FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

External Robot Localization in 6DoF

Nuno Filipe Leonor Nebeker

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Jorge Miranda de Sousa (Ph.D.)

October 27, 2015

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES | **2014/2015**

A Dissertação intitulada

"External Robot Localization in 6DoF"

foi aprovada em provas realizadas em 16-10-2015

o júri

Presidente Professor Doutor António Pedro Rodrigues Aguiar
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor José Luís Magalhães Lima
Professor Adjunto do Departamento de Eletrotécnia da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Professor Doutor Armando Jorge Miranda de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Nuno Filipe Leonor Nebeker

Faculdade de Engenharia da Universidade do Porto

Abstract

Mobile robots have shown the potential to change the way humans work, by taking on strenuous or repetitive tasks such as transporting heavy loads and by starting to interact with human workers. These robotic systems require robust and precise localization. External localization is relevant to complement self-localization and to provide valuable information when Automated Guided Vehicles (AGVs) move in irregular terrain.

Recent developments in Red, Green and Blue, and Depth (RGB-D) sensors have made valuable visual information available to researchers at affordable costs prompted new research into video tracking.

This masters thesis proposes a versatile and cost effective architecture for external 6 Degrees of Freedom (DoF) localization. The system uses RGB-D video tracking with multiple sensors and leverages Robot Operating System (ROS) and Point Cloud Library (PCL). The proposal includes sensor fusion strategies to improve overall localization performance. A suggested set-up would cost 2570€ (2015 prices), using 3 Kinect for Xbox One sensors and computers . The proposed system takes advantage of the target's natural geometry.

Simulation analysis, on a Xubuntu 14.04 system with a 2006 Intel Pentium 4 processor and 3 GB of RAM, shows the proposed tracker, based on Iterative Closest Point (ICP) point cloud registration, performs with maximum translation error under 5 cm in either axis and maximum rotation error under 5°, with an update frequency above 2 Hz and is robust to noise and partial occlusion.

In conjunction with the Cooperative Robot for Large Spaces Manufacturing (CARLoS) project, several datasets were collect and made publicly available. These include RGB-D data from two Kinect for Xbox 360 devices with different views of a moving AGV as well as the robot's sensor data and 6 DoF ground truth provided by a commercial motion capture system.

Resumo

Os robôs móveis têm revelado o seu potencial para mudar o modo como os humanos trabalham: assumem tarefas fisicamente exigentes e repetitivas como transportar cargas pesadas e começam a interagir com trabalhadores humanos. Estes sistemas robóticos exigem localização robusta e precisa. A localização externa é relevante para complementar auto-localização e para oferecer informação valiosa quando Automated Guided Vehicles (AGVs) (Veículos Autoguiados) se deslocam em terreno irregular.

Desenvolvimentos recentes em sensores Red, Green and Blue, and Depth (RGB-D) (cor e profundidade) vieram disponibilizar informação visual valiosa a investigadores a baixo custo levando a um novo interesse em *tracking* por vídeo.

Esta dissertação de mestrado propõe uma arquitetura versátil e com boa relação qualidade-preço para localização externa em 6 Degrees of Freedom (DoF) (Gruas de Liberdade). O sistema usa *tracking* por vídeo RGB-D com vários sensores e tira proveito de Robot Operating System (ROS) e Point Cloud Library (PCL). A proposta inclui estratégias de fusão de sensores para aumentar o desempenho de localização. Uma montagem sugerida custaria 2570€ (preços de 2015), usando 3 sensores Kinect for Xbox One e 3 computadores . O sistema proposto tira partido da geometria natural do alvo.

Revela-se por análise de simulações, num sistema Xubuntu 14.04 com processador Intel Pentium 4 de 2006 e 3 GB de RAM, que o *tracker* proposto, baseado em registo de nuvens de pontos por Iterative Closest Point (ICP), funciona com erro máximo em translação inferior a 5 cm em cada eixo e erro máximo em rotação inferior a 5° com uma frequência de atualização acima de 2 Hz e é robusto em relação a ruído e oclusão parcial.

Em conjunto com o projeto Cooperative Robot for Large Spaces Manufacturing (CARLoS), foram recolhidos e disponibilizados publicamente vários conjuntos de dados. Incluem-se dados RGB-D de dois dispositivos Kinect for Xbox 360 com vistas diferentes de um AGV em movimento bem como dados de sensores do robô e *ground truth* em 6 DoF conseguida com um sistema comercial de captura de movimento.

Acknowledgments

First of all, I would like to thank Armando Jorge Miranda de Sousa for his orientation and for cementing my fascination with automation in his lectures.

I also thank Carlos Miguel Correia da Costa for his patience and availability and for collaborating in capturing datasets for this dissertation.

I would like to acknowledge the researchers and staff at ROBIS and LABIOMEP for providing the great working environment I enjoyed.

I will also drop a line for my colleagues and friends, in particular José Pedro Fortuna Araújo and João Pedro Cruz Silva, for their valuable editorial notes, friendship and collaboration.

Finally, I am always grateful to my mother Maria da Luz and my brother Pedro for their constant support in my personal and academic life, and my girlfriend Soraia for the past few years of companionship, which I hope will extend long into the future.

Nuno Filipe Leonor Nebeker

"Without pain, without sacrifice we would have nothing.
Like the first monkey shot into space."

Chuck Palahniuk, *Fight Club*

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

API	Application Programming Interface
AGV	Automated Guided Vehicle
BRIEF	Binary Robust Independent Elementary Features
CARLoS	Cooperative Robot for Large Spaces Manufacturing
DoF	Degrees of Freedom
EKF	Extended Kalman Filter
FAST	Features From Accelerated Segment Test
FPFH	Fast Point Feature Histograms
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HSV	Hue, Value, Saturation
IR	Infrared
ICP	Iterative Closest Point
ORB	Oriented FAST Rotated BRIEF
RANSAC	Random Sample Consensus
ROI	Region of Interest
RGB	Red, Green and Blue color
RGB-D	Red, Green and Blue, and Depth
RMS	Root Mean Square
ROS	Robot Operating System
PCA	Principal Components Analysis
PDF	Probability Density Function
PCL	Point Cloud Library
PFH	Point Feature Histograms

SDK Software Development Kit

SAC-IA Sample Consensus Initial Alignment

SIFT Scale-Invariant Feature Transform

SIS Sequential Importance Sampling

SURF Speeded Up Robust Features

Chapter 1

Introduction

1.1 Context

This document is the final report for the Dissertation Thesis (Major in Automation) of the Master in Electrical and Computers Engineering at the Faculty of Engineering of the University of Porto.

The subject of the dissertation is "External Robot Localization in 6DoF" and it is supervised by Armando Jorge Sousa.

1.2 Defining the Problem

The current problem is tracking a rigid object in a dynamic environment, such that the object's poses are known for each frame of video, in 6 Degrees of Freedom (DoF). Pose in 6 DoF is defined as translation and rotation from a reference and can be expressed by x, y and z Cartesian coordinates and roll, pitch and yaw angles about fixed axes.

Object tracking is still a technologically complex problem in environments where service robots, such as robotic manipulators, operate, due to the need for quality results and for safety, as well as an interest in inexpensive systems.

In these scenarios, the need arises to use exterior localization techniques so that the real time control loop can interact with the objects of interest, identifying and tracking them in 3D with all available sensors.

Tracking the Cooperative Robot for Large Spaces Manufacturing (CARLoS) robot is an example scenario.

1.3 Objectives

The objective of this project is to develop a robotic system that will accomplish the tracking of mobile robots, mobile manipulators and other goods in semi-structured environments with real-time restrictions, exploring the advantages of the most recent Red, Green and Blue, and Depth (RGB-D) sensors.

1.4 Performance Metrics

A tracking system is evaluated on a number of performance measures. These include measurement error, update delay and frequency and versatility.

1.4.1 Measurement Error

Measurement error is defined as the error between a target's actual pose and that which is reported by the tracker. This is divided between the components of pose (translation and rotation), and quantified in the respective units, meter and radian.

Relevant quantities are the maximum and average error for any point in the object's trajectory and the accumulated and Root Mean Square (RMS) error along the entire trajectory.

1.4.2 Time delay

Object tracking is intended as a real-time on-line process, but the object's pose is calculated in finite time, resulting in a time delay between an object being in a certain pose and that pose being identified — lag — and a limited update frequency.

The number of arithmetic operations per iteration is another relevant way to express how computationally expensive an algorithm is [1] that does not depend on the performance of the machine running it.

1.4.3 Versatility

An ideal tracking system would be able to track an unlimited number of objects of any type, but this is not possible. Knowing this, one can classify different systems on their versatility by answering the following questions:

- How many objects can be tracked?
- What types of objects can be tracked? Are there restrictions in color, shape or size?
- Does the system tolerate occlusions?
- Can the system be easily adapted to new situations?

A system that can track more objects with fewer restrictions to their nature, handles occlusions and can be reconfigured is superior.

1.5 Project Goals

The final result should be a tracking system for use in external robot localization with the following characteristics:

- Capable of tracking one rigid object or the main rigid body of an articulated object, indoors;

- Capable of tracking in full 6 DoF without assumptions on the target moving on a floor plane;
- Maximum translation error of 5 cm or 5×10^{-2} m in either axis;
- Maximum rotation error of 5° or 8.7×10^{-2} rad;
- Minimum update frequency of 2 Hz.

1.6 Structure

After the Introduction, this document consists of 6 chapters.

Chapter 2 introduces the state of the art in the field.

Chapter 3 outlines relevant tools and equipment used in development and the proposed solution.

Chapter 4 presents the proposed solution for the problem.

Chapter 5 outlines development steps and results.

Chapter 6 presents data collection efforts in a case study where the solution could to be applied, as a basis for future development.

Finally, Chapter 7 introduces the conclusions and outlines future work.

Chapter 2

State of the Art

2.1 Introduction

This chapter first reviews a general survey on object tracking [2], then specific algorithms of note and finally recent work in 6 Degrees of Freedom (DoF) video object tracking.

2.2 Tracking Survey

This section is based on [2].

The first consideration in tracking is object representation. This can be shape-based (points, geometric primitives, silhouette and contours, articulated shape models or skeletal models) or appearance-based (probability densities of object appearance, templates, active appearance models, multiview appearance models).

The second consideration is feature selection for tracking. Distinctive features can include color, edges, optical flow and textures.

The first step of tracking is object detection. This can be accomplished through point detectors, such as the Harris detector and Scale-Invariant Feature Transform (SIFT) (discussed in Section 2.3.5.1), through background subtraction (using frame differencing or background models), through segmentation (such as mean-shift clustering, image segmentation using graph-cuts, active contours) and through supervised learning (for example, adaptive boosting, support vector machines).

Tracking, generating the trajectory of an object over time, can be performed jointly with or separately from object detection. This means the detection process can use information from previous frames or not.

The method used greatly depends on the object representation and both depend on the application.

2.2.1 Point Tracking

When the object is represented by a point, the possible methods can be divided into deterministic and probabilistic.

Deterministic methods apply constraints such as assuming, in frame t, an object will be close to its position in frame $t-1$, assuming an upper bound on an object's velocity or acceleration or assuming that objects in a group (particularly when these represent a single target) will have a similar velocity. Specifically for targets represented by several points, a common assumption is that the object is rigid, so the distance between its points will be constant.

Probabilistic methods include Kalman filters and particle filters (discussed in Section 2.3.2 and Section 2.3.3, respectively). Other methods deal specifically with tracking several objects simultaneously. When using the Kalman filter, the tracking problem is defined in state-space and the target's state is assumed to be affected by Gaussian error. If this assumption holds, the filter is optimal. Particle filters do not make this assumption and use several particles to represent different possibilities for the target's state. These are weighed by a probability function, such that the object's pose is determined by adding each weighted particle's contribution.

2.2.2 Kernel Tracking

Kernel tracking refers to tracking objects that are described in terms of their appearance. Kernel tracking methods include template matching or multi-view appearance models.

Possible templates include actual image intensity and color, gradient or color histogram or other representation that can be calculated within a region. Different characteristics can be combined, for example through principal component analysis (discussed in Section 2.3.4). These are generated on-line and, generally, used to track the object to the next frame in the vicinity of its previous position.

Multi-view models are generated off-line and represent appearance from different perspectives. Trackers based on these models are better prepared for sudden changes im perspective.

2.2.3 Silhouette Tracking

Non-rigid objects with complex shapes are represented by silhouettes and tracked either by shape matching, the tracker searches each new frame for a shape similar to the model generated in the previous frame and then updates it or by contour tracking, where the target's contour evolves from one frame to the next.

2.3 Algorithms of Note

This section describes several algorithms that are often used as part of tracking systems and in other applications.

2.3.1 Iterative Closest Point

Iterative Closest Point (ICP) is a method for registration - the process of finding a spatial transformation that aligns two point sets - of 3D shapes, described in [3].

The algorithm takes a model shape and a data shape and returns the transformation in 6 DoF that minimizes the distance metric between the shapes.

The steps are finding the distance between shapes — the same as the distance between their closest point pair —, computing the registration that matches the closest points, applying it and calculating the new mean-square error. If the error is lower than a set limit, the process is terminated, otherwise a new iteration starts, repeating these steps.

The process will continue until a local error minimum is reached. The authors point out that this means the registration can be incorrect if there is a great difference in orientation between model and data. To ensure the correct transformation is found, one must either start with a reasonable estimate of rotation or test several possibilities, the number of which depends on object complexity.

The estimate for translation, however, can be relaxed, as the algorithm converges correctly for very different initial estimates.

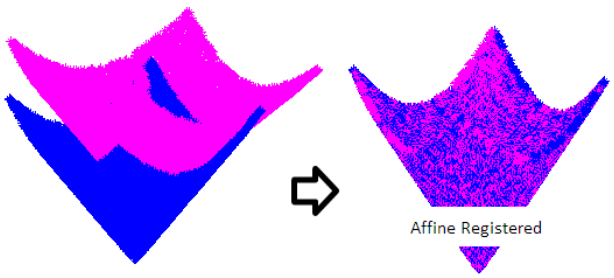

Figure 2.1: ICP demonstration [4]

ICP is extensively used in tracking, such as in [5] with range finders and [6] with Red, Green and Blue, and Depth (RGB-D) video and in self-localization such as in [7].

For tracking, the objective is to find the rigid transformation between the observed target and the model. This is the same as its pose compared to the scene's coordinate space.

In self-localization, the objective is to find the rigid transformation between the environment model — the map — and the observed environment, which corresponds to the observer's pose within the environment.

2.3.2 Kalman Filter

The Kalman filter, first proposed in [8] and explained in a more approachable fashion in [9] estimates the state of a linear system that is affected by process and measurement noise. It is optimal when the noise is Gaussian.

For a linear system in state-space form, the filter calculates estimates of the state variables and of the outputs, as well as the error covariance matrix, which describes estimate uncertainty. These

are based on a system model, estimates of noise covariance matrices and initial estimates for the system's state and error covariance.

The filter algorithm has two steps: a time-update or prediction step and a measurement-update or correction step.

The prediction step uses information up to $t-1$ to estimate system variables in t.

The correction step begins by calculating the Kalman gain, based on system and noise parameters and on the current error covariance. The corrected estimate for t is then calculated by adding the prediction from $t-1$ and the measurement from t, compared to the expected output, considering the predicted state and weighed by the Kalman gain. Finally, the error covariance is updated based on its previous prediction, system model and Kalman gain.

Due to its recursive nature, the Kalman filter algorithm only needs to store the system and noise models, the latest predictions and corrected estimates, their respective covariances and the latest input and output.

For nonlinear systems, Extended Kalman Filter (EKF) linearizes the system around the current mean and covariance. This filter is no longer optimal, but is still the most used estimation algorithm for nonlinear systems. This is noted in [10], as well as the fact that it is only effective if the sampling and update time is small enough for the dynamics to be properly approximated by linearization at each step.

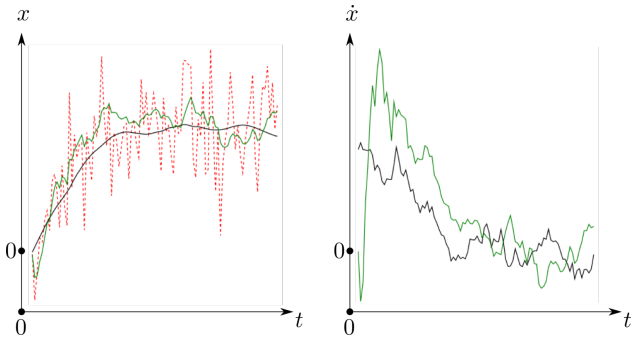

Figure 2.2: Kalman filter example — Black: truth, green: filtered process, red: observations [11]

The Kalman filter is widely used in tracking, as noted in [2]. Using a Kalman filter to predict the new pose of a tracked object, thereby saving time on new pose estimation, is proposed in [5].

2.3.3 Particle Filter

Sequential Monte Carlo estimation methods or particle filters, introduced in [12] (as the bootstrap filter) and explained in [13], are based on state-space models like the Kalman filter, but do not make assumptions on linearity.

In Baysian filters, the objective it to calculates a posterior Probability Density Function (PDF), based on available information including most recent measurements, that contains all the statistical information and is a complete solution to the problem.

A recursive algorithm, such as the Kalman filter, does this in two steps: prediction, where the prior PDF is calculated and correction, where the posterior PDF updates it with new measured data, acquiring better information on the system's state — based on all available information.

Analytically calculating this PDF is not possible, unless certain restrictions are considered. In the Kalman filter, the system is restricted to being linear and this function is restricted to a Gaussian distribution parametrized by a mean and a covariance, which are both calculated.

If the state-space is discrete and there is a finite number of states, the posterior density function can be calculated by summing the conditional probability of each state.

Without such restrictions, the PDF cannot be calculated, but it can be approximated.

Knowing there are several particle filter algorithms, this section will continue by explaining only the Sequential Importance Sampling (SIS) algorithm as this is sufficient for understanding the concept of particle filtering and it is the basis for most such methods.

This algorithm calculates the PDF in a number of samples, or particles. The final estimate results from each particle's contribution, affected by its weight.

The weight is the result of evaluating an importance density function, the design of which is the center of designing a particle filter for a given application.

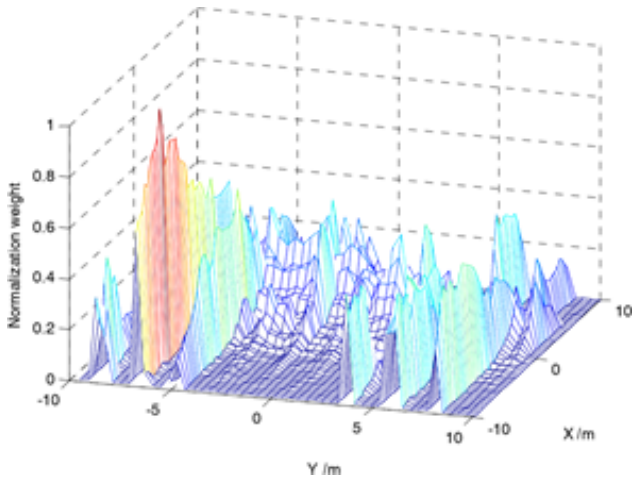

Figure 2.3: Particle filter example — particles for a robot's location [7]

As the algorithm is iterated, one particle tends to represent more of the PDF, while the others' weights approach zero - the degeneracy problem. This can require a re-sampling of particles to promote diversity.

