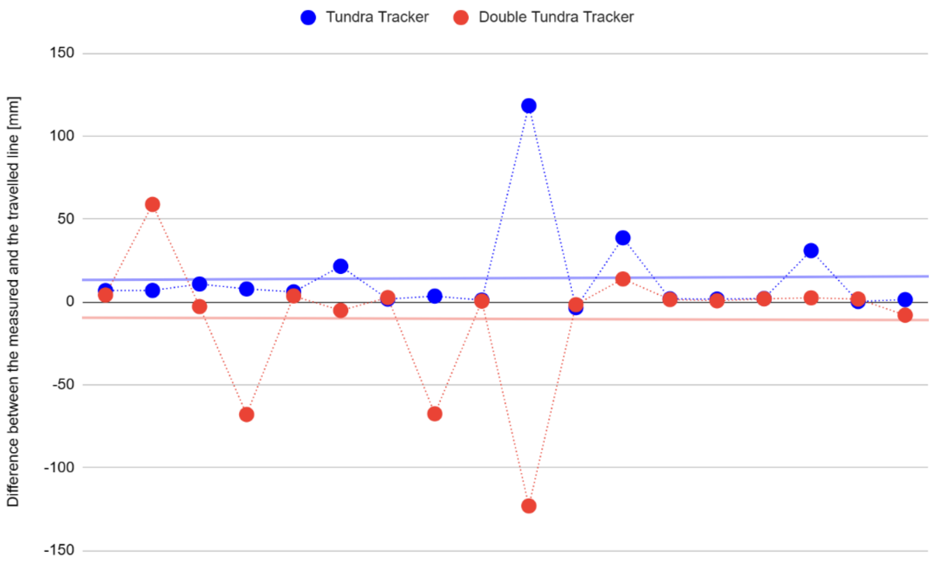


Figure 9: difference of measured distance by Tundra Tracker and double Tundra Tracker.

The Valve Index Controller provided good results in both studied behaviours, since it was able to measure the covered distance with accuracy and have small spikes during the motion, reducing the bell shape of the graph.

The HTC Vive Tracker 3.0 was good in measuring the travelled distance and gave good results in following the line during the motion. It was more consistent in the final outputs, but it provided a bigger cylinder with respect to the Valve Index.

In Figure 10, Figure 11, and Figure 12 it is possible to see the differences between the 3 sensors in terms of point-line distance from the line travelled by the robotic arm, for each frame axis.

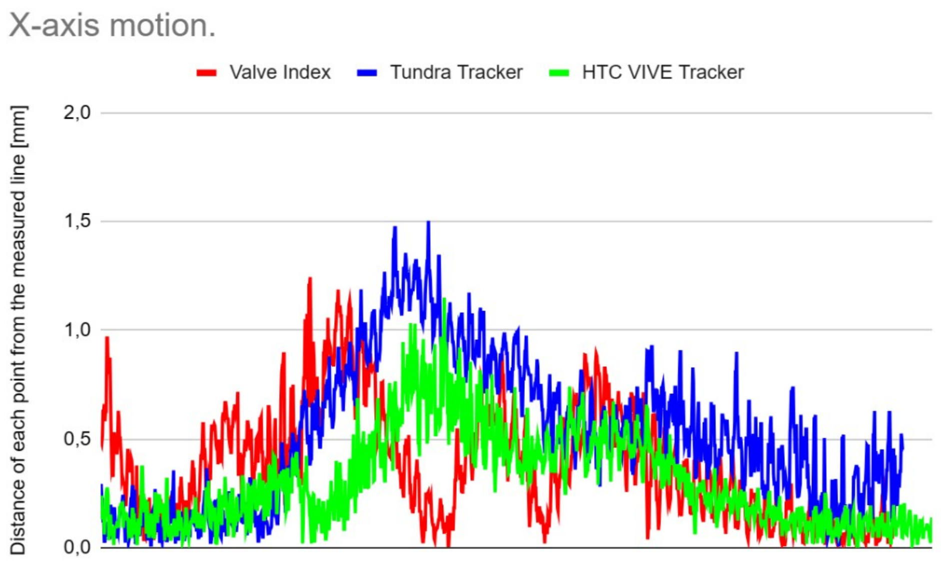


Figure 10: point-line distance during X-axis motion.

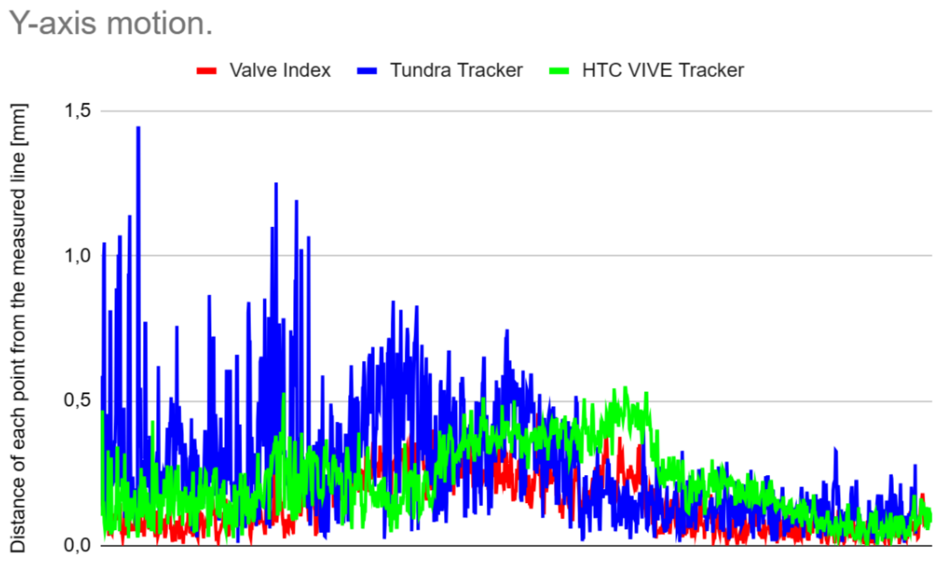


Figure 11: point-line distance during Y-axis motion.

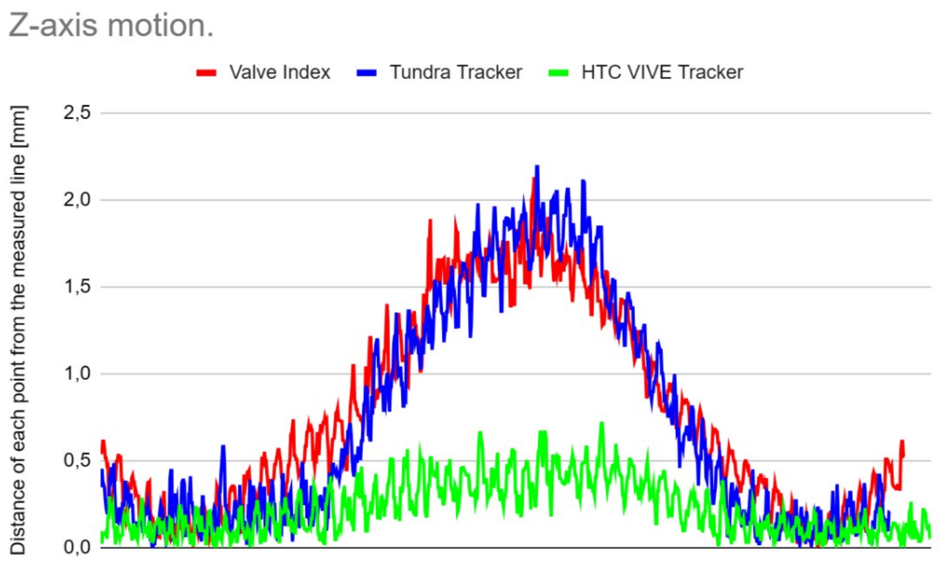


Figure 12: point-line distance during Z-axis motion.

In conclusion, according to these experiments, it is recommended to use two cameras to cover more area and have the possibility to be able to see the sensor also when one camera is obstructed. With two cameras, the position received from the sensor is more precise. For these sensors, the presence of a third camera is not necessary, unless the motions done will go to occlude both cameras.

After the stabilization of the system, for dynamic purpose, the best sensors are the HTC Vive Tracker 3.0 and the Valve Index Controller, since they provided the best results during both measuring processes.

COMPOSITION OF THE 2D PANORAMA

After the studies relating to the choice of the sensor to be used for the final part of the project, the next part concerned the creation of a final overall image, using a time-of-flight camera for data acquisition.

The time-of-flight camera was mounted on a robotic arm, which, initially, provided the position of the camera relative to the robot's world reference system, meanwhile the time-of-flight camera provided data about the scene in its field of view. Meanwhile positions were used to divide each image into different list having similar position to each other, the IFM camera provided two types of data; indeed, it created the point cloud of the scene, but also the relative 16 bits grayscale image, which is used for the image stitching. Each received information were stored in the computer to allow subsequent usage.

The ToF camera

The Time-of-Flight camera [16] [17] is a sensor that measures the distance between the camera itself and surrounding objects, calculating the time it takes for light, or other signals, to return to the camera after hitting the surface of objects.

A Time-of-Flight camera is composed by different pieces, as shown in Figure 13:

- *Sensor module*, which collects the reflected light from the scene and convert the data into a depth map, where each pixel represents the distance between the camera itself and the object in the scene.
- *Light source*, which is the module that provides the signal source that travels through space.
- *Depth processor*, which helps to convert the raw data into depth information. This processor can also provide a 2D infra-red image.

Time-of-flight camera operation is based on the time, represented by the phase of the light, it takes for the signal to complete the camera-object-camera path. The phase of the light is then converted into a distance and stored into a depth map.

Usually, the used light is emitted by a LED or a solid-state laser that works in the near infrared range, so from 780 nm to 2500 nm, light that is invisible to the human eyes. The imaging sensor must work in the same rage of light.