Particle filters are widely used in localization, such as in [7] and in tracking such as in [14].

2.3.4 Principal Component Analysis

Principal Components Analysis (PCA) was first described in [15], named in [16] and explained in [17] is a method to fit lines and planes to measured data points, thereby finding the underlying relationships between variables, while reducing the problem's dimensionality.

The results of this analysis are a new orthonormal basis for the data, such that the each axis accounts, in sequence, for most of the data's variance. These axes constitute the principal components. Data points can then be projected into this base.

An algorithm for applying PCA to a dataset starts by centering the data on the origin, by subtracting its centroid, then calculates the principal components, either through eigenvalue decomposition of the covariance matrix or through singular value decomposition (SVD) of a form of the data matrix [17].

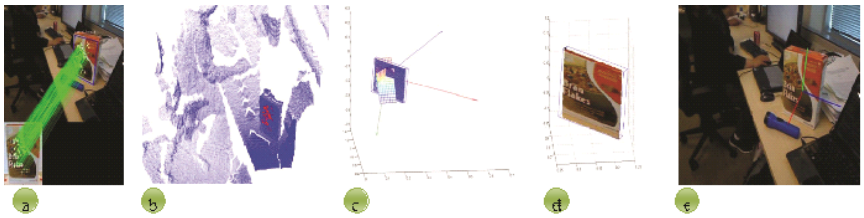

Figure 2.4: PCA in use (c) [18]

PCA is used in [18] to find a tracked object's orientation and as noted in section 2.2 to combine and compare appearance features.

2.3.5 Local Feature Descriptors

As mentioned in section 2.2, SIFT and other local feature descriptors are popular in model-based tracking. These aim to quantify the properties of points in images, so that they can be matched and objects can be recognized.

SIFT and Speeded Up Robust Features (SURF) also propose a key-point detector, which first selects points of interest. ! (!) relies on an established detector.

2.3.5.1 SIFT

SIFT was introduced in [19] and patented in the United States under [20]. It produces feature vectors from an image that are invariant to scale, translation, and rotation and nearly invariant to illumination changes and projection.

SIFT generates in the order of 1000 keys (features) for an image and requires as little as 3 to match to identify an object, so it can tolerate significant occlusion. Furthermore, enough translation is allowed so that 2D objects can be rotated up to 60° from the camera plane and 3D objects can be rotated up to 220° and still be identified.

2.3.5.2 SURF

SURF, introduced in [21] aims to be faster and as robust as SIFT and other previous descriptors.

In the interest of speed, SURF is not explicitly invariant to anisotropic scaling, and perspective effects.

Figure 2.5: SIFT features used in image recognition [19]

In the authors' tests, considering viewpoint changes, rotations, zooming and illumination variations, SURF outperformed SIFT in speed by 2 times and in recall for the same levels of precision.

Figure 2.6: Detected SURF features [21]

2.3.5.3 ORB

Oriented FAST Rotated BRIEF (ORB) was proposed in [22] with the intention of providing performance comparable to SIFT and greater speed (claimed to be two orders of magnitude higher than SURF).

It adapts the Features From Accelerated Segment Test (FAST) detector [23] with an added orientation component and scale invariance and the Binary Robust Independent Elementary Features (BRIEF) descriptor [24] with orientation invariance. Both detector and descriptor were designed with efficiency and speed in mind. In particular, BRIEF descriptors are represented by

Table 2.1: Comparison between SIFT, SURF and ORB (matches) [22]

	Inliers (%)	*N* points
Magazines data set		
ORB	36.180	548.50
SURF	38.305	513.55
SIFT	34.010	584.15
Boat data set		
ORB	45.8	789
SURF	25.6	795
SIFT	30.2	714

Table 2.2: Comparison between SIFT, SURF and ORB (speed) [22]

Detector	ORB	SURF	SIFT
Time per frame (ms)	15.3	217.3	5228.7

binary strings, so calculating the difference between two features is reduced to calculating the Hamming distance between their descriptors, a very efficient operation.

Due to its high speed it is well suited for real-time operations.

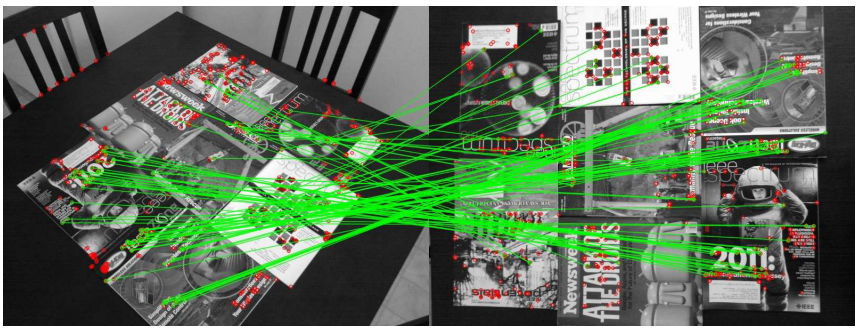

Figure 2.7: ORB features used in image matching [22]

The evaluation of ORB offers a comparison between this descriptor and SIFT and SURF. Comparison reults for matched points for two data set are shown in Table 2.1 and average detection times per frame from a set of 2686 are shown in Table 2.2.

2.3.6 FPFH feature descriptor and SAC-IA registration

Feature descriptors can also be determined for 3D points, such as Fast Point Feature Histograms (FPFH), introduced in [25] (based on Point Feature Histograms (PFH) in [26, 27]).

This descriptor relates geometrical relationships between a point and its neighbors and is calculated based upon point coordinates and surface normals. FPFH improves over PFH by caching previous results and streamlining some formulations.

Six DoF registration algorithms such as ICP risk falling into local minima in the optimization process returning incorrect results [3, 25]. Sample Consensus Initial Alignment (SAC-IA), also

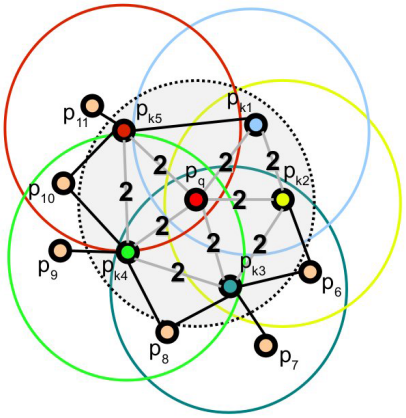

Figure 2.8: FPFH region of influence for point P_q [25]

introduced in [25], based on Random Sample Consensus (RANSAC) [28] attempts to correct this by considering and ranking several sets of correspondences.

SAC-IA selects many samples from one point cloud, but guarantees their pairwise distances are larger than a user-set distance. The corresponding point in the other point cloud is randomly selected from a set of points whose feature histograms are similar. These correspondences are used to calculate the 6 DoF transform between the point clouds and an error metric.

Repeating these steps and using an optimization algorithm on the best transform yields the final result.

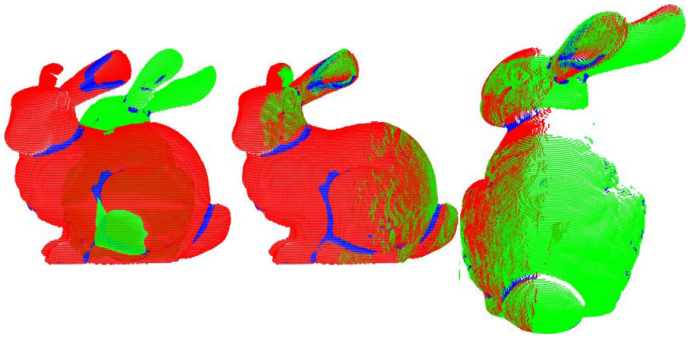

Figure 2.9: SAC-IA registration — two partial view on left; alignment result on right [25]

2.4 Recent Work in 6DoF RGBD Video Object Tracking

2.4.1 Shape and Model-Based Tracking in Stereo Video

An algorithm is proposed in [29] for shape-based and then iterative model-based 6 DoF pose estimation of single colored objects in stereo Red, Green and Blue color (RGB) images.

This algorithms first segments the target object, using Hue, Value, Saturation (HSV) color segmentation, separating a single color in the image. Initial translation is estimated through centroids of object in both captured images. Rotation is obtained from matching the left image with a multi-view model. This is accurate if the object is in the same position as in training.

To correct errors resulting from a difference in translation, a corrective rotation is calculated such that if the object were affected by this new rotation, its appearance would be the same as if it were in fact in the training position (center of the left image).

This correction affects the translation, so that is recalculated, using simulated views from a 3D model of the object and the process is repeated iteratively. The algorithm generally converges in two iterations.

Final orientation error depends on the angle resolution of training images.

This method was tested in simulation.

Based on this estimate, the algorithm proposed in [14] tracks arbitrarily shaped objects using a full 3D model in stereo video.

The algorithm consists of pose update, probability function, normalization and annealed particle filter.

The update consists in adding Gaussian noise to the current estimate, a translation vector and a rotation vector.

To calculate the probability function for each updated pose, object views are rendered from that pose and compared to the captured images via a bitwise AND operation on binary detected edges on both images.

The number of matching pixels is divided by the number of pixels in the rendered view edge image and this result is used in calculating the a-posteriori probabilities for all particles.

This technique allows for matching even with partial occlusions.

Because both normalizing over the number of edge pixels and normalizing over the total number of pixels in the image both present issues with unexpected edge filter responses and varying distance respectively, and additional weighted normalization step is performed.

The annealed particle filter runs several filters with fewer particles in layers. This aims to reduce the overall umber of particle evaluations.

Figure 2.10: Exemplary pose estimation [29] and tracking [14] results

2.4.2 Pose Estimation of Rigid Objects Using RGB-D Imagery

The algorithm proposed by [18] uses 480×640 RGB-D imagery from a Kinect sensor.

The object model consists of ORB features and their associated 3D positions on the objects, collected from different angles and distances.

ORB features in the captured image are matched with model features. A homography matrix is iteratively computed and then used to remove outliers.

A region of interest (ROI) is determined by fitting a 3D bounding box to the matching key-points in the captured point cloud. The oject points are segmented through clustering.

The object's 6D pose is determined in two steps. First, translation is calculated by taking the centroid of the object points and finding its depth in the camera space. Rotation is obtained through a bounding box along the principal axes of the point distribution.

Object detection is subsequent frames is accelerated by considering only a quadrangular ROI in the image space, corresponding to the previous 3D bounding box.

Objects are tacked at 15 Hz with a 92% detection rate.

Future work includes using GPU optimization and using contours to track untextured objects.

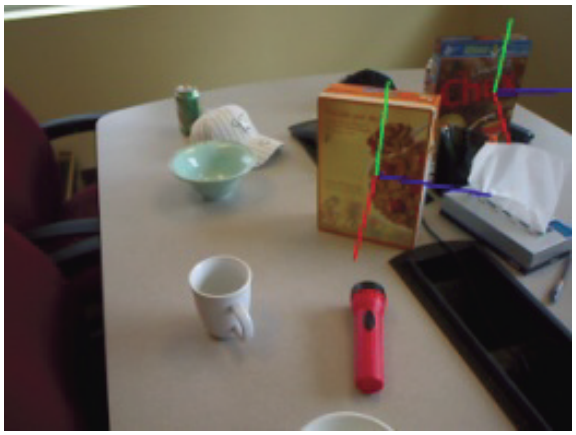

Figure 2.11: Exemplary results [18]

2.4.3 Princeton Tracking Benchmark and RGBD Tracking Algorithms

A dataset consisting of 100 RGB-D video samples and a 6 DoF tracking method are introduced in [6]. The samples were recorded with a Kinect sensor and used to evaluate the proposed baseline algorithms.

The authors present a set of algorithms based on the state of the art in 2D and 3D tracking.

The occlusion detection strategy is of particular note, due to its use of depth information: the depth data within the object's 2D bounding box is divided into an object region (depth range of target points, which are the closest to the camera) and a background region (further away). When other occluding objects enter the bounding box, their depth is lower.

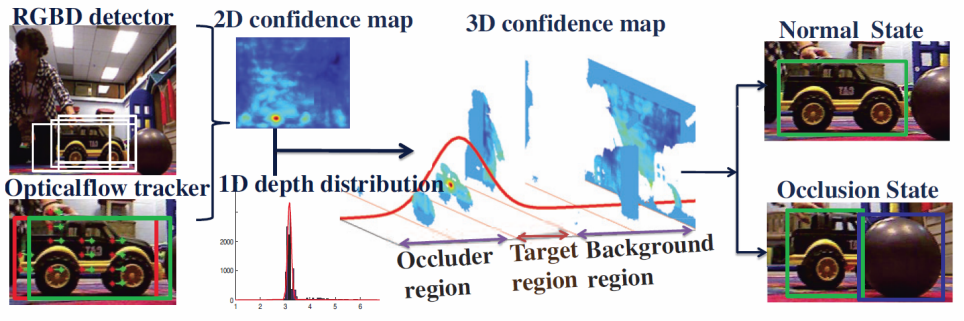

Figure 2.12: Occlusion handling [6]

2.5 Conclusion

This review of the state of the art in 6 DoF video tracking shows that this problem is non-trivial and that challenges remain. There are a number of significant design decisions that impact performance and accuracy, as well as the range of application of a tracker. Current efforts attempt to leverage new technology and are often tailored to specific targets and environments.

Chapter 3

Tools and Equipment

3.1 Introduction

Tackling any programming project involves finding the appropriate libraries, frameworks and tools. For a problem that is focused on robotics and computer vision, this means finding robotics middleware [30] that will provide an adequate hardware abstraction and common services including package management and communications (between a robot's services and between different robots), a computer vision library that provides real-time processing functions of 2D and 3D images and a complete sensor system that provides a quality imaging solution.

Finally, validation requires a solution for obtaining a Ground Truth trajectory.

3.2 ROS Robotics MiddleWare

Among current solutions for robotics middleware [30], Robot Operating System (ROS) stands out. It was designed to be: [31]

- "Peer-to-peer"
- "Tools-based"
- "Multi-lingual"
- "Thin"
- "Free and Open-Source"

ROS allows researchers to develop robotics software from hardware drivers to high-level decision making, by integrating different components by different developers.

3.2.1 Nodes and Packages

The peer-to-peer nature of ROS allows different processes to operate separately and communicate. These can be visualized in graph form and are called "nodes."

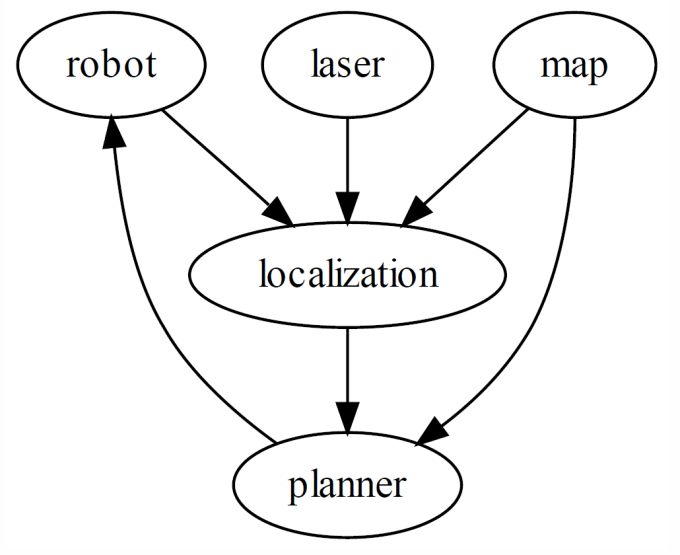

Figure 3.1: Typical ROS graph [31]

A set of nodes can be collected into a package, for example the nodes in Fig. 3.1 would constitute a `navigation` package. The `localization` node is of particular interest as it has the same purpose as this project: to provide other nodes with the robot's pose, relative to the map, using sensor information, in this case laser scans.

3.2.2 Communication

Communication between nodes uses messages — well-defined data structures. Communication follows a publisher/subscriber pattern, where any node can publish a message, on a topic, and any other can subscribe to this topic, at will. In the example of Fig. 3.1, node `localization` publishes a `pose` message to which `planner` subscribes. At the same time a `logger` node can subscribe to the same message and record the robot's trajectory.

3.2.3 Frame Transforms

Interfacing sensors, cameras, mobile robots, manipulators and the working environment requires a system to transform points and vectors between various frames.

ROS builds all these transforms into a transformation tree (the `tf2` system).

As described in the `tf2` documentation "A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time." [32]

Any node can add a transformation to the `tf2` tree by publish a transform message such as `geometry_msgs/TransformStamped`. Nodes can then request transformations between relevant frames.

Returning to Fig. 3.1, `map` might publish its message referenced to frame `world`, but `planner` could transparently access it in frame `base_link` (the robot's base) because `localization` publishes a transform between these frames to `tf2`.

3.2.4 Logging

ROS topics can be recorded for future use using the `bag` format. [33] This allow developers to replay sensor data, log relevant information and simulate nodes, by publishing their expected messages.

3.3 Point Cloud Library

The second problem can be solved with Point Cloud Library (PCL) [34]. It is a library that "contains state-of-the art algorithms for: filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation" [35] with point clouds and interfaces with ROS [36].

This library is used in ROS, for example, in [37].

The set of software libraries offered by PCL can be visualized in Fig. 3.2.

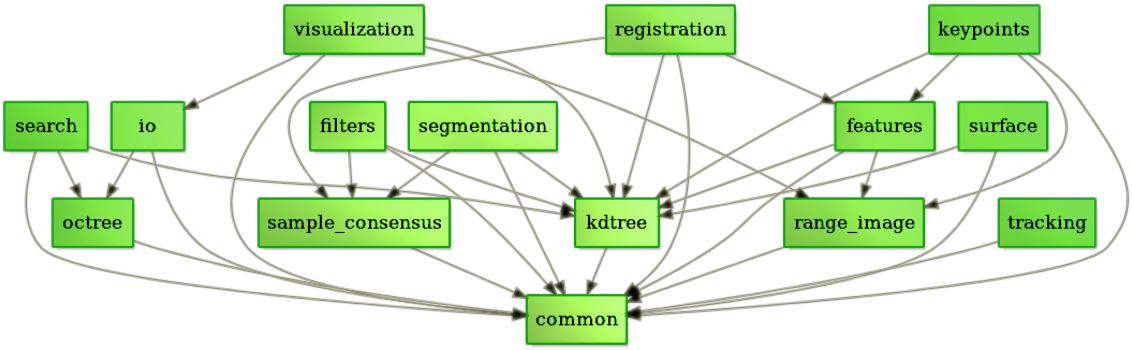

Figure 3.2: PCL libraries [34]

3.3.1 Point Cloud Data Type

PCL implements templated classes. `PointCloud<PointT>` is the class for storing point clouds. A point cloud ia a set of multidimensional (typically 3D) points that can include the Red, Green and Blue color (RGB) values for points.