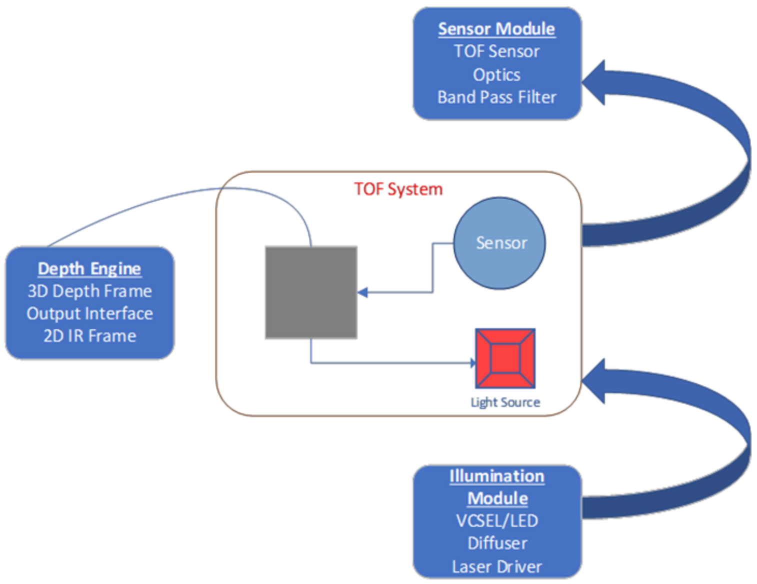


Figure 13: system of a time-of-flight camera.[10]

The time-of-flight camera has some advantages, first of all, the speed, since it works with the fastest thing in the universe, light. It is also very compact, because sensor and light source are place one near the other and there are no moving parts. The energy that the camera needs is not elevated, since the time-of-flight camera does not require a lot of computational power.

On the other hand, this type of sensor is sensitive to the background light, which can interfere with the light emitted by the ToF camera. Another problem is given if more ToF are used at the same time, since the cameras can disturb each other. Similarly, when the light is reflected by the object there is no guarantee that it will be a single path, but it can be reflected in several different path, which results in a wrong reading by the ToF camera.

IFM

In this case of study, the used time-of-flight camera was an IFM camera [18], in particular the 3D camera O3D301. The camera is shown in Figure 14:

[10] Image source: https://www.e-consystems.com/blog/camera/technology/what-is-a-time-of-flight-sensor-what-are-the-key-components-of-a-time-of-flight-camera/

Figure 14: IFM ToF O3D301. Image taken from IFM website.

Meanwhile, its technical scheme is shown in the Figure 15:

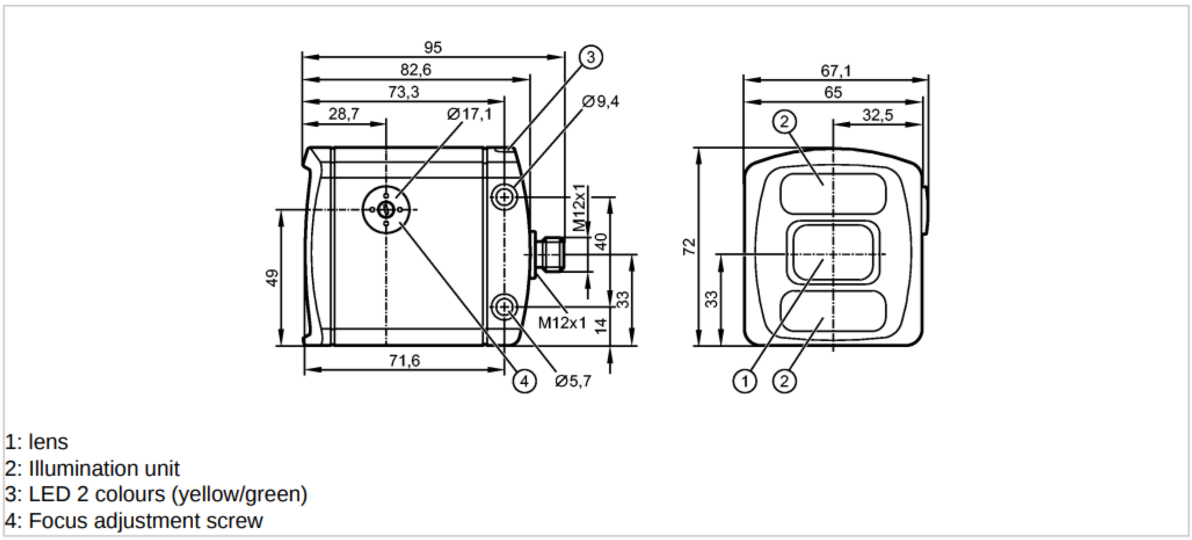


Figure 15: technical scheme of IFM O3D301. Image taken from IFM O3D301 manual.

The technical characteristics of this 3D camera are:

- Image resolution: 352 x 264 pixels.
- Angle of aperture 3D: 40° x 30° (horizontal x vertical), nominal value without lens distortion correction.
- Operating distance: from 300 mm to 10000 mm.

The data output format can be:

Data type	**Data value**	**Note**
Distance [mm]	0 to 65535 (16 bits unsigned integer)	Radial distance
Cartesian coordinates x, y, z [mm]	-32767 to 32767 (16 bits signed integer)	x, y: lateral position z: vertical distance
Amplitude [a.u.]	0 to 65535 (16 bits unsigned integer)	Object brightness

Some parameters can be set via PC with IFM Vision Assistant, which is the proprietary software used for controlling, checking, saving, or cloning the IFM sensors, or XML-RPC. For communicating with the PC or Internet, the IFM camera needs an Ethernet cable.

IMAGE PROCESSING

Every time the IFM camera was activated, the application saved the data received and added the spatial position of the camera. The position is composed of the three cartesian coordinates and the three angles. At first, the position referred to the robot's world reference system, while subsequently the VR tracker was added, in order to free the camera from the robotic arm and thus make it portable.

Once the acquisition of the images is finished, they are divided into lists based on the coordinate chosen by the user. To divide the images, it was chosen to use a tolerance threshold related to the coordinate, as specified by the Equation 5:

$$coordinate_{new} \in (cordinate_{first} \pm TOLLERANCE)$$

Equation 5: formula used for adding an image.

where:

- $coordinate_{new}$ is the coordinate of the image that is currently processed. This image is waiting to be added to its list.
- $cordinate_{first}$ is the coordinate of the first image of the list.
- $TOLLERANCE$ is the tolerance that create the boundaries for deciding if an image can be added to the list or not.

This allowed us to create a set of images where there are elements in common, which will allow OpenCV to make the final stitch. Depending on which coordinate is used for sorting images, different results can be obtained.

The IFM camera provided 16 bits grayscale images. These images provided a better quality, since their color range to represent a pixel is not limited to 255, but it can sweep from 65536 different values. From the images obtained from the IFM, it was noted that this range is not fully utilised, in fact most of the images assign maximum pixel values around 5000. Although 16-bit images provide better detail and information, in order to be able to stitch them, it was necessary to transform them into 8 bits per pixel images, otherwise OpenCV would not be able to work with them, as it only accepts 8-bit greyscale images, or 32 bits in the case of ARGB. During this transformation, of course, some information was lost, but it did not compromise the realisation of the final stitch. An example of input images received from the IFM and then transformed into 8-bits images can be seen in Figure 16.

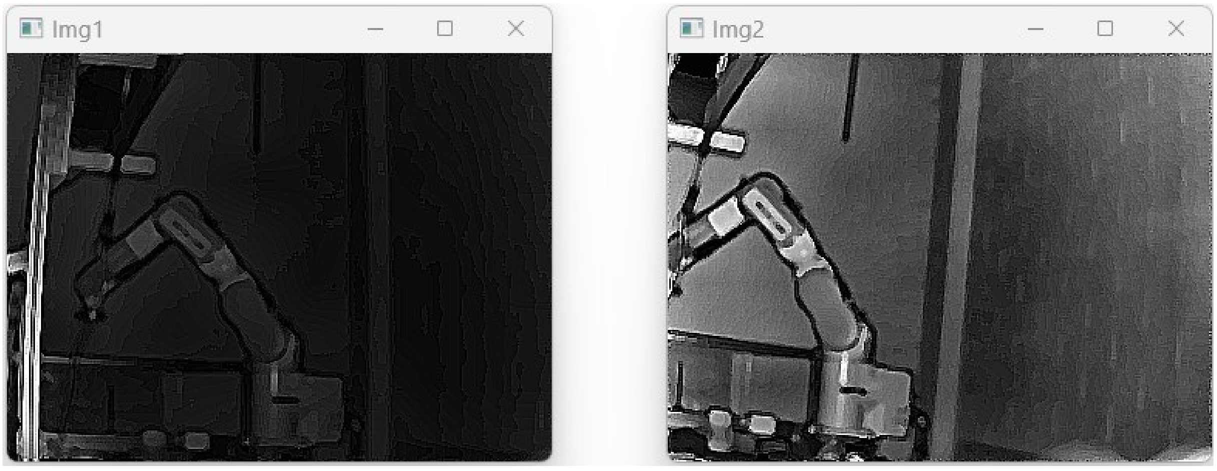


Figure 16: two input images received from IFM camera.

Image stitching

Image stitching is a well-known part of computer vision. It is a process where different images are merged together to create a panoramic picture, but input images must have overlapping fields of views, to allow the merging, and these overlapping parts can be between 15% to 30% of the whole image.

Image stitching is used for different applications, such as document mosaicking, that is the process where multiple images of the same document are stitched together for obtaining a large and high-resolution image of the document, image stabilization, where the stitching is used for compensation the motion of the camera, high-resolution image mosaics in digital maps and satellite imagery, medical imaging, video stitching and more other different applications.

Image stitching is not an easy task since multiple problems and challenges can occur. First of all, the illumination of the scene, which is impossible to have equal in every image. These difference illuminations can lead to create a panoramic image whit different illumination zone. Other problem to solve are lens distortion due to camera's lenses, presence of parallax, exposure difference between images, and scene motion.

All these problems can lead to a failure during the computation of the panorama or can lead to the creation of a panoramic image having an inadequate quality, making it unusable.

OpenCV and EmguCV

OpenCV (Open-Source Computer Vision Library) [19] is used for programs that needs real-time computer vision tasks. It is a cross-platform library and licensed as free and open-source software under Apache License 2. Developed in 1999, by Intel, it is written in C++, but there are language bindings in Python, Java, and MATLAB. To enable a wider audience to use the library, different wrapper libraries have been developed.

OpenCV was created to provide an infrastructure that was common to all to enable the use of applications that required computer vision tasks, such as facial recognition system, human-computer interaction, gesture recognition, mobile robotics, object detection, augmented reality, and segmentation and recognition. It also provides 2D and 3D feature toolkits. Since all of these tasks are quite complex, OpenCV had to include statistical machine learning libraries that execute algorithm as boosting, support-vector machine, random forest, and all the other algorithms used in machine learning. In total, OpenCV has more than 2500 algorithms, all optimized for performing all the basics and the state-of-the-art algorithms of both the computer vision and machine learning tasks.

In this case of study, the language used for developing the application was C#, so it was needed to use a wrapper library, and the chosen one was EmguCV [20]. EmguCV is a .Net wrapper for the OpenCV library and it is cross platform. This allows to call OpenCV from all the .Net compatible languages, in order to be able to use all the capacities of OpenCV, without being forced to use C++. The wrapping of OpenCV is done using 3 classes:

- Emgu.CV.CvInvoke, which wraps the functions of OpenCV. This class provides a method to invoke functions from OpenCV, using directly a .Net language.
- Emgu.CV.Structure.Mxxx, which wraps the structures present in OpenCV, and the M stands for Manged. EmguCV uses also structures present in .Net for representing structures of OpenCV.

  For example, the *MCvScalar* represents the managed structure equivalent to *CvScalar* in OpenCV, meanwhile *System.Drawing.Point*, which is a .Net structure, is used for representing the OpenCV structure *CvPoint*.
- Emgu.CV.CvEnum, which wraps the OpenCV enumerations.

Since version 3.0 of OpenCV, the *IplImage* class has slowly been abandoned, replaced by the newer *Mat* class. As a result, EmguCV also had to adapt, using the new *Mat* class. The old *Image<,>* class remained available, however for retro compatibility, although the use of the *Mat* class is recommended. There are some differences between the *Mat* class and the *Image<,>* class in EmguCV. First of all, in the *Mat* class, there is no constraint on the memory allocation, which means that it is not mandatory to pre-allocate memory, but an empty object can be used. On the other hand, a *Mat* object can change its dimension during its lifetime, so there is no method that can access the pixel like it were a managed array, which is possible in the deprecated Image<,> class, using the Data method. To overcome this problem, one can transform the *Mat* object into an *Image<,>* object and then use the *Data* method to be able to use the image as if it were an array. This method is, however, rather slow, but also the safest. There is, in fact, also another faster, but much more dangerous solution, which forces the use of pinned managed memory. This way, if the new *Mat* object is created with an incorrect size, a solution with all zeros is obtained and no exception is thrown, not allowing programmers to know the error.

Stitching algorithm

OpenCV provides a class that can do all the work for the image stitching, which is *Stitcher*. Using this class, it is necessary to write only few rows of code to obtain the final result, but, the provided method returns only the final panorama, without all the matrices used for the transformation of input images into the computed panorama.

The pipeline used by the provided class is shown in the Figure 17:

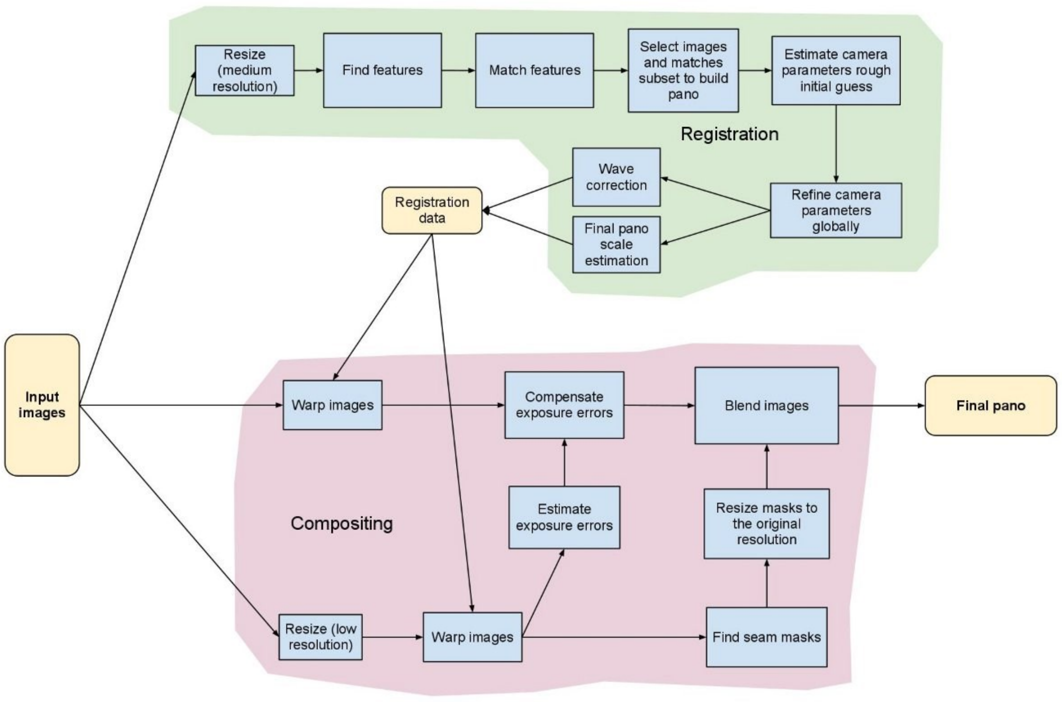

Resize (medium resolution)
Find features
Match features
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Wave correction
Registration data
Refine camera parameters globally
Final pano scale estimation
Input images
Warp images
Compensate exposure errors
Blend images
Final pano
Compositing
Estimate exposure errors
Resize masks to the original resolution
Resize (low resolution)
Warp images
Find seam masks

Figure 17: OpenCV stitching pipeline taken from OpenCV documentation.

In this case of study, the matrices were important to know since they were used in the computation of the rotation and translation of the matrices for the points clouds. Since it was necessary to know all the transformation matrices, the provided class was not used, and the image stitching pipeline was manually implemented, following these subsequently steps:

1. Keypoints detection.
2. Matching keypoints.
3. Homography estimation.
4. Panorama computation.

Since it was necessary to merge more possible images, it was decided to search for the best match between the computed panorama and all the images.

Keypoints detection

The first step implemented in the stitching process was the computation and detection of the keypoints for each image.

For computing the keypoints was used the SIFT (*Scale-Invariant Feature Transform*) algorithm [21], developed by David Lowe in 1999, because it is a very reliable keypoint detector and descriptor. SIFT is powerful, since it is invariant to rotation, translation, uniform scaling, illumination changes, and, partially, to affine distortion. Additionally, SIFT features are robust to occlusions, and very dense, since many features can be found also in small objects. Even the computation of the SIFT features is quite fast, despite having different steps to do before creating the final features, in fact, for computing 1000 SIFT keys are necessary less than 1 second.

The first step in the computation of the SIFT features is to find key locations, which is done by performing a difference of Gaussian (DoG) applied in scale space and then selecting the key locations at maxima and minima of this DoG, because they represent points of high variation. Scale space is divided into octaves and for each image contained into an octave, it is performed a gaussian smoothing, in the vertical and horizontal directions. The gaussian smoothing performed for each image of an octave has its σ increased: $k\sigma, k^2\sigma, k^3\sigma$.

When moving from one octave to the next, the images are down sampled, which means that although the same Gaussian filter is always used, in the next octave this filter behaves as if it had a double σ. To find the keypoints, it is necessary to calculate the DoG, an operation that is conducted for each octave in the space-scale representation by subtracting the images within each octave two by two, after applying the Gaussian filter, as shown in the Figure 18.

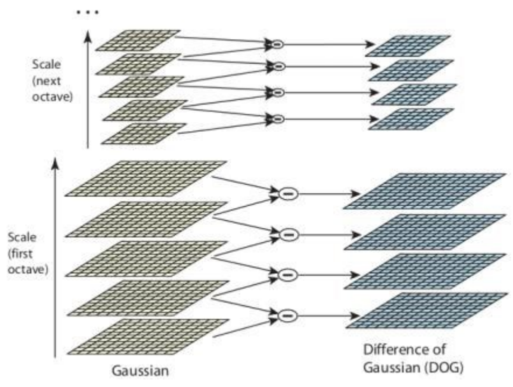


Figure 18: Difference of Gaussian (DoG).[11]

The maxima and minima of the DoG are calculated by comparing each pixel with its 8 neighbours at the same level of the pyramid composing the scale space. The pixel is also

[11] Image source: D. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints.*

checked against all 9 corresponding pixels in the other neighbouring levels of the octave. In practice, the pixel under consideration is checked against its neighbours in a cube of size 3x3x3, as shown in the Figure 19. If the pixel turns out to be either a maximum or a minimum, then it is a candidate to be a SIFT keypoint.