3.3.2 Relevant Algorithms

PCL implements several registration algorithms including Iterative Closest Point (ICP) and Sample Consensus Initial Alignment (SAC-IA) built upon base class `Registration<PointSource, PointTarget, Scalar>`.

Filtering algorithms are based on the `Filter<PointT>` and include pass-through, statistical and voxel grid filters.

Table 3.1: Kinect for Xbox 360 characteristics [37]

Camera (RGB)	640 × 480 VGA, bit color to 32-RGB-@ 30 Hz)
Depth sensor	Structured light (Infrared (IR)) emitter (projector) and IR monochrome CMOS sensor (320 × 240 QVGA, bit to 16-levels deep -65.536 @ 30 Hz)
Angular field of view	57° horizontal and 43° vertical

3.4 Microsoft Kinect Devices

Microsoft's range of Kinect sensors have been extensively used in robotics, starting with the original Kinect for Xbox 360, for example in [37], [38], [39] and [40].

The characteristics of the Kinect for Xbox 360 are listed in Table 3.1.

The Kinect for Windows has similar components, but is designed to connect directly to computers running Microsoft Windows and features color images 1280 × 960 pixels [41] and depth images up to 640 × 480 @ 30 fps [42] with a depth range of 0.8 m to 4 m [43].

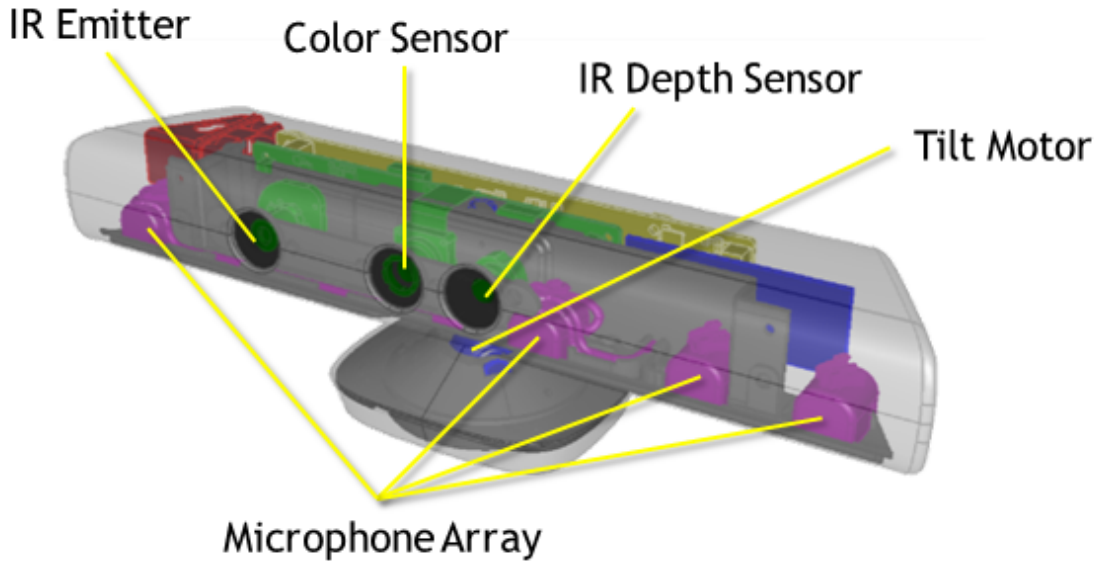

Figure 3.3: Kinect for Windows components [41]

The more recent Kinect for Xbox One can be connect to a computer with an extra adapter (with a total estimated retail price of €200, in Portugal) [44] and has improved featuresas listed in Table 3.2 and components in Fig. 3.4.

3.4.1 Linux/ROS Drivers

Microsoft offers an Software Development Kit (SDK) for Kinect [47] along with drivers for Microsoft windows. Research interest in this range of devices has led to the availability of several open source drivers for different platforms, including Linux, which is of particular interest for this project, as ROS runs on Linux operating systems.

Table 3.2: Kinect for Xbox One characteristics [45]

RGB camera	1080p at 30 Hz (15 Hz in low light)
Depth sensor	512×424 resolution at 30 Hz
IR sensor	Active sensor; 512×424 images at 30 Hz
Angular field of view	$70° \times 60°$

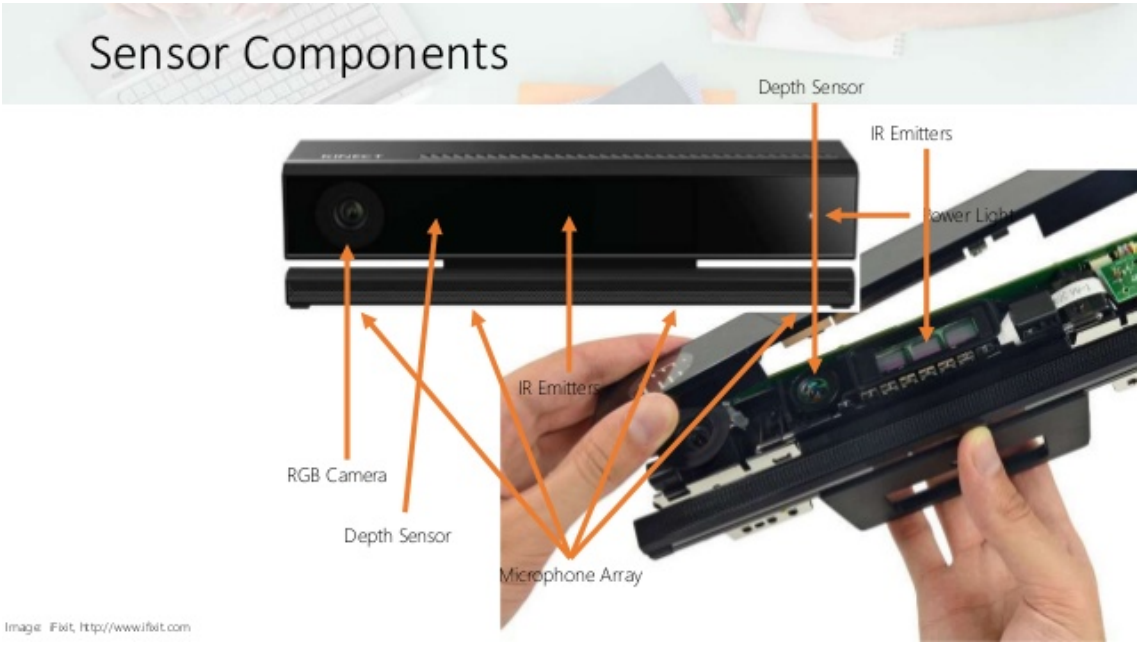

Figure 3.4: Kinect for Windows v2 components [46]

The drivers used in this project are part of `libfreenect`.[48]

3.4.2 Similar Devices

These devices are particularly interesting due to their market availability and the research interest and they have received, as the first affordable off-the-shelf Red, Green and Blue, and Depth (RGB-D) sensors.

There are other options, as this market diversifies, including:

- ASUS Xtion PRO LIVE [49] with similar specifications, compared to the Kinect for Xbox 360 and a smaller size and lower power consumption [50].
- The Structure depth sensor [51], developed for use with iOS devices, but compatible with other platforms. [52]

3.5 Qualisys Oqus Motion Capture System

Validation of results requires a trajectory ground truth. In this case that would be some tracking system with 6 Degrees of Freedom (DoF) capabilities, independently of the particular technology.

One possible system is that in use in the LABIOMEP - Porto Biomechanics Laboratory [53] that uses Qualisys Oqus [54] cameras for tracking in human biomechanics. These cameras track 4 mm or 19 mm spherical IR markers (for capturing distances of 9 m or 25 m respectively). Qualisys systems are often used in human tracking.[55] Six DoF tracking capability is achieved by connecting several cameras to a PC and using several markers per target link. Measurement precision is not available, but is estimated to be within 1 mm and 0.1°.

This system was used to produce the ground truth for the data sets discussed in Chapter 6. Multiple cameras are placed around and above the working area, as seen in Fig. 3.5.

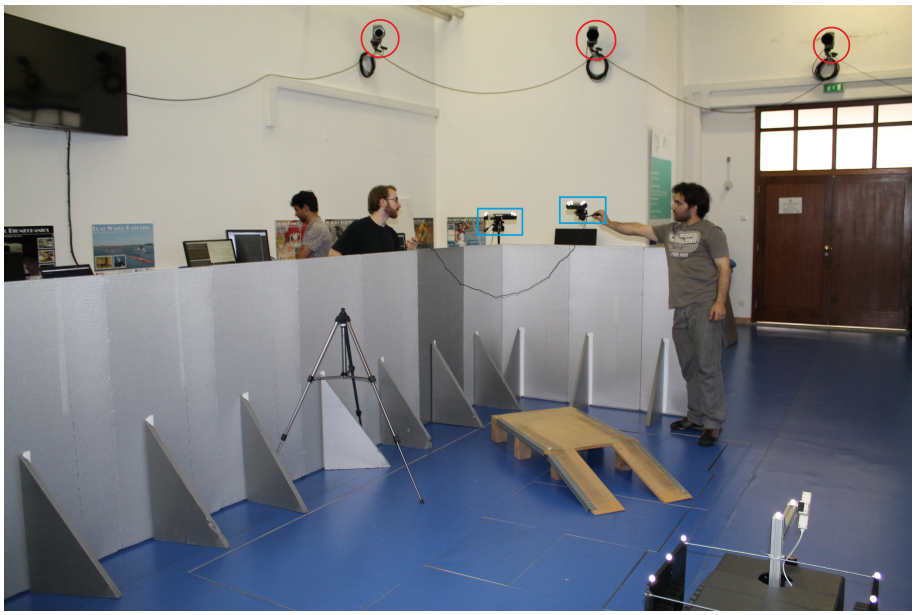

Figure 3.5: Qualisys Oqus Motion Capture System in use at LABOMEP (Qualisys Oqus cameras highlighted in red and Kinect for Xbox 360 devices in blue)

3.6 Conclusion

A set of relevant software and hardware has been identified, for both development and deployment use, including a flexible robotics middleware platform and RGB-D library and devices.

The system was developed with and will use ROS and PCL. The hardware used in development was the Microsoft Kinect for Xbox 360 sensor and Microsoft Kinect for Xbox One is recommended for deployment.

Chapter 4

Proposed Solution

4.1 Introduction

This tracking solution aims to be scalable, versatile and economical. To that end it is based on:

- Robot Operating System (ROS) — allows the end-user to connect several senors and computers;
- Microsoft Kinect for Windows — low-cost Red, Green and Blue, and Depth (RGB-D) sensors.

This system tracks objects based on their natural geometry without artificial markers.

It provides external localization functionality with one or more sensors.

It was designed around the Microsoft Kinect for Windows sensor, which publishes `pointcloud2` messages in ROS through the Freenect driver. It can use any sensor that can publish the same data, including time-of-flight laser depth sensors and other RGB-D or depth sensors.

The `pointcloud2` message includes, essentially, a point cloud comprised of the points visible to the sensor with their XYZ coordinates and Red, Green and Blue color (RGB) color. The RGB data is discarded, at this point in development, in favor of an XYZ point based approach.

The ROS package `six_dof_tracker` includes the following nodes:

- `tracking` — single-sensor tracking;
- `fusing` — data-fusion from several single-sensor tracking nodes;
- `localization` — tracking based self-localization for small rotations or translation of a sensor;
- `mapping` — stationary mapping for self-localization;
- `testing` — testing for the tracking node.

4.2 Prerequisites

Some necessary information and services are assumed to be available to the tracking system. At the same time, some decisions are left to the final user.

4.2.1 Sensor Localization

The system's sensors' poses must be known in reference to the global frame `world`, as the tracking target's pose is referenced to this frame. The method by which this information is made obtained is outside the scope of this system, but it should be made available to `tf` (see: Section 3.2.3).

A `mapping` and a `localization` node are provided for testing purposes and to allow for the correction of a sensor's pose when moved from a known location by a small rotation or translations (single digits in degrees of rotation or centimeters of translation).

The user can go as far as to manually publish a transformation between or can use one of many widely available ROS mapping and self-localization packages.

4.2.2 Communication

Communication between the sensors, tracking nodes and data-fusion node is handled transparently through ROS services and may include transitions between different computers. This will depend on a local network whose bandwidth can support the necessary messages. For reference, a Microsoft Kinect for Windows point cloud with resolution 640×480 and color for each point encoded in 8 floats takes up 9.3Mb and is captured at 30 Hertz, resulting in a bandwidth of 279MB/s. These considerations must be taken for other types of sensors.

4.2.3 Physical Setup

The advantages of having a multiple sensor tracking system are twofold: For one, occlusions effectively disappear when different sensors have their own views of the target, so that even if the target is occluded for one sensor it will be in full view for the global system. On the other hand, the effective tracking area can be transparently increased by adding new sensors.

It is up to the end-user to balance these advantages. Fig. 4.1 demonstrates this duality.

4.3 Tracking

The `tracking` node uses Fast Point Feature Histograms (FPFH) descriptors and Sample Consensus Initial Alignment (SAC-IA) registration to determine an initial pose and Iterative Closest Point (ICP) registration to track the target. Algorithm 1 summarizes this node's work flow.

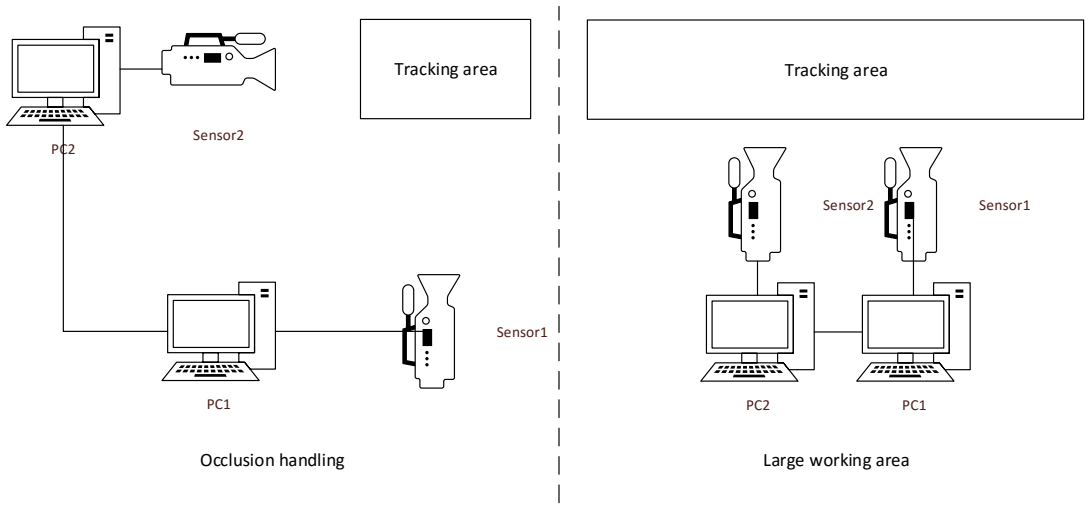

Figure 4.1: Proposed physical setup alternatives — Multiple views of target (left) and greater tracking area (right)

4.3.1 Point Cloud Preprocessing

Real world sensors are subject to noise to varying degrees and return large datasets that must be filtered to become manageable in a real-time system.

For each new point cloud, a pass-through filter removes points whose depth is beyond the sensor's effective working range. In the case of the Microsoft Kinect for Windows the working range is 0.4 m to 4 m.

A statistical out-lier removal filter removes points that are further from their neighbors than a set multiple of the standard deviation of pairwise distances.

A voxel grid filter collapses points within small voxels whose size is compatible with the senor's actual resolution.

Finally, the point cloud is transformed into the global frame so that further processing uses global frame coordinates.

Examples of this preprocessing routine are shown in Chapter 5. The example of Fig. 4.2 is of particular note, as it shows the result of filtering and under-sampling by voxel grid a noisy point cloud.

4.3.2 Initial Pose Estimation

Upon starting or having lost track of the target, the `tracking` node will initiate initial pose estimation. The objective of this process is to obtain an approximate transform for the target in the scene that will lie within convergence range for the continuous tracking algorithm.

This step uses the whole scene point cloud, as the target's position is completely unknown and no assumptions are made about it, other that it should be visible at this point.

This process relies on FPFH and SAC-IA, as described in Section 2.3.6.

Algorithm 1 Single-sensor tracker

Input: M (target model point cloud), C' (captured point cloud), T_{cw} (camera to world transform), t (maximum error threshold)
Output: T (new target pose transform), e (registration error) C_c (cropped point cloud)

function PROCESS(M, C', T_{cw})
  $C \leftarrow$ PREPROCESS(C', T_{cw}) ▷ Algorithm 2
  if *trackerinitialized* = *false* **then**
    $T, e \leftarrow$ FIND(M, C) ▷ Algorithm 3 — get initial estimate
  else
    $T, e \leftarrow$ TRACK(M, C, T') ▷ Algorithm 4 — continuous tracking
  end if
  if $e > t$ **then**
    trackerinitialized $\leftarrow$ *false* ▷ Lost target and should reinitialize
    return
  else
    $C_c \leftarrow$ CROP(M, C, T) ▷ Algorithm 5
  end if
  return T, e, C_c
end function

FPFH descriptors depend on surface normals, so these are estimated for the whole scene and for the target model (target model normals are estimated on node startup upon loading the respective point cloud).

The target model is aligned to the scene using SAC-IA which returns the approximate initial pose and the alignment error. If the alignment error is greater than a threshold, the target has not been found and the tracker is reinitialized.

4.3.3 Continuous Tracking

The tracker assumes proximity as a constraint on motion. This means the displacement from frame $i-1$ to frame i is assumed to be smaller than a predetermined distance: $|x_i - x_{i-1}| < d$.

As long as this assumption holds, using the latest pose as an initial estimate for ICP registration will yield good results in few iterations.

Algorithm 2 Preprocess point cloud

Input: C' (unprocessed point cloud), T_{cw} (camera to world transform)
Output: C (preprocessed point cloud)

function PREPROCESS(C)
  $C_{temp} \leftarrow$ PASSTHROUGHFILTER(C', min, max, z) ▷ Applied over the z axis
  $C_{temp} \leftarrow$ STATISTICALOUTLIERFILTER(C_{temp}, k, a) ▷ Number of neighbors and multiple of std
  $C_{temp} \leftarrow$ VOXELGRIDFILTER($C_{temp}, leafsize$)
  $C \leftarrow$ TRANSFORM(C_{temp}, T_{cq}) ▷ Transforms to global frame
  return C'
end function

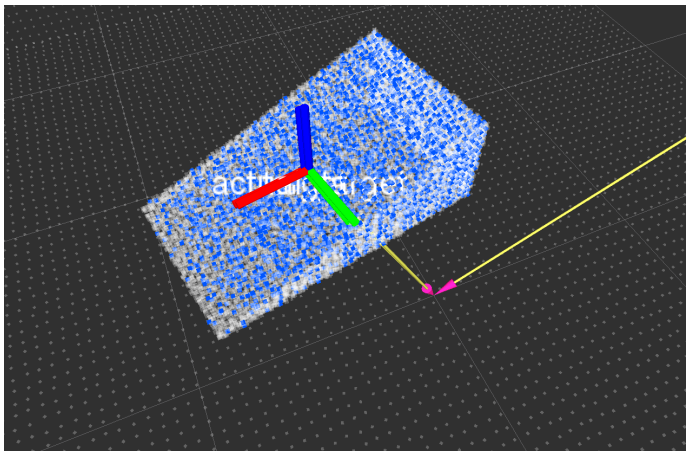

Figure 4.2: Captured point cloud (white) and preprocessed point cloud (blue)

This assumption has a useful corollary: points that are very far from the latest pose will not be part of the target. With this consideration in mind, each new point cloud is cropped around the target's location. The crop box is translated and rotated to match the latest pose and is enlarged, compared to the target model, to account for the object's movement between frames. This Region of Interest (ROI) is demonstrated in Fig. 4.3.

This cropped point cloud is used in ICP registration, instead of the full scene, to improve performance, by reducing the number of points in the cloud. It is also published for use by a data fusion node.

At any point during tracking, if the registration score is worse than a threshold, the target is considered lost and a new initialization is required. This can be avoided with the use of multiple sensors, whereupon an individual tracker will be able to use the global tracking result for the next iteration.

Algorithm 3 Estimate initial pose

Input: M (target model point cloud), C (captured point cloud)
Output: T (target pose transform), e (registration error)

1: **function** FIND(M, C)
2: $C \leftarrow$ PREPROCESS(C) ▷ Algorithm 2
3: $N_T \leftarrow$ COMPUTENORMALS(T)
4: $N_C \leftarrow$ COMPUTENORMALS(C)
5: $F_T \leftarrow$ COMPUTEFPFH(T, N_T)
6: $F_C \leftarrow$ COMPUTEFPFH(C, N_C)
7: $T, e \leftarrow$ SAC-IA(T, C, F_T, F_C)
8: **return** T, e
9: **end function**

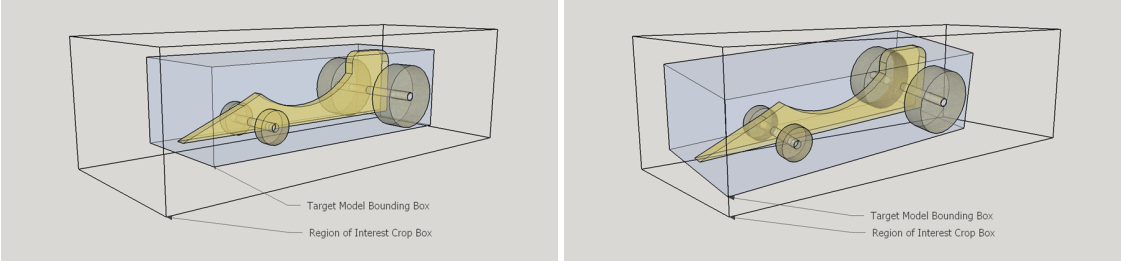

Figure 4.3: ROI from frame $t-1$ (left) and target within ROI on frame t (right)

4.4 Data Fusion

The work flow of the `fusing` data fusion node is described in Algorithm 6. This node takes the local results from single-sensor tracking nodes, merges their partial views and returns a higher quality tracking result.

4.4.1 Data Flow

Taking advantage of the publisher/subscriber paradigm offered by ROS, any number of single-sensor local trackers can broadcast their pose estimates and have access to the latest global estimate.

The obvious advantage of data fusion is having a higher quality global result, as the fusion tracker has access to more information, be that different views of the target or a wider operating range. The less obvious, but equally important advantage is that local trackers can use the latest global pose as an estimate for the next iteration.

Not only does this allow for faster loss recovery, as a local node will not have to reinitialize, but will instead use global information, it also means non-deterministic tracking methods can be used, without burdening individual single-sensor trackers. The global tracker can, for example, use an Extended Kalman Filter (EKF) and provide the filter's prediction as an estimate.

This multi-directional information flow is displayed by the sequence diagram in Fig. 4.4. Note that all messages are published, but only subscribed messages are represented, for simplicity and readability.

The first node, `Tracker 1`, estimates the target's initial pose and publishes it, along with the registration error and cropped point cloud. The global tracker subscribes to these and performs its

Algorithm 4 Track target

Input: M (target model point cloud), C (captured point cloud), T' (latest target pose transform)
Output: T (new target pose transform), e (registration error)

1: **function** TRACK(M, C, T)
2: $C_c \leftarrow \text{CROP}(M, C, T)$ ▷ Algorithm 5
3: $T, e \leftarrow \text{ICP}(M, C_c, T')$ ▷ Uses last pose as initial estimate
4: **return** T, e
5: **end function**

Algorithm 5 Crop point cloud

Input: M (target model point cloud), C (captured point cloud), T (target pose transform)
Output: C_c (cropped point cloud)

function CROP(M, C, T)
    $minpoint, maxpoint \leftarrow$ MINMAX3D(M) ▷ Get bounding box corners
    $C_c \leftarrow$ CROPBOXFILTER($C, minpoint + d, maxpoint + d, T$) ▷ Crop around target with orientation leaving extra space
    return C_c
end function

own tracking iteration, after which it publishes a global pose and error, as well the merged point cloud, to which other ROS nodes would subscribe.

The single-sensor tracker also subscribes to the global pose and error, but these will only truly come into play once `Tracker 2` is started. Instead of initializing by estimating an initial pose, it will subscribe to the global data and work from there. From this time, the tracking system continuously shares its results, so that the global results are refined by having access to extra views and quantitative data and the local trackers don't lose the target and perform better, based on a quality previous pose.

4.4.2 Off-line Fusion

As mentioned in Section 3.2.4, ROS offers data logging tools that can, upon playback, emulate nodes' communication over their published topics.

When operating independently, the output from single-sensor trackers can be recorded and later played back for off-line data-fusion.

4.5 Output

The tracking problem's output can be expressed in the form of a homogeneous transformation matrix that includes information on rotation and translation:

Algorithm 6 Data fusion tracker

Input: M (target model point cloud), C_c (cropped point clouds), t (maximum error threshold), T' (target pose transforms), e' (registration errors)
Output: T (new target pose transform), e (registration error) C_m (merged point cloud)

function PROCESS(M, C_c, T')
    for $n \leftarrow 1, n_{sensors}$ **do**
        $C_m \leftarrow C_m + C_c[n]$ ▷ Merge point clouds
    end for
    $T_{best} \leftarrow T'\left[e'_{best}\right]$
    $T, e \leftarrow$ TRACK($M, C_m, T'\left[e'_{best}\right]$) ▷ Algorithm 4 — continuous tracking
    return T, e, C_m
end function

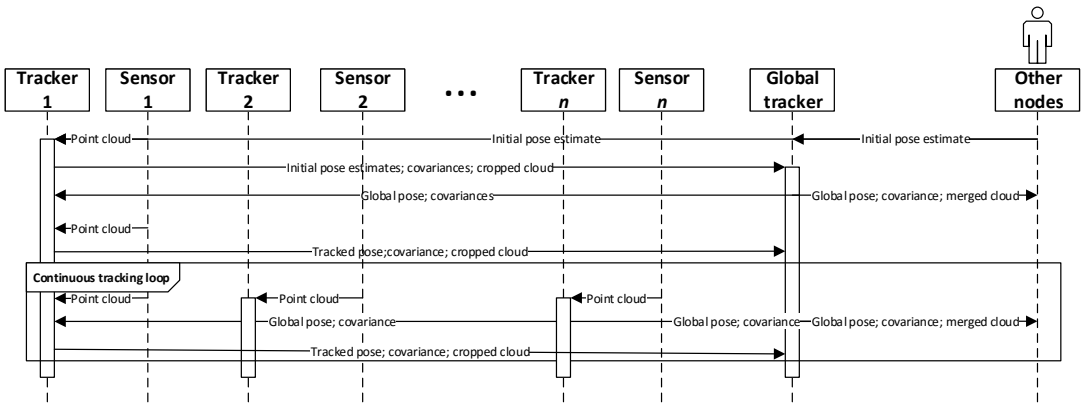

Figure 4.4: Data fusion sequence diagram

Translation vector:

$$d = \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{4.1}$$

Rotation matrix:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.2}$$

Combined into the homogeneous transformation matrix:

$$T = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} \tag{4.3}$$

This is the output of Point Cloud Library (PCL) registration methods.

Rotation can be represented by matrix R and by quaternion q:

$$q = \left\langle \begin{array}{cccc} x & y & z & w \end{array} \right\rangle \tag{4.4}$$

ROS provides transparent conversions between R and q.

This information is transmitted in ROS encapsulated into a `geometry_msgs/TransformStamped` message which includes:

- A time stamp;
- The name of the parent frame;
- The name of the child frame;

Table 4.1: Proposed tracker cost estimate

Component	Cost (€)
General purpose computer	650
Microsoft Kinect for Xbox One	200
Network switch	20
Total (2 sensors; 2 computers)	1720
Total (3 sensors; 3 computers)	2570

- The translation vector;
- The rotation quaternion.

This is particularly useful, because it includes the tracked object in the system's `tf2` transform tree (see Section 3.2.3).

Along with this information, a confidence rating is also useful. Registration algorithms also return this in the form of the internal error metric for the last iteration. This is transmitted along with the target's frame name as a `float64`. This should be further refined into a covariance matrix.

4.6 Cost Analysis

The proposed tracker is financially competitive, as it uses off the shelf hardware. Commercial RGB-D sensor solutions, such as the proposed Kinect for Xbox 360, are more and more cost effective and have adequate characteristics. Other hardware requirements are limited to general purpose computers and basic networking hardware.

A Microsoft Kinect for Xbox One with a PC adapter has a total estimated retail price of €200, in Portugal (or 200 USD, in the United States) [44].

The hardware specifications for this device are: [56]

- "64-bit (x64) processor"
- "Physical dual-core 3.1 GHz (2 logical cores per physical) or faster processor"
- "USB 3.0 controller dedicated to the Kinect for Windows v2 sensor or the Kinect Adapter for Windows for use with the Kinect for Xbox One sensor"
- "4 GB of RAM"
- "Graphics card that supports DirectX 11"

A market survey of 2015 prices leads to an estimate of €650 for a desktop computer with these characteristics. A networking switch can cost as little as €20.

Cost estimates for different configurations are listed in Table 4.1.

Table 4.2: Summary of tracker design considerations

Design consideration	Decision
Object representation	Shape-based: point cloud
Feature selection	FPFH; point cloud
Object detection	FPFH, SAC-IA
Joint detection and tracking	Yes
Tracking method	Deterministic; ICP
Tracking assumptions	Maximum velocity
Loss recovery; occlusion handling	Reinitialization; data fusion

For comparison, the tracking system used in data collection, discussed in Chapter 6 has an approximate cost of 150 000€. This system offers better precision than the proposed solution, but its price is two orders of magnitude greater than what would be expected for the proposed system and it depends on Infrared (IR) markers.

4.7 Conclusion

Revisiting the tracker design decisions outlined in Section 2.2, the proposed tracker is summarized by Table 4.2.

Chapter 5

Development and Evaluation

5.1 Introduction

This chapter chronologically details development efforts in various areas, examining avenues of development that were eventually not followed and providing greater insight into design decisions.

Development logs are available on the project's website (`http://paginas.fe.up.pt/~ee10192/`).

5.2 Familiarization with Kinect, PCL and ROS

The first weeks of development brought a greater practical understanding of Robot Operating System (ROS), Point Cloud Library (PCL) and the Microsoft Kinect for Windows sensor. There is a great wealth of information and resources for each ranging from software and hardware documentation [34, 41, 57] to research papers [37, 38, 39, 40] and hobbyist tutorials.

5.3 Mapping and Self-Localization

As stated in Section 4.2, tracking a target in the global frame requires sensors' locations. These, in turn, can be acquired through map-based self-localization.

Initial mapping efforts consisted of simply taking one point cloud, filtering it (as described in Section 4.3.1) and saving it as a map. Self-localization was performed with Iterative Closest Point (ICP) registration.

This approach is computationally inexpensive and produces good results for small translations and rotations around the initial pose (few centimeters and degrees).

It should be noted that large translations and rotations (on the order of 0.5 m and 45°) result in ICP registration falling into local minima and unreasonable output poses.

In an effort to improve the performance of the self-localization node, ICP was replaced by Generalized ICP [58]. This extension to the original algorithm combines ICP with point-to-point variants into a probabilistic framework and promises more robustness. In testing, this proved

decidedly more robust, but much more resource intensive. The decision was made to use standard ICP and limit the range of the `localization` node.

5.4 Target Detection With Background Subtraction

Initially, target detection was to be focused on background subtraction. This would build upon mapping and self-localization.

Upon acquiring and preprocessing a point cloud, it would de added to an octree, along with the map, and points not present in the map (outliers) would be identified as in [59].

Intermediate results for this process are show in Fig. 5.1. The box in the center, on the floor, was correctly segmented, but many other points were incorrectly identified as outliers.

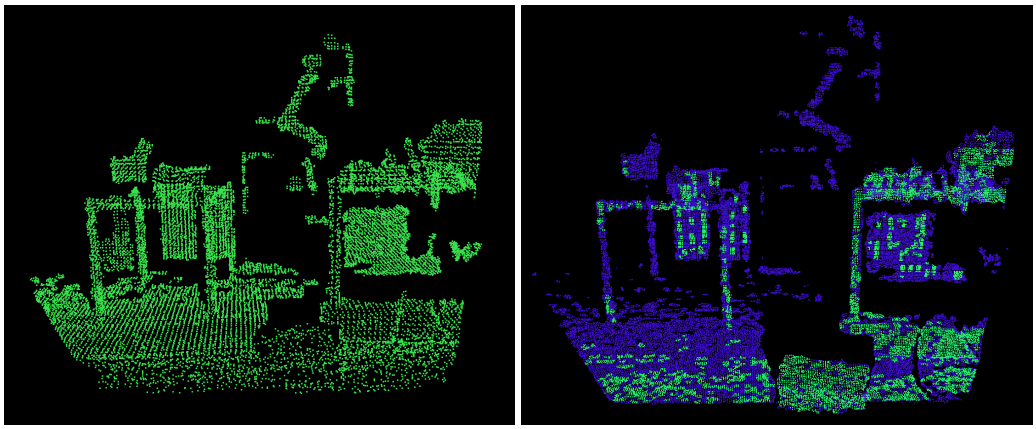

Figure 5.1: Map (left) and captured point cloud and outliers (right) in background subtraction

Foreground points were segmented into different objects through Euclidean clustering as described in [60].

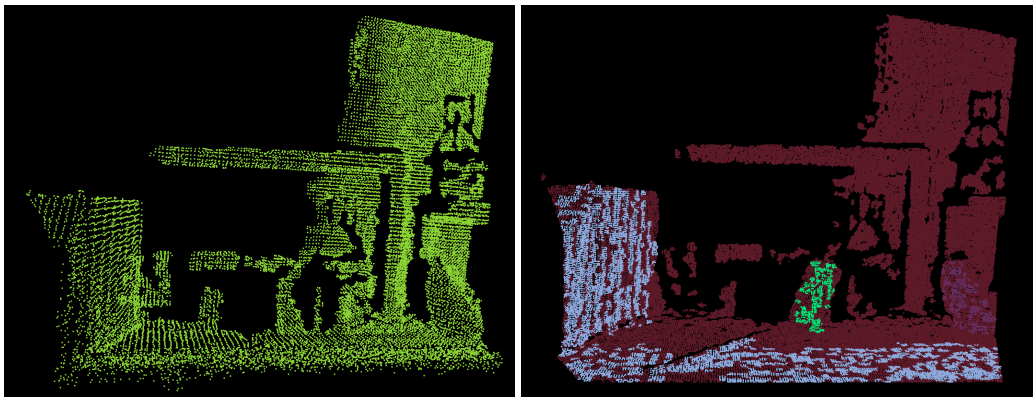

Figure 5.2: Map (left) and captured point cloud and out-lier clusters (right) in background subtraction

The box in green in Fig. 5.2 was correctly segmented, but, once again, large elements that are clearly part of the map were also clustered as target candidates.

5.5 Initial Pose Estimate With ICP

The initial approach to initial pose estimate took the previously obtained out-lier clusters and registered each one to the target model with ICP and select the cluster with the best registration score as the target and use its transformation matrix as the initial pose estimate.

This algorithm carries several problems. First, as already discussed, ICP has a tendency to fall into local minima, resulting in incorrect poses - in fact, it is not at all suited for initial estimates, but can specifically depend upon a quality estimate. Second, an arbitrary cluster could match the target better than the correct one, due to the less than ideal segmentation results.

These issues lead to abandoning background subtraction and ICP as an initial estimate option.

5.6 Using Multiple Sensors

As suggested in Chapter 4, this tracker has the option of using more than one sensor for the same area.

Capturing and merging data from two Kinect sensors was successful in [61], specifically demonstrated in [62]. Proper merging of point clouds requires the relative pose of the sensors [62], which is available if the prerequisites discussed in Section 4.2 are met.

Testing in the laboratory brought satisfactory results, as shown in Fig. 5.3. The inconsistencies in color are a result of using software instead of hardware color registration, as color data is not essential to the tracker, at this point, and is only displayed for human readability.