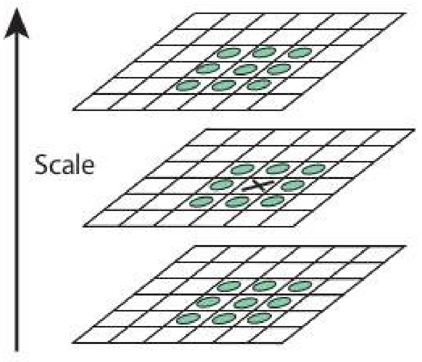


Figure 19: analysed point and its neighbours.[12]

This way of finding keypoints produce a large number of results, which must be refined, since it is necessary to eliminate any low-contrast and edge keypoints. Using the quadratic Taylor expansion of the DoG for obtaining a better position of the extrema, it is checked if its intensity is greater than a threshold. If this condition is not satisfied, then the extrema is discarder. Since the DoG has a strong response to the presence of edges in the images, it is necessary to eliminate these extrema, since they are not a good keypoint choice. For doing this elimination, it is used a 2x2 Hessian matrix.

Once the useless keypoints are eliminated, the second step for the computation of the SIFT features is the orientation assignment, that is the key step to achieve the invariance to rotation. For each detected keypoint, now, it is necessary to compute the gradient orientation, *θ(x, y)*, and the gradient magnitude, *m(x, y)*, for each pixel of its neighbourhood in the scale image L(σ), which is the image that had generated the keypoint. The gradient magnitude is computed according to Equation 6, meanwhile the gradient orientation is computed according to Equation 7.

[12] Image source: D. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints.*

$$m(x,y) = \sqrt{\left(L(x+1,y) - L(x-1,y)\right)^2 + \left(L(x,y+1) + L(x,y-1)\right)^2}$$

Equation 6: gradient magnitude computation.

$$\theta(x,y) = Atan2\left(L(x,y+1) + L(x,y-1), L(x+1,y) - L(x-1,y)\right)$$

Equation 7: gradient orientation computation.

When the computation is done, the gradient magnitudes are weighted with a zero mean gaussian centered in (x, y) with standard deviation equals to 1.5*σ, meanwhile the gradient orientations are quantized with a 10° step and their magnitude is stored into a 36 bins histogram. At the end, the highest peak of the histogram corresponds to final orientation of the keypoint. If other peaks greater that 80% of the highest peak are found, then for each of them a keypoint is generated, with same location and scale, but with different orientation.

Now, each keypoint has its position, its scale, and its orientation. The following step is to generate the keypoint descriptor. This last step aims to describe in a unique way the keypoints, such that it will be highly distinctive one from the other. Taken the scaled image closed to the scale of the keypoint, it is computed the gradient magnitude and orientation around the keypoint, usually in a neighbourhood of 16x16 pixels, and the gradient orientations are rotated according to the orientation of the keypoint. As in the step before, gradient magnitudes are weighted with a zero mean gaussian centered in (x, y) and with standard deviation 1.5 * σ. Now, the area around the keypoint is divided into smaller 4x4 regions, and for each of them an orientational histogram having 45° steps for each bean is computed. All the keypoints in the smaller region vote with their orientation and magnitude, as before. Now, the orientational histogram is stored as a vector, normalized, thresholded, and normalized again. This is done in order to reduce problems due to illumination changes in images. The normalized vector is now the keypoint descriptor, which is returned with the keypoint position, scale, and orientation.

The SIFT algorithm applied to the images collected by the IFM camera produced excellent results, both in terms of computational time and the number of keypoints detected, as shown in Figure 20.

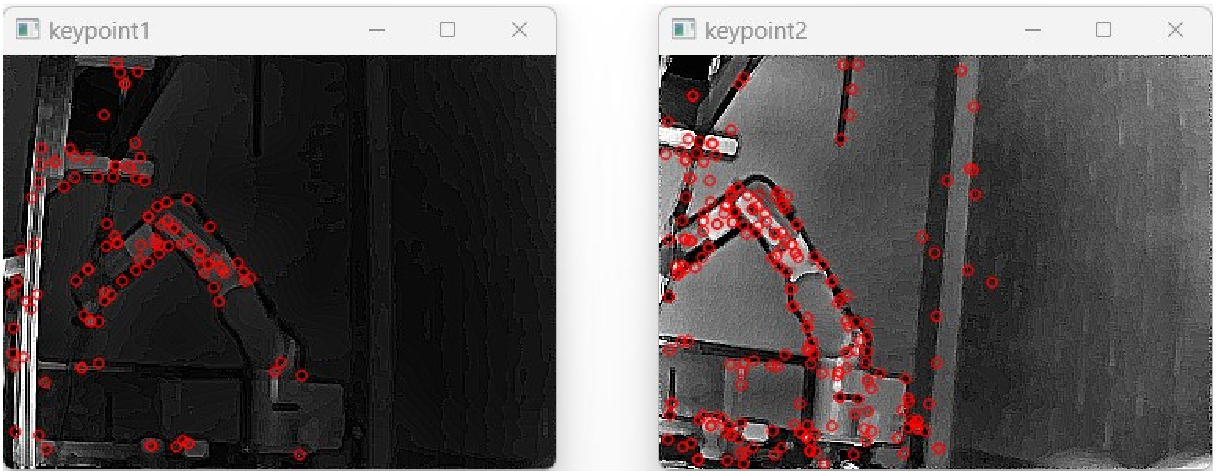


Figure 20: keypoints detected using SIFT.

Matching keypoints

When the keypoints are found in an image, the following step is to match them with the keypoints in another image for finding a possible correspondence between them. To be able to obtain a correspondence between the keypoints of an image and the keypoints of another image, a brute force matcher was used, i.e. a descriptor matcher that compares two sets of keypoint descriptors and generates a list of matches. It is called brute force matcher because it is not optimized but scans every single keypoint descriptor of the first image and searches for the match between all the keypoints of the second image. In the implementation of this thesis, the brute force matcher used was KnnMatch, as suggested by D. Lowe in his paper [22], setting k = 2 as the number of neighbours to find. To determine whether two keypoints were good matches, KnnMatch used the Euclidean distance, expressed in Equation 8, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the vectors representing the keypoints descriptors and are composed of n features each. This choice was also made in accordance with what D. Lowe wrote in his 2004 paper relating to SIFT features.

$$d(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Equation 8: Euclidian distance.

The result produced by the brute force matcher is shown in the Figure 21 and, as you can see, each keypoint in the first image is associated with two corresponding keypoints in the second image and vice versa.

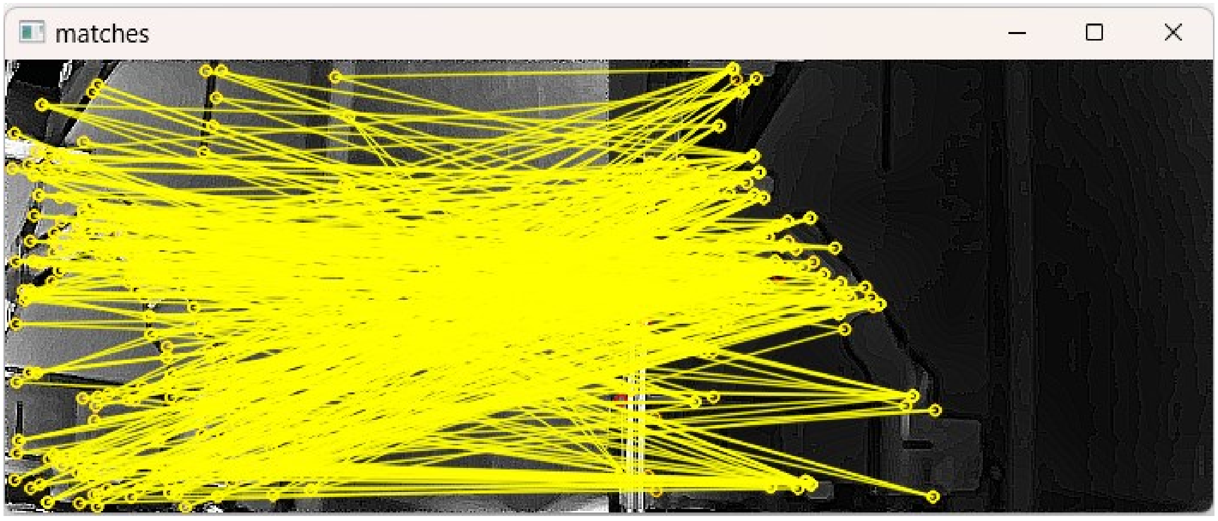
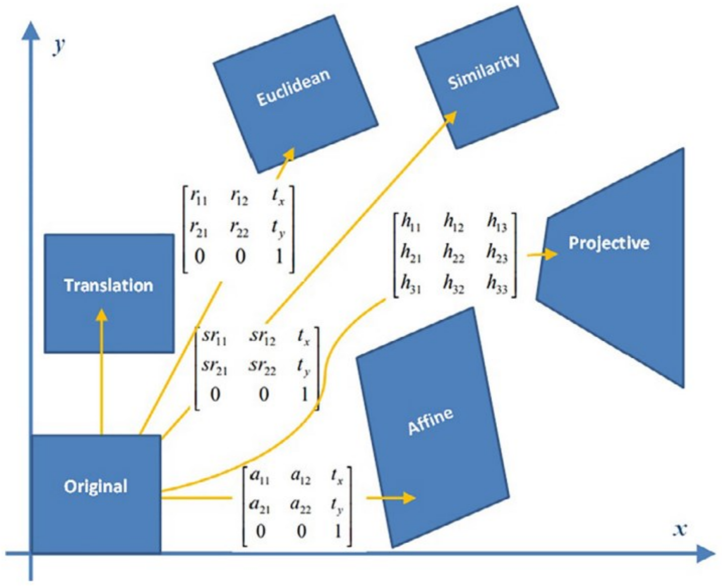


Figure 21: matched keypoints between two input images.

Homography estimation

Now that the matches between the keypoints have been calculated, it is necessary to find the transformation matrix that links the various keypoints in order to be able to merge the two images in question. Images can undergo different transformations [23] as they try to show a 3D world in a two-dimensional space, but also due to the different camera angle when taken. The different transformations, shown in the Figure 22, can be:

- Euclidian transformation. It is the simplest transformation, being composed of a rotation and a translation, providing 3 DoF.
- Similarity transformation. This transformation includes the Euclidean transformation but adds the ability to change the scale of the image. It therefore has 4 DoF.
- Affine transformation. This transformation includes the similarity transformation, but adds the ability to preserve parallel lines, i.e. the lines that were parallel in the original image are also parallel in the image that has undergone a similar transformation. It provides 6 DoF.
- Projective transformation, also called homography transformation, is a non-linear transformation that maps any two images of the same planar surface in space. It provides 8 DoF, and preserves collinearity, but does not preserve parallelism, lengths, and angles of lines.



y
x
Euclidean
Similarity
Projective
Translation
Affine
Original
$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$
$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Fig. 1. Different types of geometric transformations on an input image (original).

Figure 22: image of geometric transformations taken from "Perspective correction of building facade images for architectural applications".

These transformations have a hierarchical structure. In fact, the affine transformation includes the similarity transformation, exactly as the projection transformation includes the affine transformation. By setting $h_{31} = h_{32} = 0$ and $h_{33} = 1$ in the homography matrix it is possible to obtain the affine matrix. Therefore, to be able to link the various matched keypoints together, it was necessary to reconstruct the homography matrix between the two images. To do this, a method provided by OpenCV was used, namely the `GetHomographyMatrixFromMatchedFeatures()`[13], using the Features2DToolbox class. This method returns the homography matrix, if found, calculated using the RANSAC method.

Random sample consensus (RANSAC) [24] is an iterative method for finding the mathematical model parameters that best fits the input data set, input data that may contain outliers, that is, values that differ significantly from the input values. These outliers are not considered when estimating the parameters of the mathematical model. By increasing the number of iterations allowed to the algorithm, the greater the probability of its success. RANSAC works by iteratively following two steps:

1. selects, randomly, a subset of the starting data and constructs for these, providing the parameters that satisfy the required mathematical model.

[13] *GetHomographyMatrixFromMatchedFeatures()*, https://emgu.com/wiki/files/3.0.0/document/html/284ab4f5-1ea4-bae9-3ec5-4e1b1fa11d72.htm

2. calculate the number of inliers and outliers compared to the newly generated model. Inliers are data that reflect the model, while outliers are values that are clearly out of context.

By repeating these two steps many times, RANSAC provides the model that provides the highest number of inliers, as shown in Figure 23.

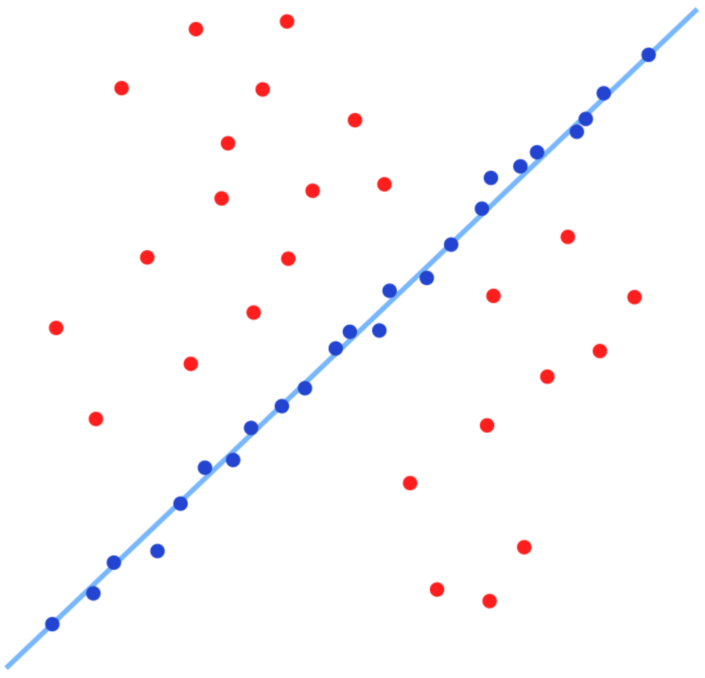


Figure 23: model fitted with RANSAC; outliers have no influence on the result.[14]

Let $\boldsymbol{u} = [x \quad y \quad 1]^T$ a point from the first image and let $\boldsymbol{u}' = [x' \quad y' \quad 1]^T$ the corresponding point in the second image. It is known that these two points are related to each other, since they were matched in the previous step. $\boldsymbol{u}$ and $\boldsymbol{u}'$ are related by the homography matrix H, according to the Equation 9:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Equation 9: homography relation between two points.

[14] *Image source*: https://en.wikipedia.org/wiki/Random_sample_consensus#/media/File:Fitted_line.svg

It is possible to rewrite Equation 9, such that it is obtained the expressions shown in Equation 10.

$$\begin{Bmatrix} xh_1 + yh_2 + h_3 = x' \\ xh_4 + yh_5 + h_6 = y' \\ xh_7 + yh_8 + h_9 = 1 \end{Bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & y'x & y'y & y' \\ x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ -y'x & -y'y & -y' & x'x & x'y & x' & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Leftrightarrow A\boldsymbol{h} = \mathbf{0}$$

Equation 10: rewritten homography relation.

From Equation 10, it is easy to see that the last row of matrix A is a linear combination of the first two rows, hence, each point correspondence contributes with 2 equations for the 9 unknown entries of **h**. Since the homography matrix is homogeneous, it is known the value of the last entry: $h_9 = 1$. This results in the need for at least 4 points for the calculation of the homographic matrix, where there are no 3 collinear points. For solving the homography matrix, 4 points are stacked up, as shown in Equation 11.

$$\begin{bmatrix} 0 & 0 & 0 & -x_1 & -y_1 & 1 & y_1'x_1 & y_1'y_1 & y_1' \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y_2'x_2 & y_2'y_2 & y_2' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2'x_2 & -x_2'y_2 & -x_2' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

Equation 11: stacking of the 4 points needed to calculate the homography matrix.

To estimate the homography matrix, therefore, RANSAC selects 4 random points correspondences from the vector containing the matches between the keypoints of the two images being worked on, such that no three points are collinear, and then solve the linear system shown in Equation 11. It then proceeds to compute inlier and outlier of the founded solution. The final model output is the one with lowest outlier. By repeating the process up to a maximum of 2000 times, RANSAC provides the best estimate of the homography matrix, calculated using the matches between keypoints.

The homography matrix calculated by OpenCV using RANSAC for the images shown in Figure 16 is shown in Equation 12.

$$H = \begin{bmatrix} 1{,}00135781000071 & -0{,}0877269188397514 & -40{,}0127309066868 \\ 0{,}109446594398558 & 0{,}992696439617528 & -22{,}7583202632904 \\ 0{,}000153741421464847 & -8{,}75170770843245E-05 & 1 \end{bmatrix}$$

Equation 12: homography matrix for the input images.

Final output

Using the newly calculated homography matrix, the two images were stitched together, as shown in Figure 24.

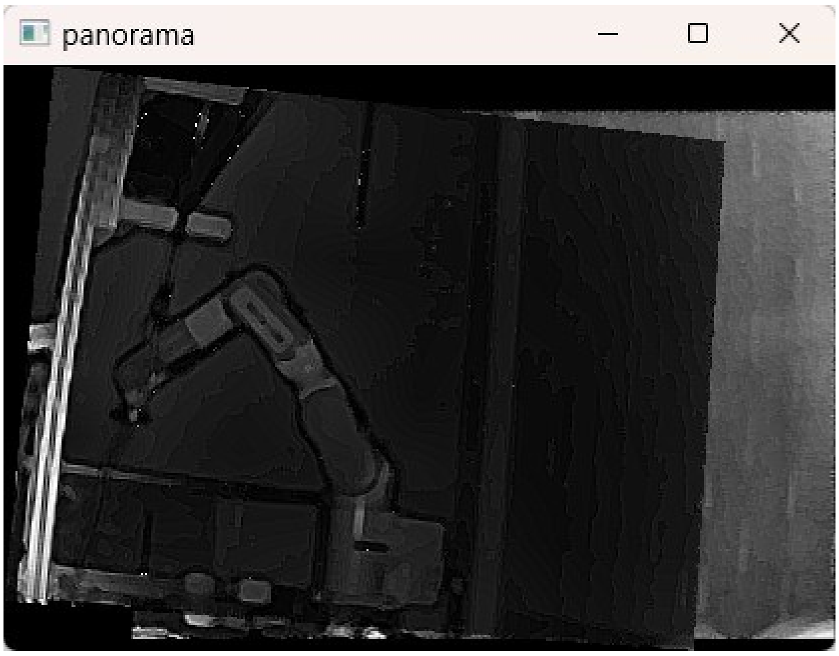


Figure 24: final result for the manually implemented stitching pipeline.

It was first necessary, however, to calculate the size of the final image and to do this the 4 corners of the first image were taken, transformed according to the homography matrix and from these the size of the final image was obtained, then adding the possible horizontal and vertical translation of the second image.

Merging images didn't always produce an optimal result, like the one in Figure 24, but sometimes it led to results that didn't make any sense. Taking the images shown in Figure 25 as an example, stitching them did not produce an acceptable result (Figure 26). This result, from a human point of view, is particularly incomprehensible, because it is easy to see how the two images can be superimposed on each other. However, the estimate of the homography matrix clearly produced an incorrect result, as the first image is exaggerated and causes the stripes that can be seen in the final output.