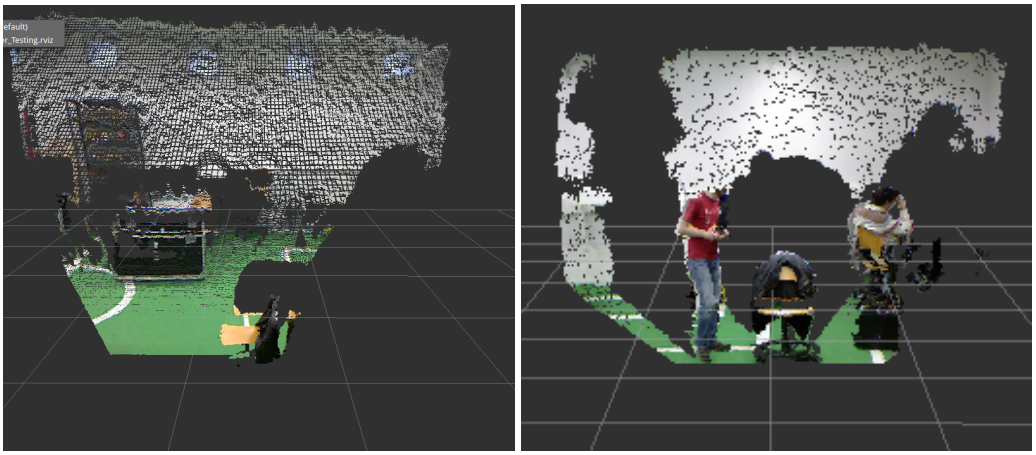

Figure 5.3: Point clouds from two Kinect sensors simultaneously pointing at chair

5.7 Tracking

Testing of continuous tracking assumed a known initial pose and used the `testing` node. Fig. 5.4 shows the simulated target point cloud along with the visualizations of the `target` (tracking result) and `actual_target` (simulated) transforms.

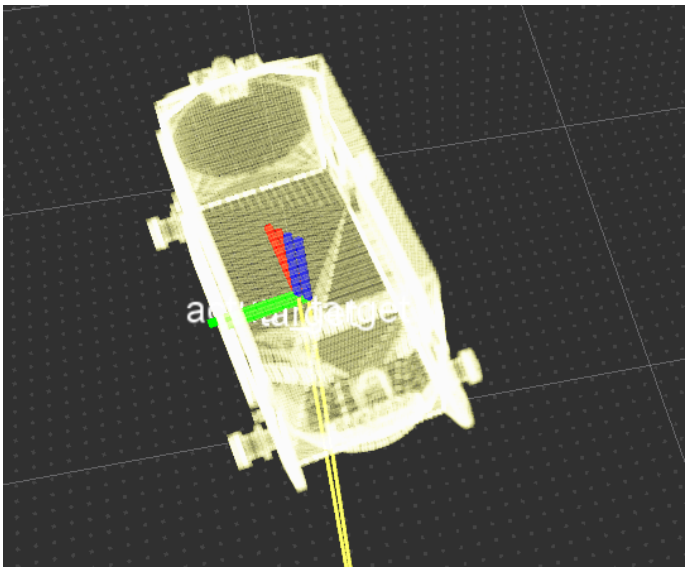

Figure 5.4: Tracking test — point cloud and transforms

While performing a translation at $0.01\,\mathrm{m\,s^{-1}}$ along the y axis, the tracking error was within the objectives defined in Section 1.5, as demonstrated by the graphics in Fig. 5.5. The target was stationary until $t = 20$ and then started its motion.

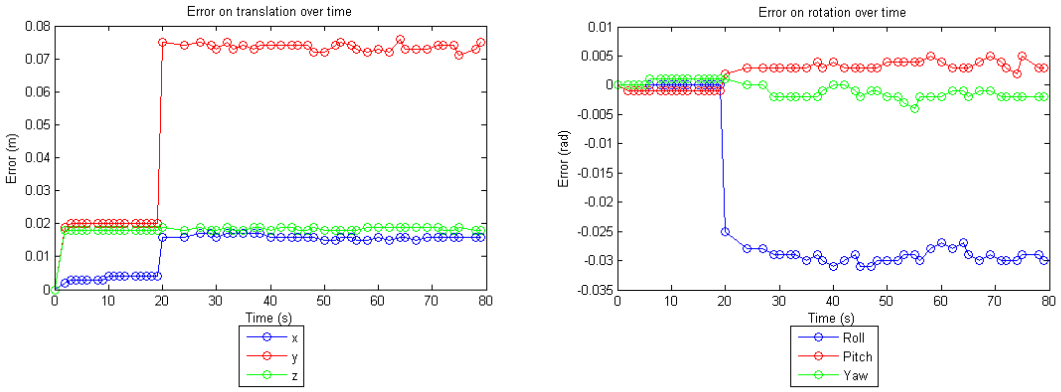

Figure 5.5: Tracking error during translation

While performing a rotation at $1.5 \times 10^{-2}\,\mathrm{rad\,s^{-1}}$ over the z axis, the tracking error was within the objectives defined in Section 1.5, as demonstrated by the graphics in Fig. 5.6.

Combining both translation and rotation, the target is quickly lost as demonstrated by the graphics in Fig. 5.7, showing increasing error in the y axis and in yaw — the tested variables.

The target is lost, because the large difference in transformation between one frame and the next requires more ICP iterations, slowing down the process to the point where the assumption of proximity between tracking iterations no longer holds. Higher performing hardware and data fusion would mitigate this issue.

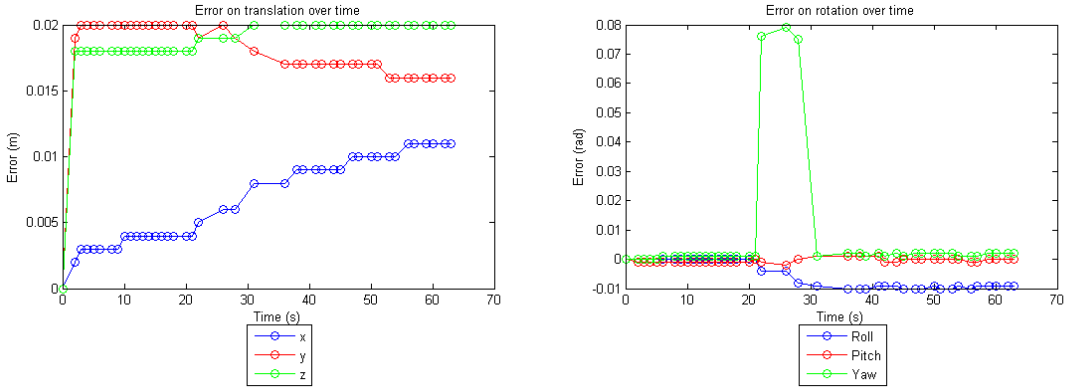

Figure 5.6: Tracking error during rotation

5.8 Tracking Simulations

5.8.1 Target

The target is a prism whose base is a right triangle that is not isosceles. The target mesh and point clouds are represented in Fig. 5.8. This solid was selected, because it is geometrically simple and its orientation is unambiguous to an observer.

The target's dimensions are $0.75 \times 0.45 \times 0.25$ m.

5.8.2 Simulation Environment

These simulations were conducted in a Xubuntu 14.04 system with a 2016 Intel Pentium 4 processor and 3 GB of RAM.

This configuration does not take advantage of a dedicated Graphics Processing Unit (GPU). Using a GPU would improve performance, as PCL offers GPU optimized algorithms, particularly for NVIDIA's CUDA Application Programming Interface (API). [63]

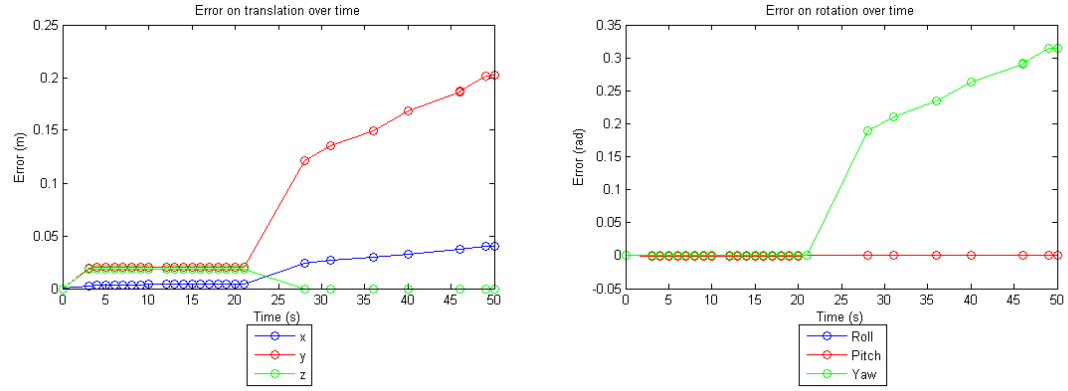

Figure 5.7: Tracking error during translation and rotation

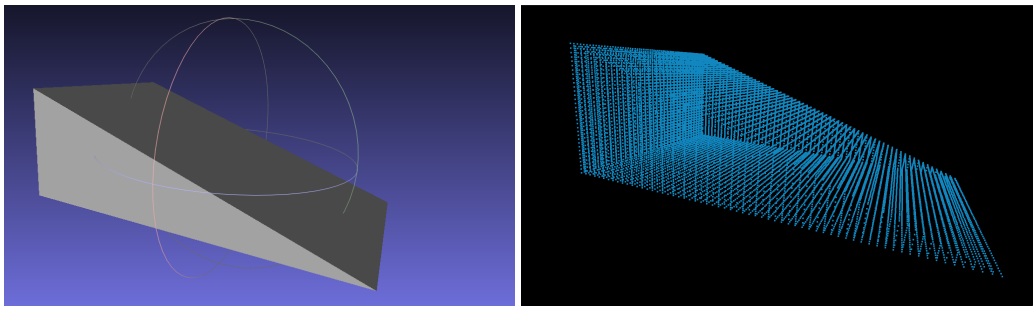

Figure 5.8: Simulation target mesh (left) and point cloud (right)

5.8.3 Simulation Procedure

Using the `testing` node, the target point cloud was published as a `pointcloud2` message referenced to `/camera_depth_optical_frame`, as would be expected from a Kinect sensor. As suggested in Section 4.2.1, a transformation was manually published from the `world` frame to the simulated camera frame, thereby providing the pose of the sensor. The target's original pose was provided to the initial pose estimation routine (bypassing Algorithm 3).

The error in pose is the difference between calculated and actual pose. This is obtained by querying `tf2` for the transform from `target` to `actual_target`.

For the following tests, the target was stationary, translated along the y, rotated in yaw (around the z axis) and finally simultaneously translated and rotated.

These tests were replicated in 4 sets of increasingly more challenging circumstances:

1. No noise and no occlusions (Section 5.9);
2. Random noise and no occlusions (Section 5.10);
3. Random noise and 25% occlusion (Section 5.11);
4. Random noise and 75% occlusion (Section 5.12).

5.9 Simulation 1 — No Noise and No Occlusion

5.9.1 Stationary Target

In this simulation the target remained stationary in the initial pose that was available to the tracker. All points of the published cropped cloud match the simulated target clod, as shown in Fig. 5.9.

The intuitive expectation would be that the error would be null, as the system starts with perfect knowledge and is not disturbed, however, as shown in Fig. 5.10, this is not the case.

Due to the preprocessing described in Section 4.3.1, the tracked cloud is not the same as the captured cloud: it is filtered and under-sampled to improve performance and increase robustness to sensor noise. As a result of this, the pose error quickly stabilizes in a small value, within the defined limits for the system.

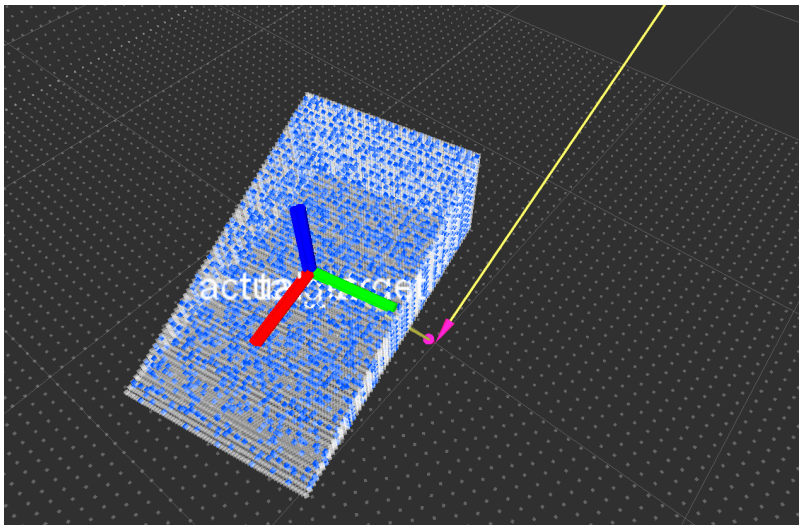

Figure 5.9: Simulation 1 — Simulated target (white) and cropped point cloud (blue) while stationary

5.9.2 Translation

For this simulation, the target moved along the y axis at a velocity of $0.01\,\mathrm{m\,s^{-1}}$.

By introducing motion two new sources of errors arise: lag (the target's position is not known at time t, but only at $t + t_{update}$) and the possibility of the assumptions made in tracking not holding. As introduced in Section 4.3.3, it is assumed that the displacement of the target between frames is small. If the pose update is too long, this will no longer old.

As lag comes into play, the published cropped point cloud does not match the target's current position, as displayed in Fig. 5.11.

In this case, the errors stabilize at higher values, but continuous tracking is maintained and results are satisfactory.

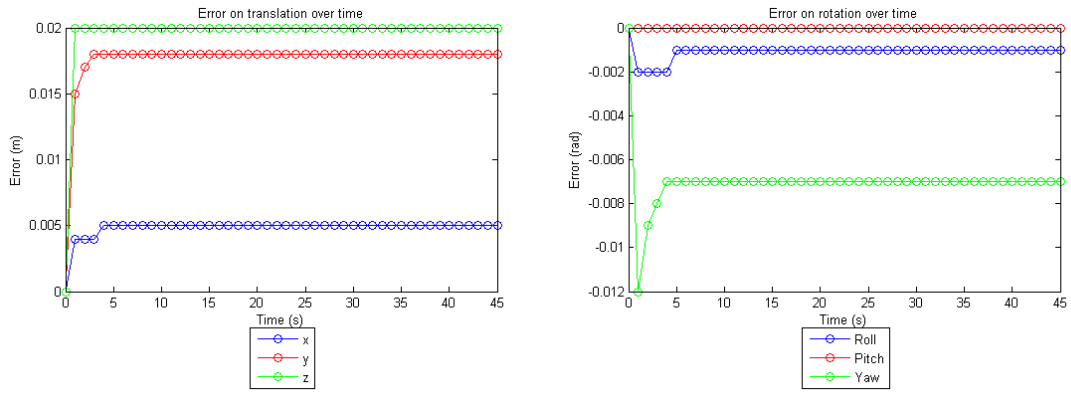

Figure 5.10: Simulation 1 — Tracking error with stationary target

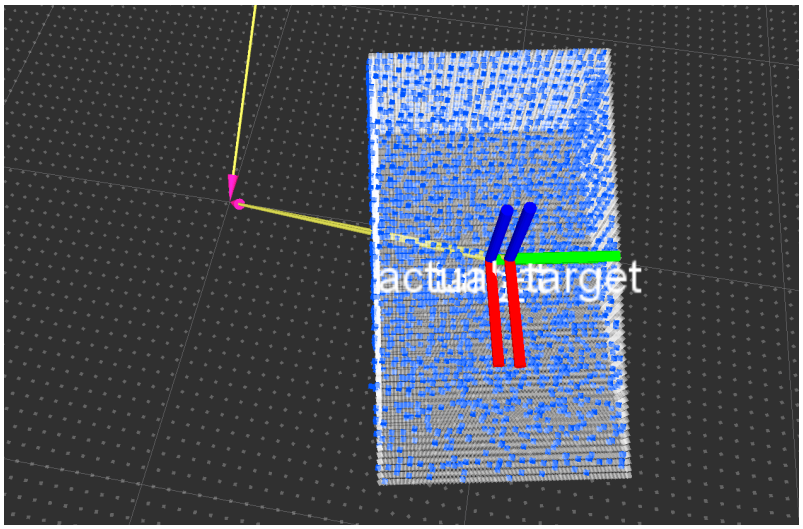

Figure 5.11: Simulation 1 — Simulated target (white) and cropped point cloud (blue) during translation

5.9.3 Rotation

For this simulation, the target rotates around the z axis (yaw) at an angular velocity of $1.5 \times 10^{-2}\,\mathrm{rad\,s^{-1}}$.

This scenario focuses more directly on the system's interest in 6 Degrees of Freedom (DoF) pose.A simulation visualization is shown on Fig. 5.13, where the published cropped cloud almost completely matches the simulated target.

As shown in the graphs of Fig. 5.14, error on yaw is well bounded, but have a greater variance than pitch and roll. Translation error on the z axis is small and constant, but on x and y it is sinusoidal.

Causes of this behavior include numerical errors, incorrect interpolations in the `tf2` query and point cloud preprocessing (see Section 4.3.1). The voxel grid leaf size is $2 \times 10^{-2}\,\mathrm{m}$.

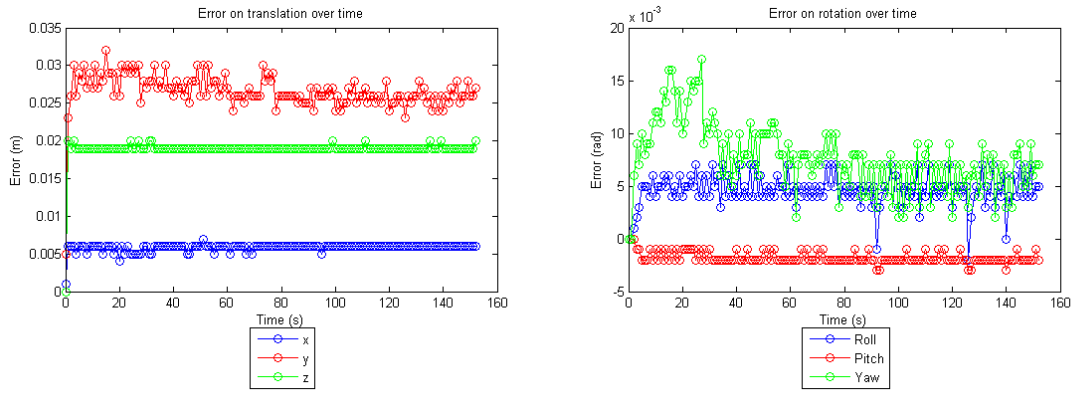

Figure 5.12: Simulation 1 — Tracking error during translation

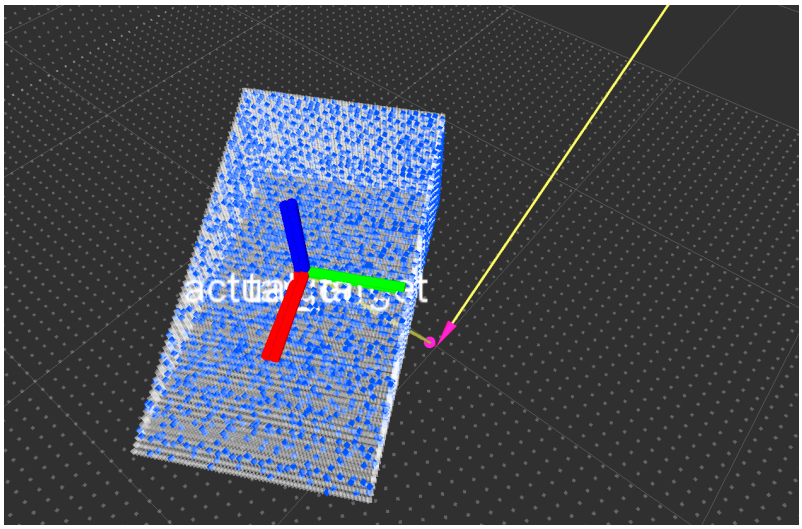

Figure 5.13: Simulation 1 — Simulated target (white) and cropped point cloud (blue) during rotation

5.9.4 Rotation With Smaller Voxel Grid

Using a voxel grid leaf size of 10^{-2} m as shown in Fig. 5.15 with error in the graphs of Fig. 5.16 and in Table 5.4.