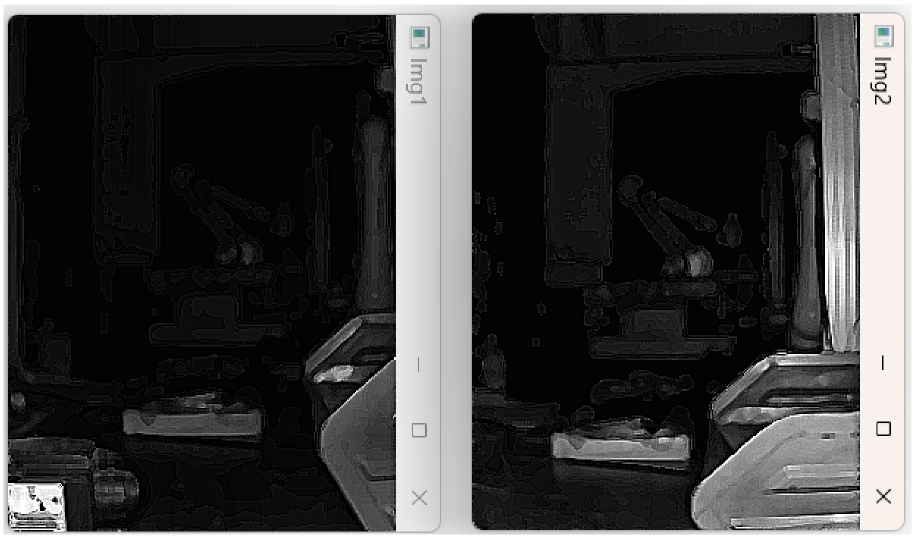


Figure 25: two input images.

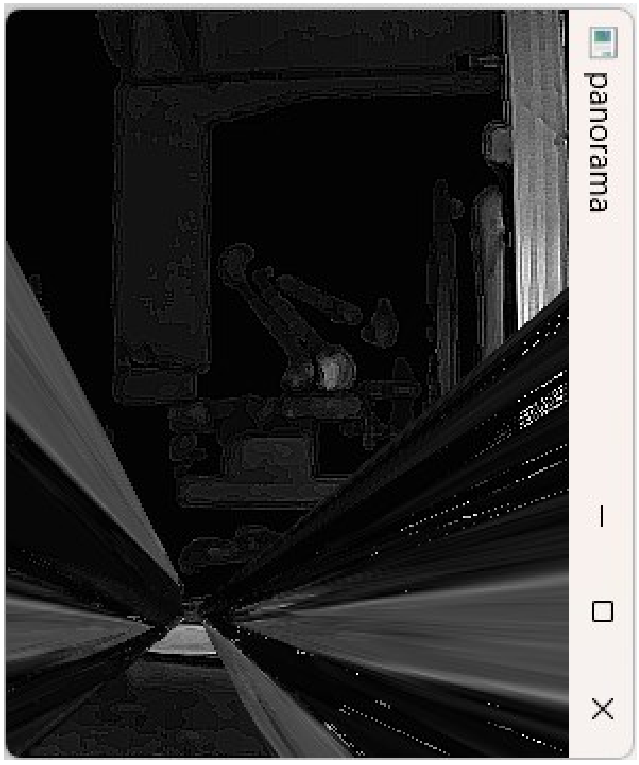


Figure 26: final output of stitching together the two previous images, where it is possible to see result of a wrong homography matrix.

Composition of the 3D panorama

Now that the stitching of the images has been performed, it was necessary to bring the transformations made to compose the two-dimensional image also into the three-dimensional world. The main purpose for which the IFM time-of-flight camera was used is for its ability to measure distance in space, through a light beam. The time taken by the light to travel the camera-object-camera path determines the distance of the object from the camera ToF itself, a distance which is then used to generate the point cloud relating to what is seen by the camera.

To transform the transformations from the 2D world to the 3D world, OpenCV and the methods that the library makes available were used. Once the roto-translation matrices were retrieved, they were applied to the objects as homogeneous matrices and visualized in the 3D world.

Point cloud

The IFM time-of-flight camera was used to generate the point cloud. The IFM measures the distance between the camera and the nearest surface point by point, obtaining a complete scan of what is in its field of view. The generated point cloud represents the space in front of the camera and is made up of a series of points, represented through the 3 Cartesian coordinates (x, y, z) and, with each scan, 92’928 points are acquired. Since it is needed a significant overlap of the images to be able to carry out the stitching, it is easy to deduce that the number of points to be processed is high, resulting in a slowdown in the manipulation of the data produced by the point cloud.

Specifically, Figure 27 represents the point cloud associated with the left image present in Figure 16. Figure 27 was viewed through a viewer[15] for viewing PCD files, and it is easy to recognize the profile of the robotic arm framed by the camera. The viewer used allowed each distance to be associated with a different color, allowing for a simpler vision of the different distances involved.

[15] *Online LIDAR point cloud viewer*: http://lidarview.com/

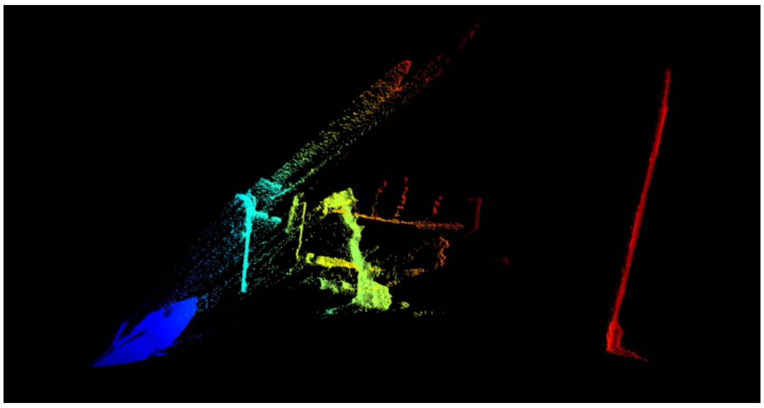


Figure 27: point cloud collected by the IFM camera.

Roto-translation computation

The homography matrix [25] allows to connect two images to each other, describing the relative movement between these two images, when the camera (or the observed object) moves. This means that it is possible to reconstruct the movement performed by the camera, based on the previously calculated homography matrix. This means that it is possible to reconstruct the movement performed by the camera, based on the previously calculated homography matrix. Having two points $\boldsymbol{x} = [x \quad y \quad 1]^T$ belonging to the first image and $\boldsymbol{x'} = [x' \quad y' \quad 1]^T$ belonging to the second image, they are linked by the formula expressed in Equation 13.

$$\boldsymbol{x'} = H\boldsymbol{x} \Leftrightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Equation 13: homography relation between two points.

The pinhole camera model [26], shown in Figure 28, is a mathematical representation of a camera, that describes the relationship between a point in the 3D world and its projection onto the image, using an ideal pinhole camera. The ideal pinhole camera is a camera where the light is passing through a single hole, without any lens, removing therefore the problem of lens distortions. This model is characterized by Equation 14, where $[u \quad v \quad 1]^T$ is a point in the image place, K is a 3x3 matrix representing the intrinsic parameters of the camera, M_{ex} is the 3x4 extrinsic matrix of the camera, and $[X \quad Y \quad Z \quad 1]^T$ is the point in the three-dimensional world. Both points are expressed in homogeneous coordinates.

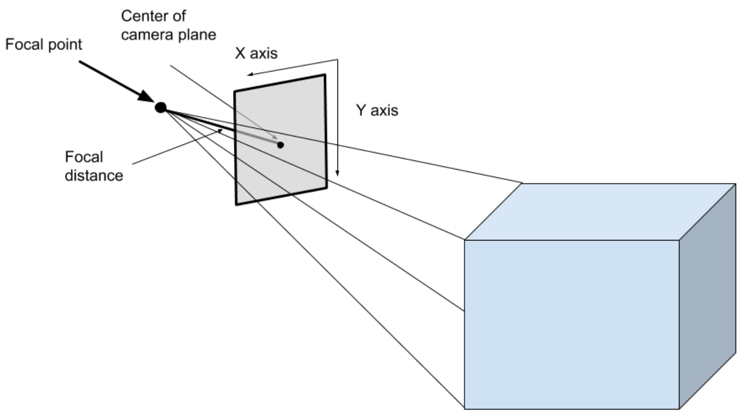


Figure 28: pinhole camera model.[16]

The intrinsic matrix K contains information about the camera intrinsic details, such as the focal length f and the position of the principal point, which is the point that represents the intersection between the image plane and the optical axis. The extrinsic matrix M_{ex} contains information about the pose of the camera, using a rotation matrix and a translation vector.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K * M_{ex} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Equation 14: pinhole camera model equation relating a point in the image plane to a point in the three-dimensional world.

From Figure 29, it is possible to observe how the homography matrix relates a pixel to the corresponding point of the observed object, assuming that the point rests on a plane with coordinate Z=0, according to Equation 15. From Equation 15, it is possible to see that the homography matrix contains the roto-translation of the camera.

The goal was to recover the roto-translation matrix from the homography matrix in order to apply these translations and rotations to the point cloud collected with the IFM. To obtain these matrices, a method provided by OpenCV was used: decomposeHomographyMat(). This function returns up to 4 possible sets of results, if found. This method uses the single value decomposition of the homography matrix, using the Zhang SVD-based decomposition.

[16] *Image source:* https://miro.medium.com/v2/resize:fit:1100/format:webp/1*zVeZ6F0RFwB4imnhOQ0EDg.png

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K * M_{ex} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{33} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{33} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & 0 & t_x \\ r_{21} & r_{22} & 0 & t_y \\ r_{31} & r_{32} & 0 & t_z \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Equation 15: derivation of homography matrix.

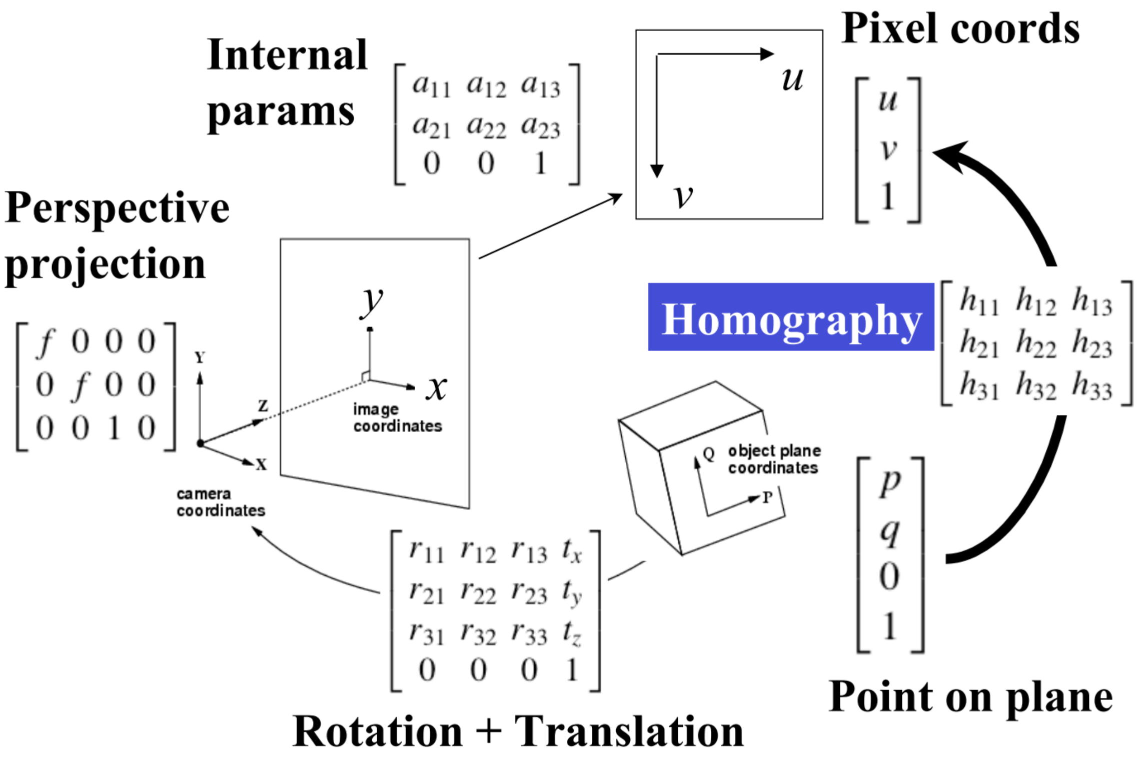

Internal params
$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$
u
v
Pixel coords
$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$
Perspective projection
$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Y
Z
X
camera coordinates
y
x
image coordinates
Homography
$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$
Q
object plane coordinates
P
$\begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$
$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation + Translation
Point on plane

Figure 29: planar projection.[17]

[17] *Image source:* https://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf

Once 4 roto-translation matrices were obtained, it was necessary to choose the correct matrices. We started by checking whether the image to be stitched with the other was to be added to the right or left of the base image, in order to check which translation vector satisfied this condition. Once the correct translation vector was found, the two rotation matrices associated with it were subjected to further control in turn. The rotation matrices, in fact, produced the same rotation, but one was performed on the image plane, while the other was performed with a rotation of 180° with respect to the image plane. The matrix that performed a rotation in front of the camera was therefore chosen.

Zhang SVD-based decomposition

The Zhang SVD-based decomposition [27] [28] is the SVD decomposition method used by OpenCV to obtain rotation and translation matrices from the homography matrix. Authors of this method starts the decomposition by computing the eigenvalues and eigenvectors of matrix $H^T H$, where H is the homography matrix:

$$\boldsymbol{H}^T\boldsymbol{H} = \boldsymbol{V}\boldsymbol{\Lambda}^2\boldsymbol{V}^T$$

with $\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}, \lambda_1 \geq \lambda_2 = 1 \geq \lambda_3$, and $\boldsymbol{V} = [\boldsymbol{v_1} \quad \boldsymbol{v_2} \quad \boldsymbol{v_3}]$.

After this, $\boldsymbol{t}^* = \boldsymbol{R}^T\boldsymbol{t}$ is defined as the normalized translation vector in the desired camera frame, authors propose the following relations:

$$||\boldsymbol{t}^*|| = \lambda_1 - \lambda_3; \qquad \boldsymbol{n}^T\boldsymbol{t}^* = \lambda_1\lambda_3 - 1$$

and

$$\begin{gathered} \boldsymbol{v_1} \propto \boldsymbol{v_1'} = \zeta_1\boldsymbol{t}^* + \boldsymbol{n} \\ \boldsymbol{v_2} \propto \boldsymbol{v_2'} = \boldsymbol{t}^* \times \boldsymbol{n} \\ \boldsymbol{v_3} \propto \boldsymbol{v_3'} = \zeta_3\boldsymbol{t}^* + \boldsymbol{n} \end{gathered}$$

where $\boldsymbol{v_i}$ are unitary vectors, while $\boldsymbol{v_i}'$ are not, and $\zeta_{1,3}$ are scalar function of $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$.

Authors propose then to use the follow expression to compute the first solution for $\boldsymbol{t}^*$ and $\boldsymbol{n}$:

$$\boldsymbol{t}^* = \pm \frac{\boldsymbol{v_1'} - \boldsymbol{v_3'}}{\zeta_1 - \zeta_3} \qquad \boldsymbol{n} = \pm \frac{\zeta_1\boldsymbol{v_3'} - \zeta_3\boldsymbol{v_1'}}{\zeta_1 - \zeta_3}$$

and the second solution:

$$\boldsymbol{t}^* = \pm \frac{\boldsymbol{v}_1' + \boldsymbol{v}_3'}{\zeta_1 - \zeta_3} \qquad \boldsymbol{n} = \pm \frac{\zeta_1 \boldsymbol{v}_3' + \zeta_3 \boldsymbol{v}_1'}{\zeta_1 - \zeta_3}$$

$\zeta_{1,3}$ are computed according to the following equation:

$$\zeta_{1,3} = \frac{1}{2\lambda_1\lambda_3}(-1 \pm \sqrt{1 + 4\frac{\lambda_1\lambda_3}{(\lambda_1 - \lambda_3)^2}}$$

The rotation matrix is then obtained with:

$$\boldsymbol{R} = \boldsymbol{H}(\boldsymbol{I} + \boldsymbol{t}^*\boldsymbol{n}^T)^{-1}$$

Final result

The roto-translation matrices calculated during image stitching were applied to the associated point clouds. The result can be seen in Figure 30.

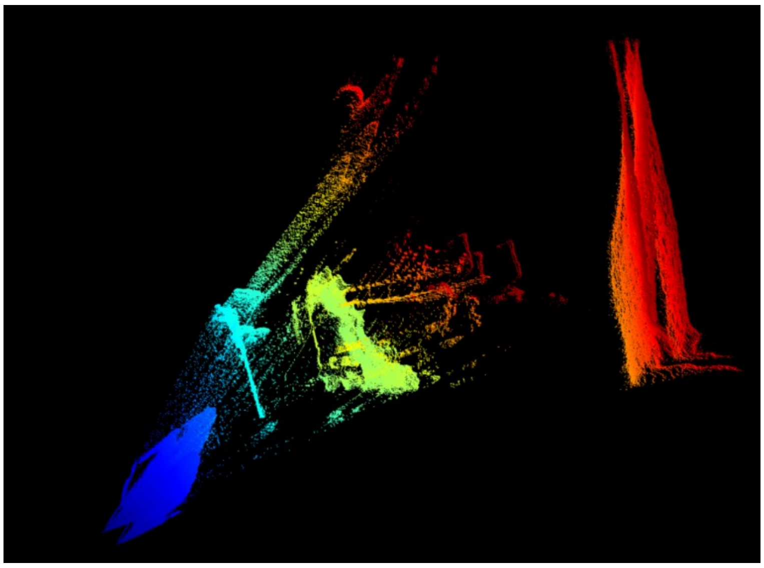


Figure 30: point cloud associated to the stitching of two input images.

It is possible to notice how the calculated roto-translation matrices do not produce a perfect result, especially as regards the Z axis. To reduce the error present on the Z axis, it was chosen to allow a refinement of the points, merging them together points that have the same x and y coordinates and similar z values. This operation is performed on a considerable number of points and can lead to a considerable slowdown of the system. Each single stitched image, in fact, produces 92,928 points and the more images that make up the final panorama, the more points will accumulate, making it necessary for several minutes to refine all the points of the cloud. An example of a refined cloud can be seen in Figure 31, where we have gone from 278784 points shown in Figure 30 to 208336 points, so 30% of the total points were merged.

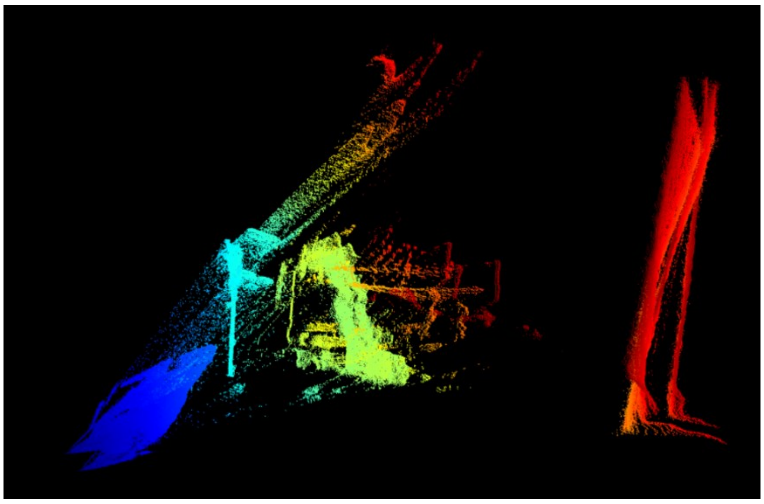


Figure 31: refined point cloud.