This shows that the error in orientation is reduced, but the error in translation shows only a slight reduction. This leads to the conclusion that the major causes of this error are numerical errors and lag between `tf2` data.

5.9.5 Translation and Rotation

For this simulation, the target moves along the y axis at a velocity of $0.01\,\mathrm{m\,s^{-1}}$ and rotates around the z axis (yaw) at an angular velocity of $1.5 \times 10^{-2}\,\mathrm{rad\,s^{-1}}$.

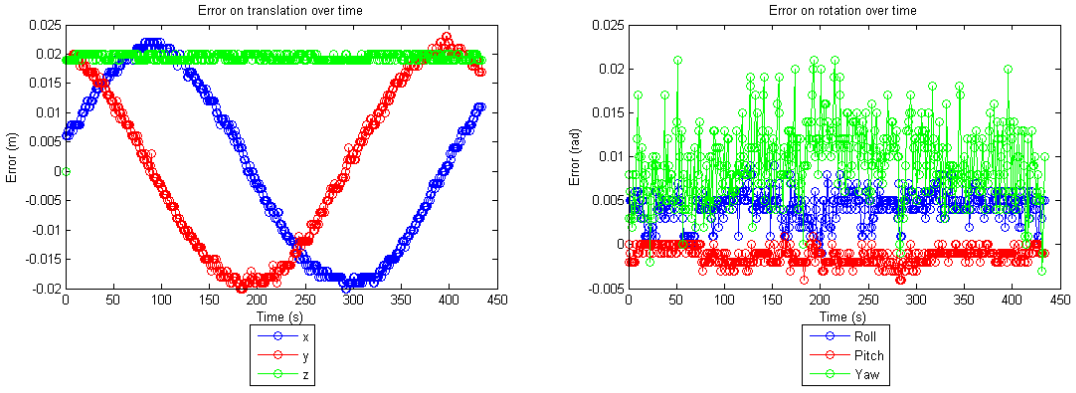

Figure 5.14: Simulation 1 — Tracking error during rotation

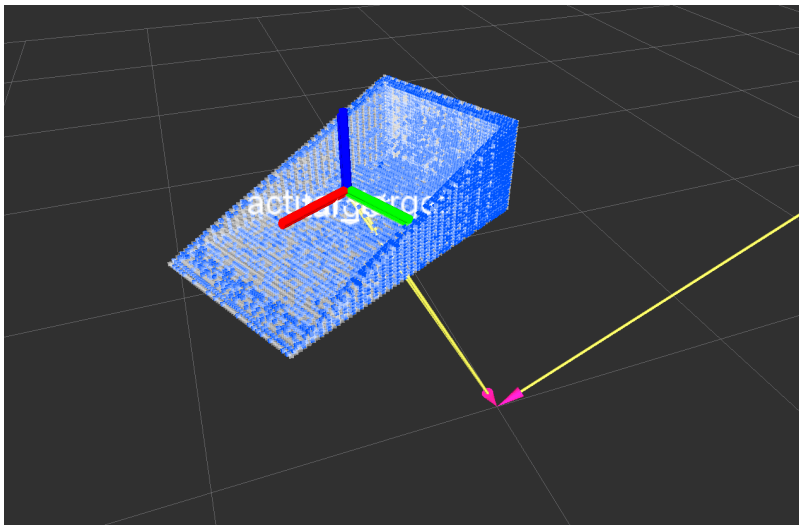

Figure 5.15: Simulation 1 — Simulated target (white) and cropped point cloud (blue) during rotation

In this case the difference in pose from one frame to the next is greater and requires more ICP iterations. This lower the update rate, which can eventually result in a lost target.

While the pose in a frame is computed, the object moves away from the previous position, causing a greater computational cost for the next frame and further lowering the update rate. As the lag increases, the target moves out of the neighborhood of its last know position, leaving the cropped point cloud, which results in unsuccessful registration.

This process of failure is demonstrated in Fig. 5.17, where, on the left, a significant error in localization is visible and, on the right, the target has been lost and is leaving the area of the cropped point cloud.

As is evident in the graphs of Fig. 5.18, the target was lost. In fact, at $t = 17$, the tracker is reinitialized with the initial pose, due to a very high registration error, and there is a sharp increase in error, as the object is no longer being tracked.

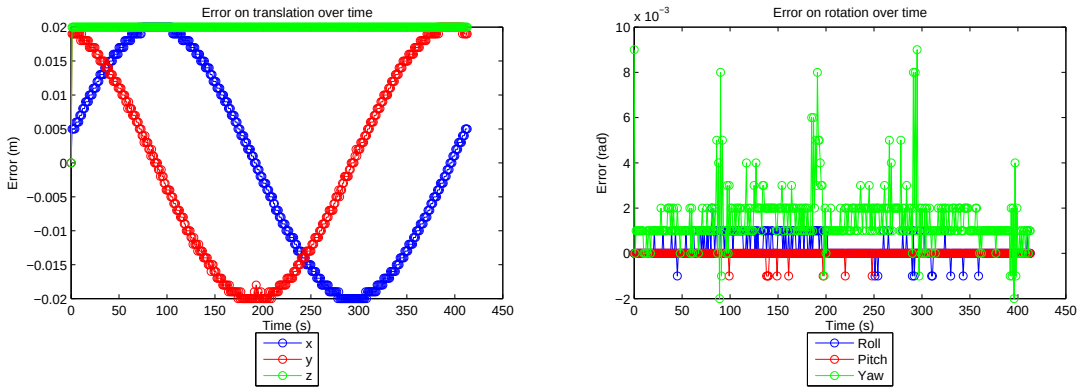

Figure 5.16: Simulation 1 — Tracking error during rotation (with smaller voxel grid)

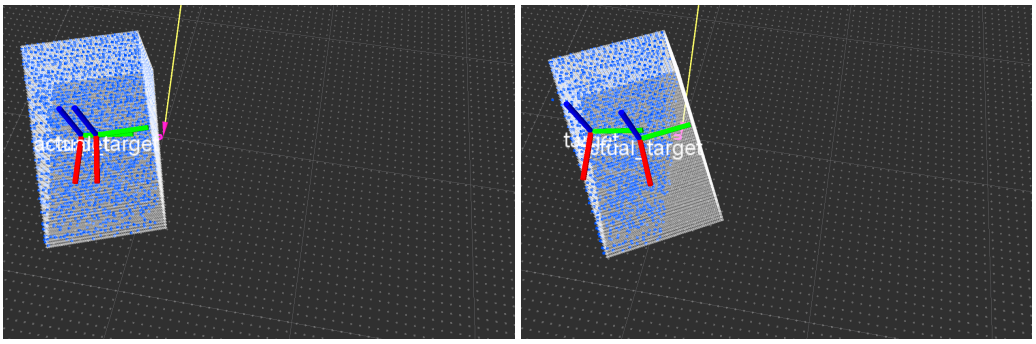

Figure 5.17: Simulation 1 — Simulated target (white) and cropped point cloud (blue) during translation and rotation

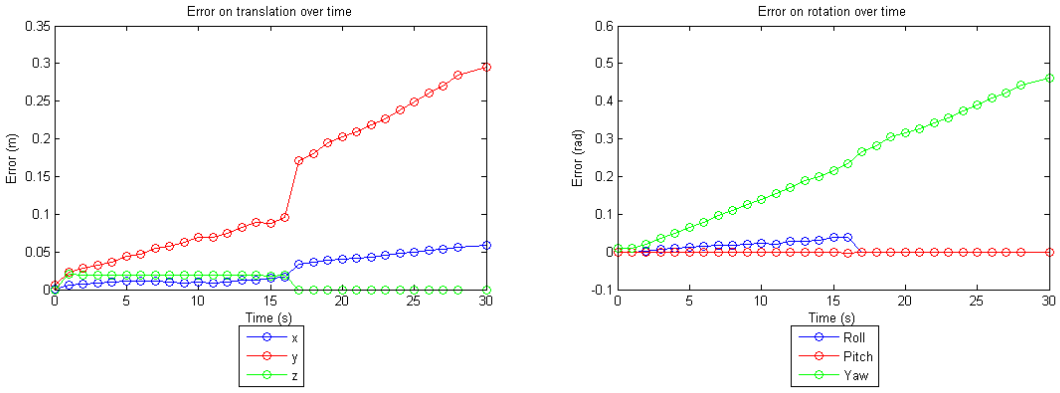

Figure 5.18: Simulation 1 — Tracking error during translation and rotation

Table 5.1: Simulation 1 — Errors while stationary

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0050	0.0180	0.0200	0.0087	0.0020	0	0.0120
RMS	0.0049	0.0177	0.0198	0.0085	0.0011	0	0.0071

Table 5.2: Simulation 1 — Errors during translation

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0070	0.0320	0.0200	0.0121	0.0070	0.0030	0.0170
RMS	0.0059	0.0267	0.0190	0.0102	0.0048	0.0018	0.0083

Table 5.3: Simulation 1 — Errors during rotation

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0220	0.0230	0.0200	0.0381	0.0090	0.0040	0.0210
RMS	0.0140	0.0144	0.0194	0.0243	0.0046	0.0016	0.0103

Table 5.4: Simulation 1 — Errors during rotation

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0200	0.0200	0.0200	0.0346	0.0020	1.00×10^{-3}	0.0090
RMS	0.0140	0.0143	0.0200	0.0243	4.9×10^{-4}	1.4×10^{-4}	0.0019

Table 5.5: Simulation 1 — Errors during translation and rotation

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0150	0.0890	0.0200	0.0260	0.0390	0.0020	0.2160
RMS	0.0101	0.0101	0.0184	0.0175	0.0202	0.0013	0.1241

5.10 Simulation 2 — Noise and No Occlusion

Identical simulations were carried out, after introducing random noise in simulated points. Each point was affected by random noise in its x, y and z coordinates, up to 1×10^{-2} m. The aim of these simulations is to evaluate the tracker's robustness to noise levels that would be expected of current Red, Green and Blue, and Depth (RGB-D) sensors.

5.10.1 Stationary Target

Tracking performance with a stationary target is comparable to the previous case. The visual effect of noise is shown in Fig. 5.19 and error results are shown in the graphs of Fig. 5.20 and in

Table 5.6.

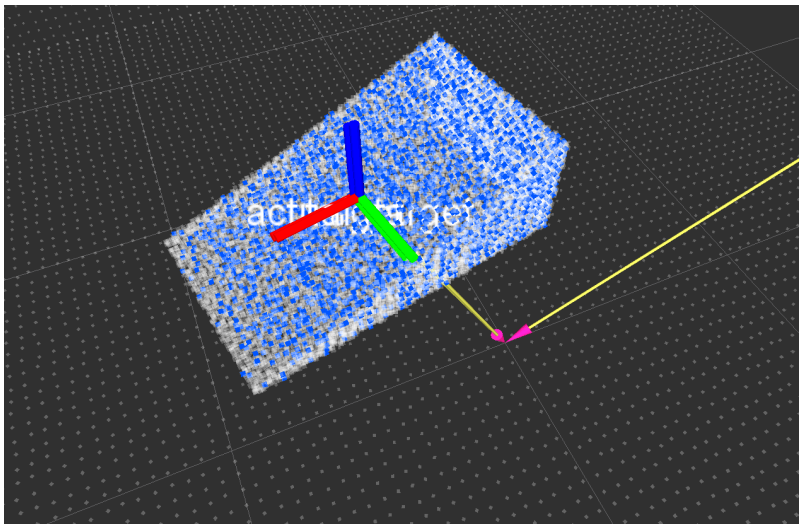

Figure 5.19: Simulation 2 — Simulated target with noise (white) and cropped point cloud (blue) while stationary

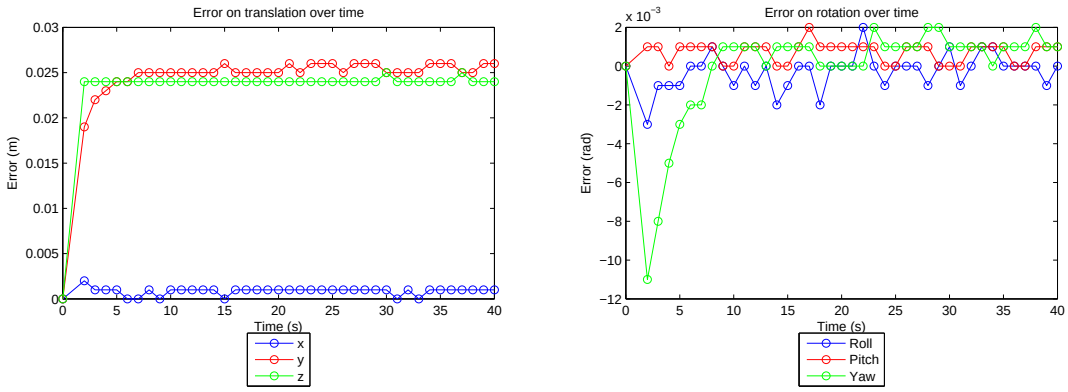

Figure 5.20: Simulation 2 — Tracking error with stationary target

5.10.2 Translation

During translation the tracker lags further behind the target, as is notable in Fig. 5.21. Error values increase, approaching the set limits, as shown in Fig. 5.22 and Table 5.7.

5.10.3 Rotation

Once again, performance in rotation is comparable to the original simulation without incorporating noise. The results are visualized in Fig. 5.23. Error values are well within defined limits as shown in the graphs of Fig. 5.24 and in Table 5.8.

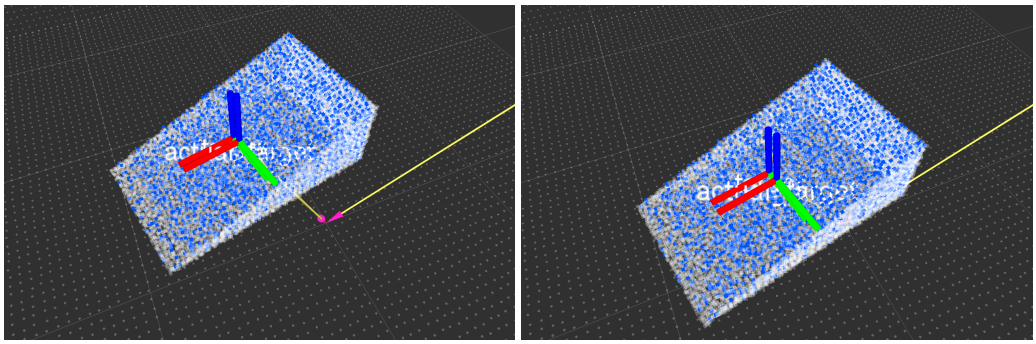

Figure 5.21: Simulation 2 — Simulated target with noise (white) and cropped point cloud (blue) during translation

5.10.4 Translation and Rotation

As was the case without noise, the target is quickly lost, this time after 14 s. the phenomenon of the target leaving the previous frame's neighborhood is clearly visible in Fig. 5.25.

Error values in the graphs of Fig. 5.26 and Table 5.9 are larger than in the previous simulation and show that the target was lost 3 s sooner. This is a result of compounding the increased tracking error as seen in the translation with noise simulation with the adverse results of lag in this dynamic situation.

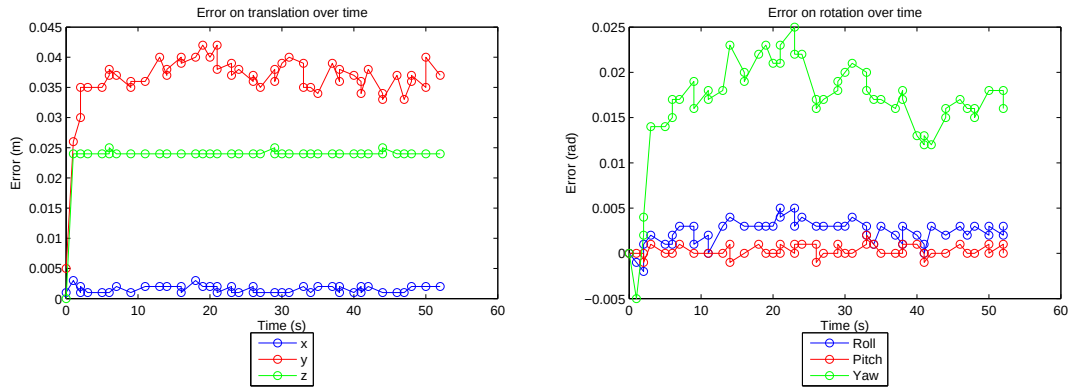

Figure 5.22: Simulation 2 — Tracking error during translation

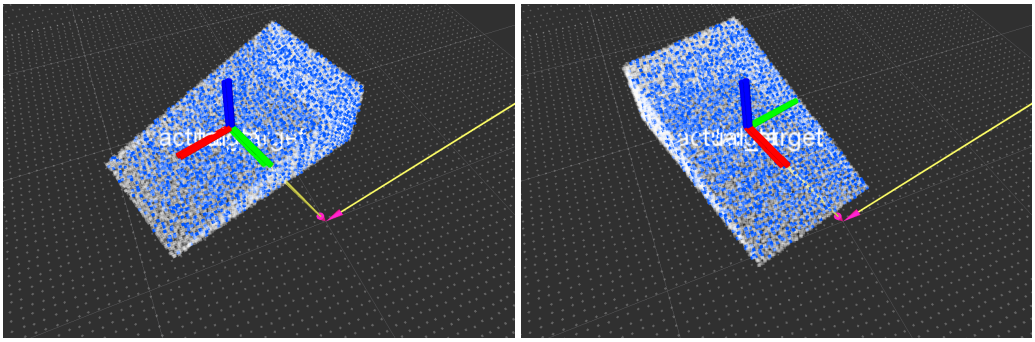

Figure 5.23: Simulation 2 — Simulated target with noise (white) and cropped point cloud (blue) during rotation

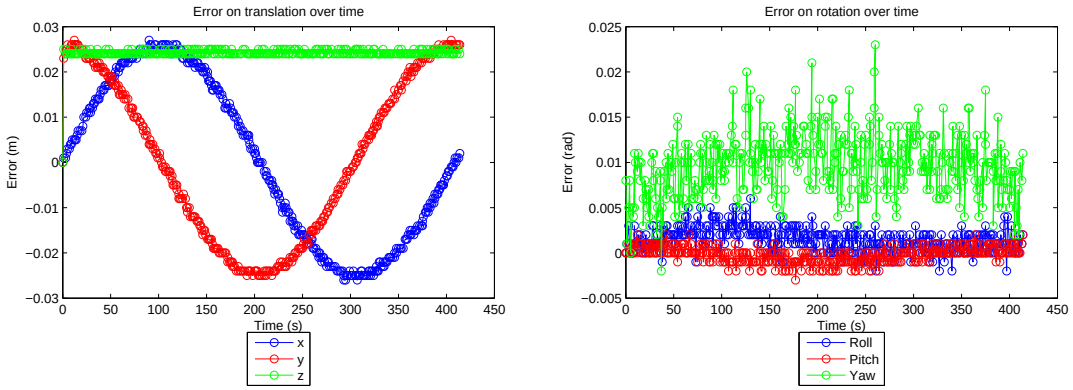

Figure 5.24: Simulation 2 — Tracking error during rotation

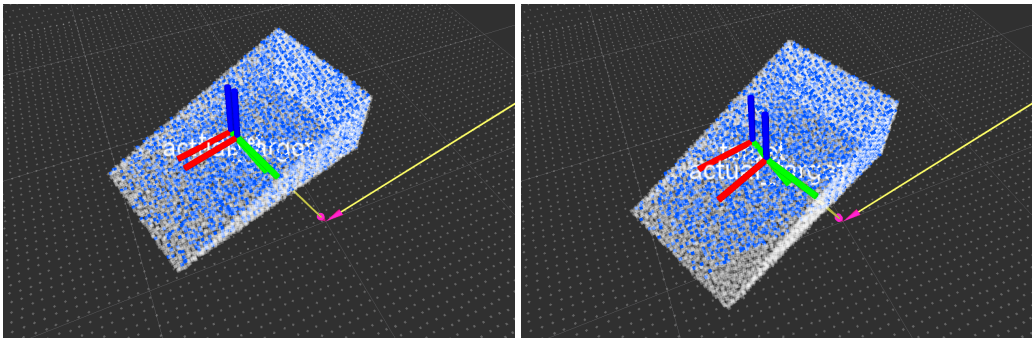

Figure 5.25: Simulation 2 — Simulated target with noise (white) and cropped point cloud (blue) during translation and rotation

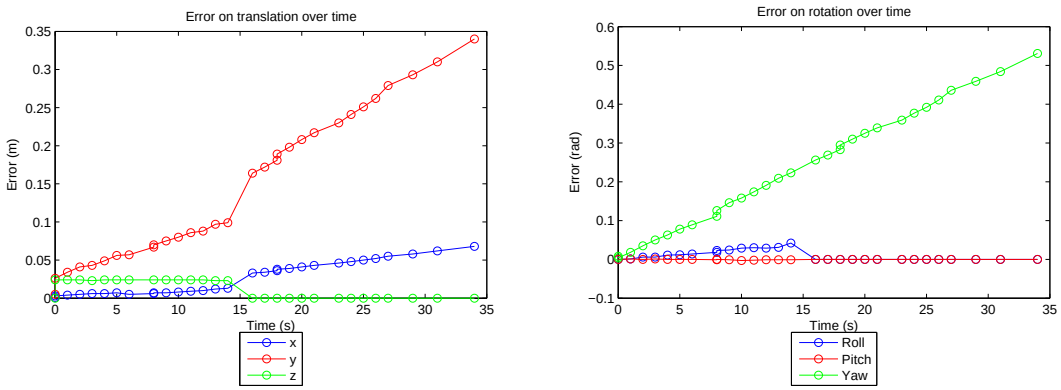

Figure 5.26: Simulation 2 — Tracking error during translation and rotation

Table 5.6: Simulation 2 — Errors while stationary (with noise)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0020	0.0260	0.0250	0.0035	0.0030	0.0020	0.0110
RMS	9.5×10^{-4}	0.0247	0.0237	0.0016	9.4×10^{-4}	8.7×10^{-4}	0.0026

Table 5.7: Simulation 2 — Errors during translation (with noise)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0030	0.0420	0.0250	0.0052	0.0050	0.0020	0.0250
RMS	0.0017	0.0365	0.0238	0.0029	0.0027	6.7×10^{-4}	0.0175

Table 5.8: Simulation 2 — Errors during rotation (with noise)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0270	0.0270	0.0250	0.0468	0.0060	0.0030	0.0230
RMS	0.0178	0.0179	0.0242	0.0309	0.0020	9.1×10^{-4}	0.0104

Table 5.9: Simulation 2 — Errors during translation and rotation (with noise)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0100	0.0880	0.0240	0.0173	0.0300	0.0030	0.1910
RMS	0.0064	0.0602	0.0231	0.0111	0.0180	0.0011	0.1080

5.11 Simulation 3 — Noise and 25% Occlusion

To evaluate the tracker's robustness when faced with occlusions, the simulated sensor was moved away from the target's initial pose so that the pass-through filter would discard a portion of its points. The sensor was placed 3.75 m away from the target's center, while the pass-through filter was set to discard points further than 3.9 m. This leaves up to ¼ of the object's volume occluded.

During these simulations, it quickly became apparent that the t (maximum error threshold) parameter (used in Algorithm 1) would have to be increased, compared to previous simulations. This parameter is used to internally determine whether continuous ICP tracking is successful or the tracker has lost its target. Increasing it allows the system to maintain confidence in its performance and continue to follow the object in this challenging situation. This comes with the trade-off of not reinitializing, when a succession of incorrect estimations occurs.

5.11.1 Stationary Target

Tracking performance with a stationary target is comparable to the previous case, despite a greater variance in rotation error. The visual effect of noise combined with occlusion is shown in Fig. 5.27

and error results are shown in the graphs of Fig. 5.28 and in Table 5.10.

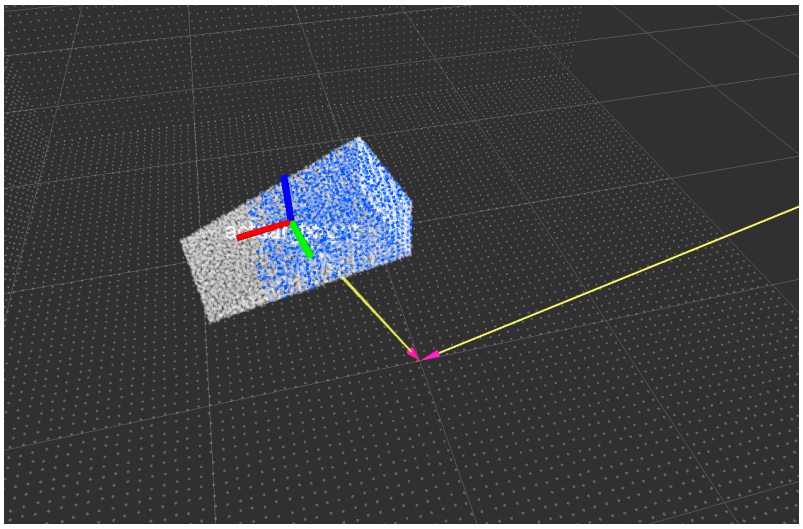

Figure 5.27: Simulation 3 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) while stationary

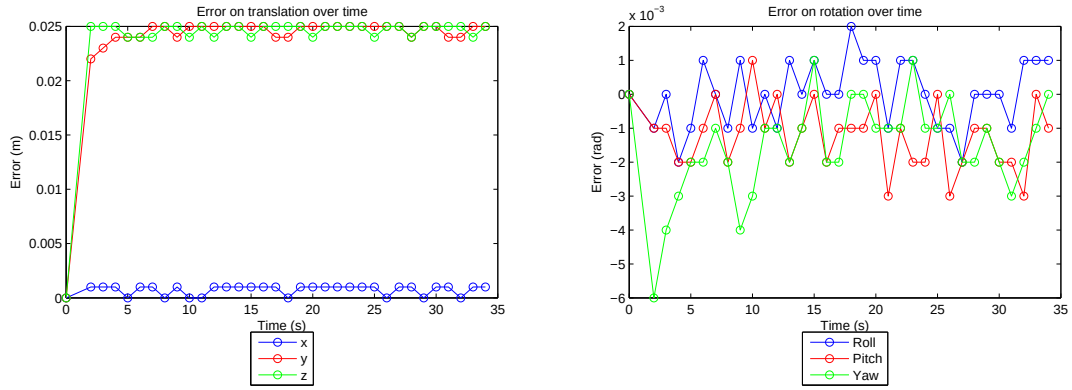

Figure 5.28: Simulation 3 — Tracking error with stationary target

5.11.2 Translation

The effect of noise and occlusion is shown in Fig. 5.29. The error value on *y* increases, approaching the set limits, as shown in Fig. 5.30 and Table 5.11. This would be expected as this is the direction of translation of the target.

5.11.3 Rotation

Results in rotation are similar to the previous case. The simulation is visualized in Fig. 5.31. Error values are well within defined limits as shown in the graphs of Fig. 5.32 and in Table 5.12.

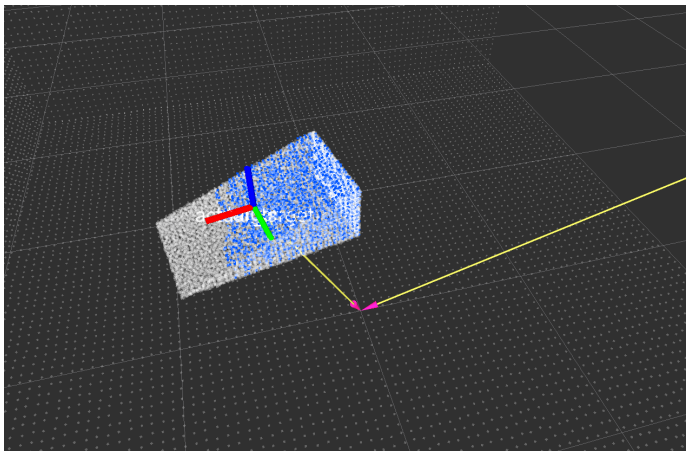

Figure 5.29: Simulation 3 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during translation

5.11.4 Translation and Rotation

As a result of increasing the tracker's internal confidence threshold, the target is not actually lost as shown in Fig. 5.33.

In this case, relevant error values in the graphs of Fig. 5.34 increase over time, as the tracker maintains confidence in its pose estimates, but does not accomplish effective tracking. This results in the error higher values of Table 5.13.

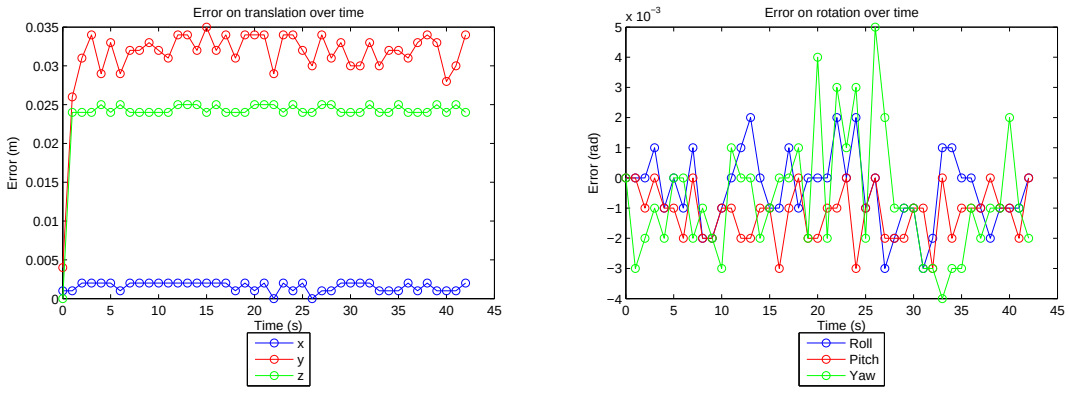

Figure 5.30: Simulation 3 — Tracking error during translation

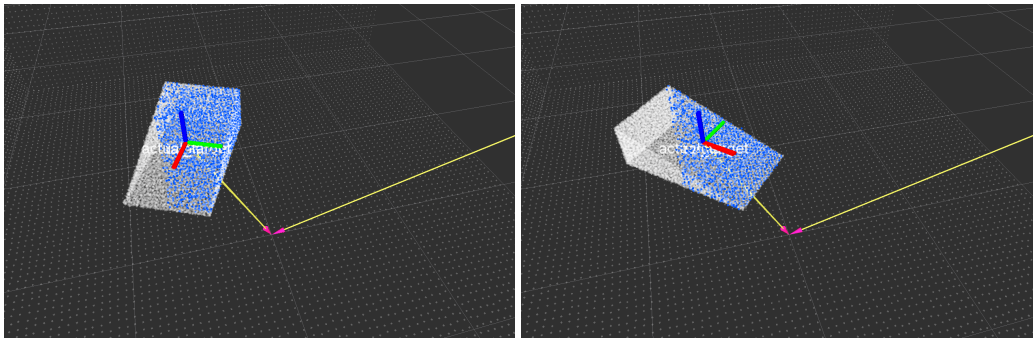

Figure 5.31: Simulation 3 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during rotation

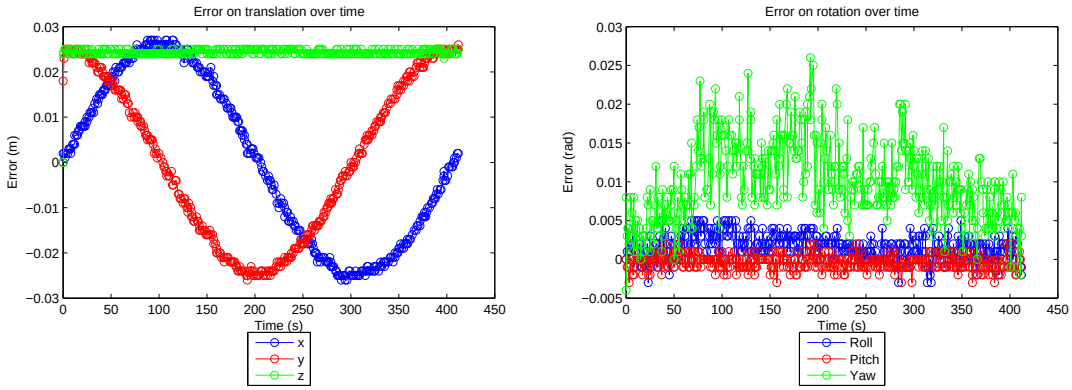

Figure 5.32: Simulation 3 — Tracking error during rotation

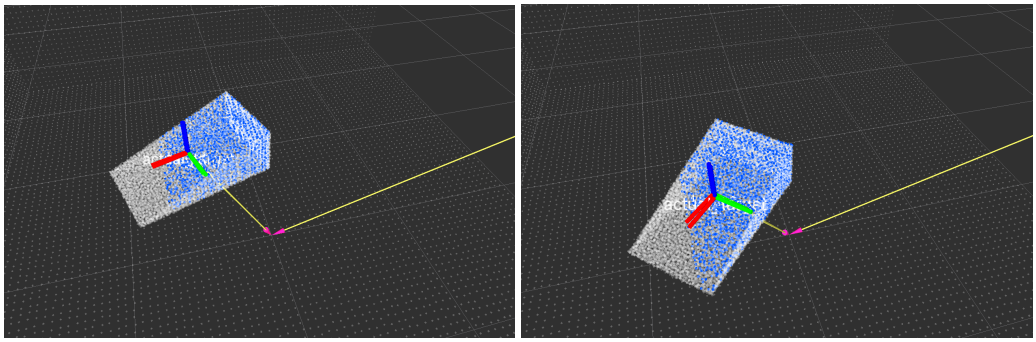

Figure 5.33: Simulation 3 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during translation and rotation

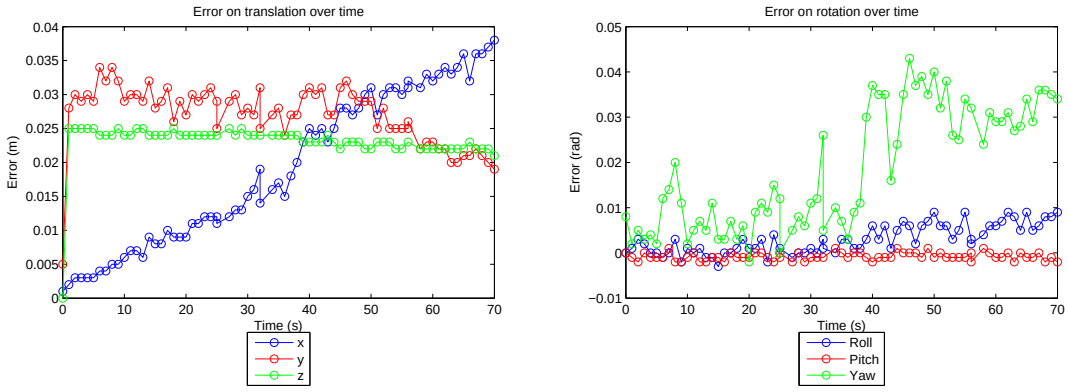

Figure 5.34: Simulation 3 — Tracking error during translation and rotation

Table 5.10: Simulation 3 — Errors while stationary (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	1.0×10^{-3}	0.0250	0.0250	0.0017	0.0020	0.0030	0.0060
RMS	8.6×10^{-4}	0.0242	0.0243	0.0015	9.7×10^{-4}	0.0015	0.0021

Table 5.11: Simulation 3 — Errors during translation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0020	0.0350	0.0250	0.0035	0.0030	0.0030	0.0050
RMS	0.0017	0.0317	0.0241	0.0029	0.0013	0.0015	0.0021

Table 5.12: Simulation 3 — Errors during rotation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0270	0.0260	0.0250	0.0468	0.0050	0.0030	0.0260
RMS	0.0180	0.0175	0.0244	0.0312	0.0023	0.0010	0.0113

Table 5.13: Simulation 3 — Errors during translation and rotation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0380	0.0340	0.0250	0.0658	0.0090	0.0020	0.0430
RMS	0.0224	0.0273	0.0233	0.0389	0.0044	0.0012	0.023

5.12 Simulation 4 — Noise and 75% Occlusion

Further occlusion was simulated by placing the sensor 4 m away from the target's center, while the pass-through filter was set to discard points further than 3.9 m. This leaves up to ¾ of the object's volume occluded.