Conclusion

During the development of this thesis, three major goals were set: to determine the most suitable sensor between Tundra Tracker, Valve Index Controller, and HTC Vive Tracker 3.0, to construct the panoramic image using the images received from the ToF camera, and, finally, report the transformations undergone by the images for the construction of the final panorama in the three-dimensional world, applying the transformations to the point clouds associated with each image.

To determine which sensor was the most suitable and accurate, it was decided to perform three different tests for each sensor: the drift test, the noise test, and the dynamic test. The drift test was used to calculate how long after the sensor and the related tracking system stabilized and lost the drift shift typical of the ignition phase. The noise test was used to determine the accuracy of the position detected by each sensor and determine the error with which the position is reported. Both of these two tests were carried out with the sensor always fixed in position. The last test performed on the three sensors under study was the dynamic one, where the sensor had to correctly report the linear movement performed by the robotic arm to which it was connected and measure the distance travelled. The data received from these tests was processed and, at times, produced unexpected results. As for the drift test, the sensor that stabilized first was the HTC Vive Tracker 3.0, which took about 20 minutes to stabilize, compared to 30 minutes for the Valve Index Controller and 50 for the Tundra Tracker. In the noise tests, all three sensors worked very well, correctly detecting the position, with the exception of some recordings from the HTC Vive Tracker 3.0, where the sensor could be seen moving in the virtual world, although it was completely still in reality. As regards the dynamic tests, however, most of the time the expected graph was obtained, i.e. a graph having a bell shape, where the highest point of the bell coincided with the end of the acceleration of the robotic arm and the start of the deceleration necessary to perform the requested movement. Also in this case there were some unexpected results, where it was possible to notice a jump in the linear trajectory performed, which could also be observed in the graph representing the distance of each single position received with respect to the line travelled as it was no longer a graph bell with a gentle climb towards the point of maximum distance, but had an extreme and steep increase in distance from the line travelled. In carrying out the dynamic tests, to increase the precision obtained from the sensor, it was chosen to try to record the position using two Tundra Trackers. This did not lead to major changes because when the two sensors worked correctly, the improvement obtained by averaging the two sensors was minimal, while in the event of failure of one of the two, the sensor that was still working was unable to contain the error of the other sensor. From the tests carried out we also tried to understand how many base stations it was necessary to work with. Recordings were performed with one, two or three base stations, positioned in 4 different ways. It has been found that the sensor must be located in the center of the working area of the cameras, more precisely in the overlap area of the working area of the cameras themselves. To obtain precise results, remembering that it was required to obtain a precise position to 1mm, it is advisable to use at least two base stations. It is, in fact, possible to notice a significant increase in the precision of the detected position when switching from using one camera to two. On the contrary, going from two cameras to three did not lead to significant increases in precision. In conclusion, it is therefore possible to state that it is necessary to use at least two base stations to compress a greater area and prevent any obstruction, i.e. when the sensor is no longer visible from the camera, of the sensor with respect to the camera. For dynamic reasons, the sensors that produced the best results are HTC Vive Tracker 3.0 and Valve Index Controller, both in terms of measuring the distance travelled and in terms of measuring the line travelled.

For the second objective, that is the creation of the panorama using the images received from the IFM camera, it was necessary to manually implement the stitching pipeline. In fact, OpenCV automatically stitched the images, but did not return the homography matrix, necessary to recover the rotations and translations for the three-dimensional world. Manually implementing the stitch resulted in a number of issues. First of all, it was necessary to search among all the images for those having the greatest number of points in common to increase the possibility of finding the correct homography matrix and therefore being able to perform the final stitch. Once we found the two images with the most points in common, which correspond to the features found in the two images, we proceeded to create the panorama. The process was then repeated for the newly constructed panorama in relation to all the other images in the list. Once the final panorama was obtained for each list of images, these were stitched together in turn, to provide a single final panorama. However, the success rate of this final stitch was not very high, mainly due to the fact that each constructed panorama could have different dimensions and different brightness. Trying to give each image the same size and equalize the histograms didn't result in better performance. Using RANSAC to obtain the value of the homography matrix, it is possible that the obtained matrix does not produce correct image fusion, therefore making the final panorama unusable.

For the third goal of the thesis, it was necessary to find the rototranslation matrices associated with the homography matrix. Again, OpenCV, using the RANSAC algorithm, returned the rototranslation matrices. Being an estimate, they are not precise, in fact, there is not a perfect overlap of the point clouds, and you can notice the point clouds distinctly from each other, which is why a method has been implemented to refine the points, fusing together points having the same x and y coordinates and similar z coordinates. Although this refinement leads to a consistent elimination of points, it is also very slow having to compare every single point with all the others.

As can be seen from Figure 24 and Figure 30, the developed application is able to build the panorama and bring the transformations undergone by the images into the three-dimensional world.

Further developments can be made, such as finding a method to reduce the time required for refining the points present in the final point cloud, performing another study regarding the sensors to understand why the system performs random jumps and therefore being able to adopt countermeasures to reduce the impact that these jumps have on the system and try to improve image stitching, improving the success rate of the same.

Bibliography and website

[1] ISO, “ISO 8373:2012 - Robots and robotic devices — Vocabulary,” ISO, 2012. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en. [Accessed February 2024].

[2] R. O. Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures,” *Communications of the ACM,* vol. 15, pp. 11- 15, 1972.

[3] P. Viola and M. J. Jones, “Robust Real-Time Face Detection,” *International Journal of Computer Vision,* vol. 57, p. 137–154, 2004.

[4] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 770-778, 2016.

[5] R. Gonçalves, A. L. Pedrozo, E. S. F. Coutinho, I. Figueira and P. Ventura, “Efficacy of Virtual Reality Exposure Therapy in the Treatment of PTSD: A Systematic Review,” *PloS one,* vol. 12, 2012.

[6] Euclid Labs, “MARVIN SOFTWARE,” [Online]. Available: https://www.euclidlabs.it/marvin-no-code-robot-programming-software/. [Accessed February 2024].

[7] Valve, “VALVE INDEX BASE STATION 2.0,” [Online]. Available: https://www.valvesoftware.com/en/index/base-stations. [Accessed February 2024].

[8] SteamVR, “SteamVR Tracking,” [Online]. Available: https://partner.steamgames.com/vrlicensing#Tracking. [Accessed February 2024].

[9] Tundra, “TUNDRA TRACKER,” [Online]. Available: https://tundra-labs.com/products/tundra-tracker-1?variant=40250107494609. [Accessed February 2024].

[10] Valve, “VALVE INDEX CONTROLLER,” [Online]. Available: https://www.valvesoftware.com/en/index/controllers. [Accessed February 2024].

[11] HTC, "HTC VIVE TRACKER," [Online]. Available: https://www.vive.com/eu/accessory/tracker3/. [Accessed February 2024].

[12] HTC, "HTC Base Station 1.0," [Online]. Available: https://www.vive.com/eu/accessory/base-station/. [Accessed February 2024].

[13] HTC, "HTC Vive Pro," [Online]. Available: https://www.vive.com/eu/product/vive-pro2-full-kit/overview/. [Accessed February 2024].

[14] Kuka, "KR QUANTEC nano," [Online]. Available: https://www.kuka.com/it-it/prodotti-servizi/sistemi-robot/robot-industriali/kr-quantec-nano. [Accessed February 2024].

[15] ValveSoftware, "OpenVR SDK," [Online]. Available: https://github.com/ValveSoftware/openvr. [Accessed February 2024].

[16] P. Kumar, "What is a ToF sensor? What are the key components of a ToF camera?," e-con Systems, [Online]. Available: https://www.e-consystems.com/blog/camera/technology/what-is-a-time-of-flight-sensor-what-are-the-key-components-of-a-time-of-flight-camera/. [Accessed February 2024].

[17] L. Li, "Time-of-Flight Camera – An Introduction," *Technical White Paper, Texas Instruments,* January 2014, Revised May 2014.

[18] IFM, "IFM," [Online]. Available: https://www.ifm.com/it/it. [Accessed February 2024].

[19] OpenCV, "OpenCV," [Online]. Available: https://opencv.org/. [Accessed February 2024].

[20] Emgu, "Emgu," [Online]. Available: https://www.emgu.com/wiki/index.php/Main_Page. [Accessed February 2024].

[21] D. Lowe, «Object recognition from local scale-invariant features,» *Proceedings of the Seventh IEEE International Conference on Computer Vision,* vol. 2, pp. 1150 - 1157, 1999.

[22] D. Lowe, «Distinctive Image Features from Scale-Invariant Keypoints,» *International Journal of Computer Vision,* vol. 60, p. 91–110, 2004.

[23] A. Soycan and M. Soycan, “Perspective correction of building facade images for architectural applications,” *Engineering Science and Technology, an International Journal,* vol. 22, pp. 697-705, 2019.

[24] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM,* vol. 24, p. 381–395, 1981.

[25] D. Biswas, “Homography,” [Online]. Available: https://dibyendu-biswas.medium.com/homography-883f5bba92f2. [Accessed February 2024].

[26] A. Peri, “Using Homography for Pose Estimation in OpenCV,” Medium, 18 September 2020. [Online]. Available: https://medium.com/analytics-vidhya/using-homography-for-pose-estimation-in-opencv-a7215f260fdd. [Accessed February 2024].

[27] E. Malis and M. Vargas, “Deeper understanding of the homography decomposition for vision-based control,” *[Research Report] RR-6303, INRIA,* p. 90, 2007.

[28] Z. Zhang and A. Hanson, “3D Reconstruction based on homography mapping,” *Proc. ARPA96,* pp. 1007-1012, 1996.

Index of figures

Index of equations

Index of tables