5.12.1 Stationary Target

Tracking performance with a stationary target is comparable to the previous case. The visual effect of noise and greater occlusion is shown in Fig. 5.35 and error results are shown in the graphs of Fig. 5.36 and in Table 5.14.

5.12.2 Translation

The effect of noise and occlusion is shown in Fig. 5.37. The error values are as shown in Fig. 5.38 and Table 5.11.

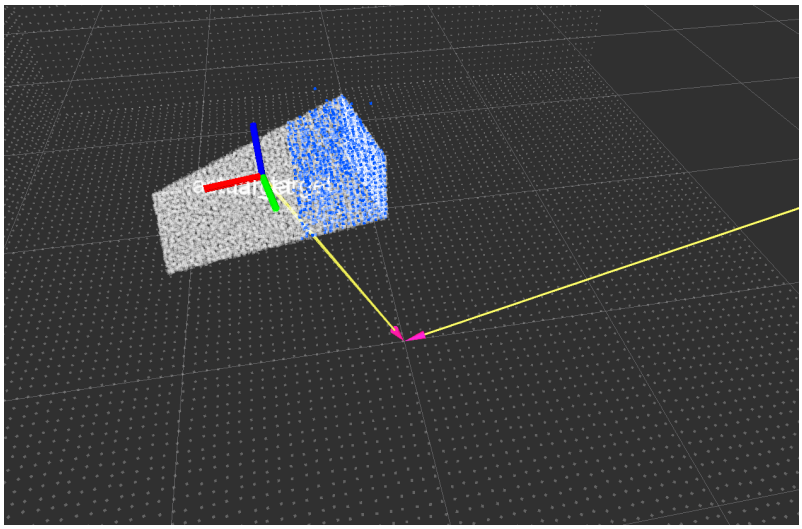

Figure 5.35: Simulation 4 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) while stationary

5.12.3 Rotation

In this case, the target is lost, after 126 s. As shown in Fig. 5.39, the number of visible points is greatly reduced during rotation, resulting in tracking failure.

ICP registration is most sensitive to incorrect initial estimates in rotation. Additionally, in this simulation there are fewer visible points, leading to an increase in ICP internal error, calculated as the sum of Euclidean squared errors. As this error increases beyond a set parameter, the target is considered lost to avoid greater errors in tracked poses.

Before this failure, error values are within defined limits as shown in the graphs of Fig. 5.40 and in Table 5.16.

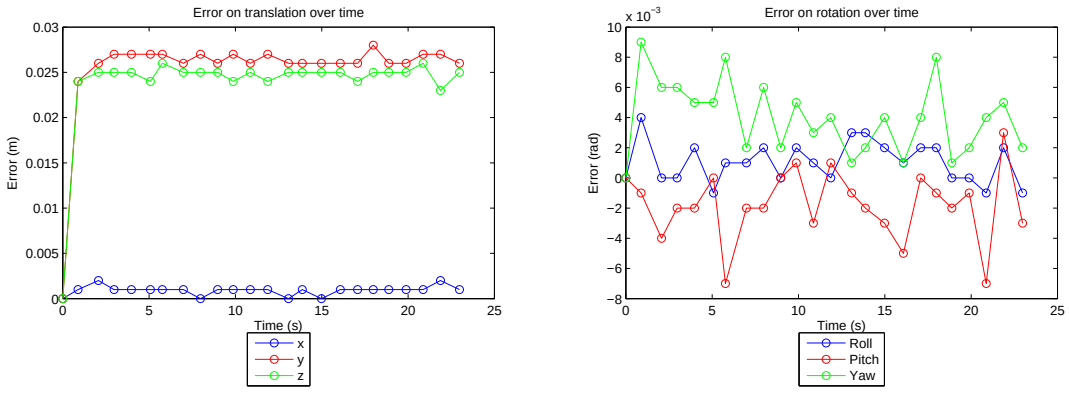

Figure 5.36: Simulation 4 — Tracking error with stationary target

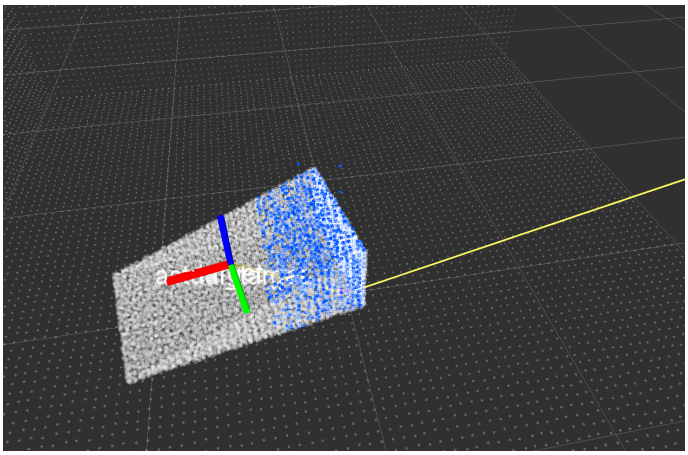

Figure 5.37: Simulation 4 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during translation

5.12.4 Translation and Rotation

Tracking performance during translation and rotation is similar to the case of rotation and fails in the same way: too few points are visible for tracking, while the update rate is not high enough. Results for this test are shown in Fig. 5.41.

In this case, relevant error values in the graphs of Fig. 5.42 and Fig. 5.43 increase over time, as the tracker maintains confidence in its pose estimates, but does not accomplish effective tracking. This results in the higher error values of Table 5.17.

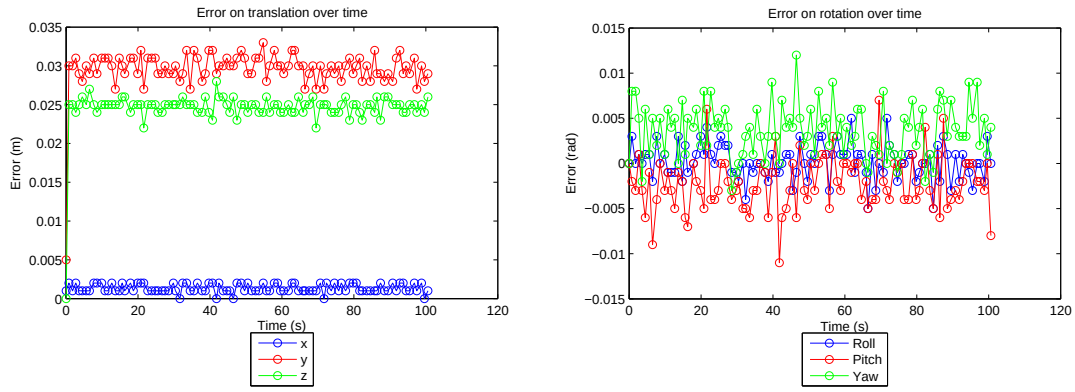

Figure 5.38: Simulation 4 — Tracking error during translation

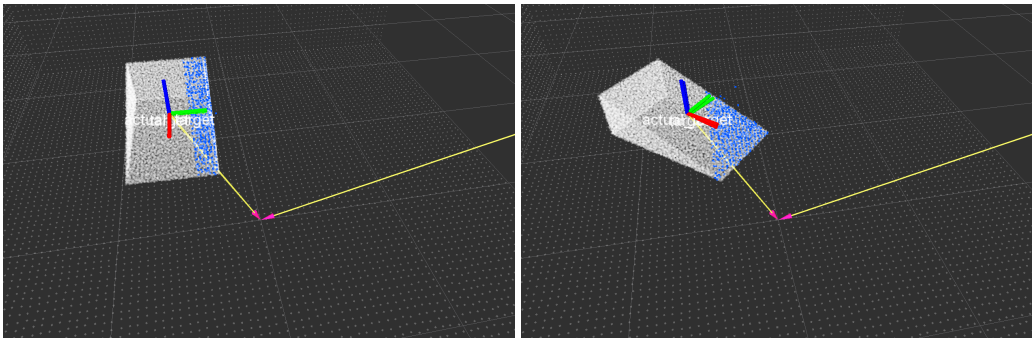

Figure 5.39: Simulation 4 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during rotation

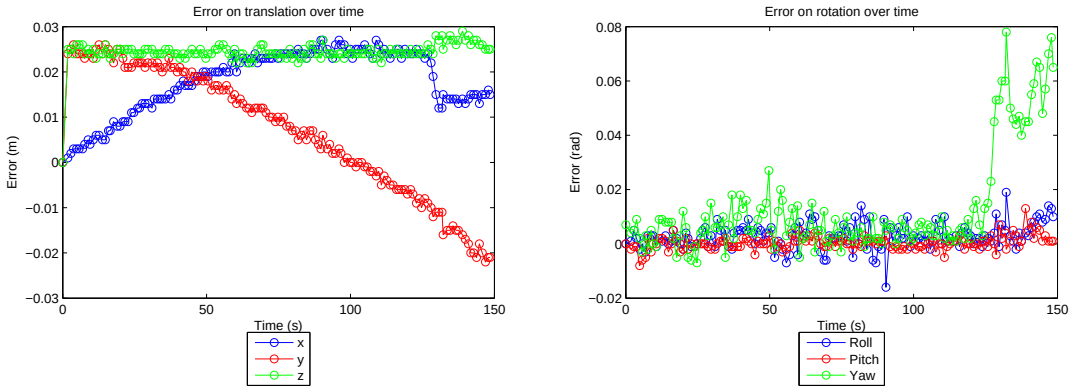

Figure 5.40: Simulation 4 — Tracking error during rotation

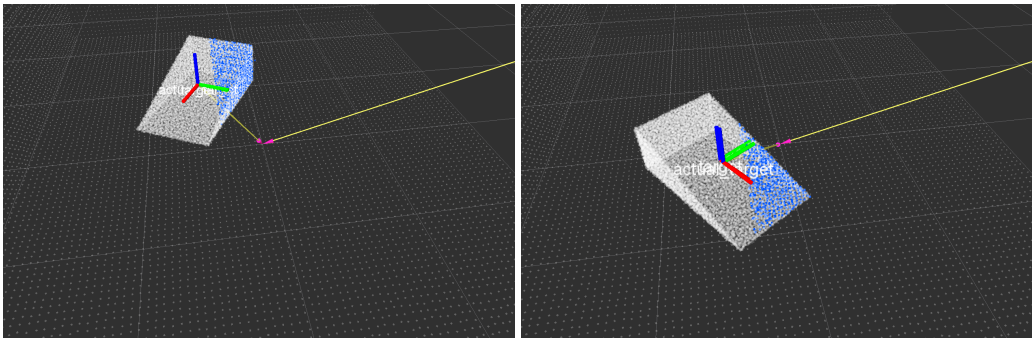

Figure 5.41: Simulation 4 — Simulated target with noise and occlusion (white) and cropped point cloud (blue) during translation and rotation

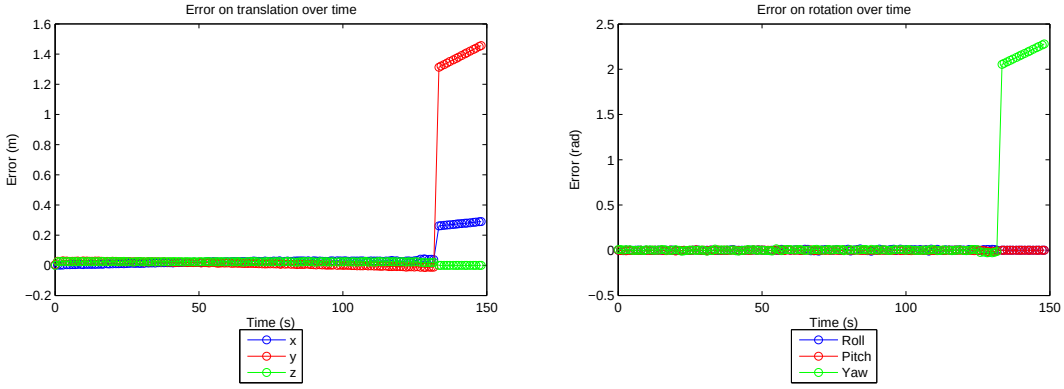

Figure 5.42: Simulation 4 — Tracking error during translation and rotation

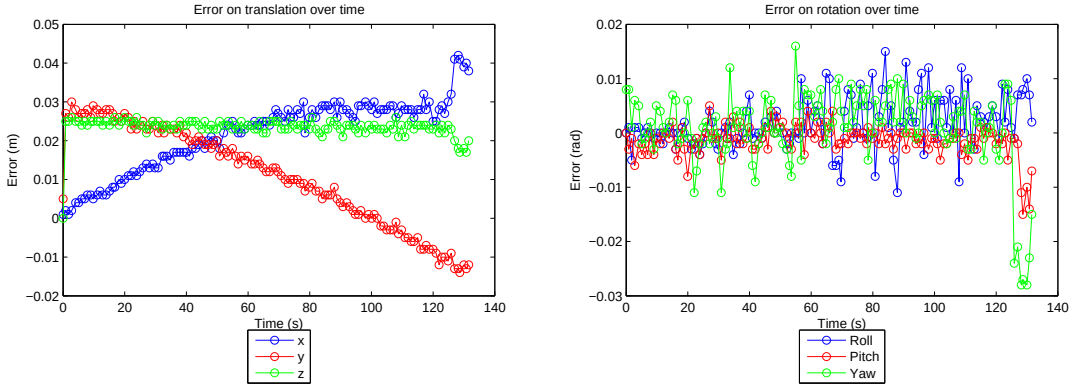

Figure 5.43: Simulation 4 — Tracking error during translation and rotation (before target is lost)

Table 5.14: Simulation 4 — Errors while stationary (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0020	0.0280	0.0260	0.0035	0.0040	0.0070	0.0090
RMS	0.0010	0.0258	0.0243	0.0018	0.0017	0.0029	0.0046

Table 5.15: Simulation 4 — Errors during translation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0020	0.0330	0.0280	0.0035	0.0050	0.0110	0.0120
RMS	0.0014	0.0297	0.0247	0.0025	0.0019	0.0037	0.0047

Table 5.16: Simulation 4 — Errors during rotation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0270	0.0260	0.0270	0.0468	0.0160	0.0080	0.0270
RMS	0.0198	0.0155	0.0240	0.0344	0.0048	0.0020	0.0080

Table 5.17: Simulation 4 — Errors during translation and rotation (with noise and occlusion)

Error	Translation (m)				Rotation (rad)		
	x	y	z	Euclidean	Roll	Pitch	Yaw
Max	0.0320	0.0300	0.0260	0.0554	0.0150	0.0080	0.0160
RMS	0.0223	0.0170	0.0238	0.0386	0.0049	0.0023	0.0052

Table 5.18: Average tracker update rates (Hz)

Simulation	Stationary	Translation	Rotation	Translation and Rotation
1 — No noise	3.7456	2.6143	2.7370	3.1314
1 — Smaller voxel grid	—	—	3.6500	—
2 — With noise	3.3353	2.6965	2.6100	2.9979
3 — Noise, 25% Occlusion	3.6977	2.7049	3.0167	2.9076
4 — Noise, 75% Occlusion	3.5073	3.0408	3.1109	2.9501

5.13 Conclusion

The project brought many challenges including familiarization with different technologies and demanding hardware requirements that could not fully be met.

During the development of this project, several approaches were explored, bringing a better understanding of their advantages and limitations and an appropriate path was established.

Simulation results are satisfactory with error in translation under 5 cm and rotation under 5°. Tolerable inaccuracies originate in necessary preprocessing and finite numerical precision. This operation adapts the system to sensor noise and increases the update rate, by reducing the number

of points that are processed in ICP registration. The system is also shown to be robust to random sensor noise and partial occlusion.

The perceived point of failure is a low update rate. Average rates for the different simulations are collected in Table 5.18. These are consistently higher than 2 Hz, but should still be increased. Further reducing the number of points in registration would have a positive impact in this, but it would reduce accuracy. Other parameters that affect update rate are ICP internal parameters such as the maximum number of iterations and minimum estimation error. This problem can be better approached by using better dedicated hardware, as suggested in Section 4.6 that would take advantage of GPU optimized algorithms in PCL.

Chapter 6

Data Collection

6.1 Introduction

The Cooperative Robot for Large Spaces Manufacturing (CARLoS) project aims to apply modern robotics solutions to the automation of stud welding in ship structures, while cooperating with human operators [64, 65].

Several organizations are involved in this project, including INESC TEC, making the Automated Guided Vehicle (AGV) robot platform available for data collection for external robot localization.

Figure 6.1: CARLoS AGV [64]

6.2 Localization Requirements

As shown in Fig. 6.1, the AGV is tracked and can traverse irregular terrain, such as ramps or steps. This possibility requires a localization system in 6 Degrees of Freedom (DoF). It is, therefore, an example of a system that could gain from the proposed tracker.

6.3 Collection Environment

Data was collected at LABIOMEP [53] with two Kinect for Xbox 360 devices mounted on tripods collecting Red, Green and Blue, and Depth (RGB-D) video and a Qualisys Oqus motion capture system tracking passive Infrared (IR) markers on the robot and on the sensors to provide their poses.

This set-up is seen on Fig. 6.2. Note the IR markers on both the AGV and the sensors. The robot moves in the room, including climbing a ramp.

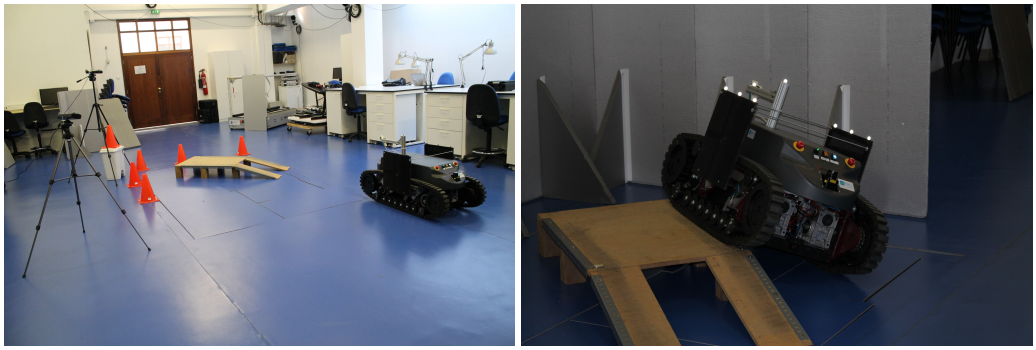

Figure 6.2: Data collection environment at LABIOMEP

The LABIOMEP tracking set-up uses 12 Qualisys Oqus cameras and cost around 150 000€. Some of these cameras are highlighted in Fig. 3.5.

The placement of Kinect devices followed the recommendation of Fig. 4.1 for improved occlusion handling and constructive data fusion.

6.4 Collected Data

The results of the session were:

- Data collected by the CARLoS robot from:
  - Structure IO 3D sensor
  - Hokuyo URG-04LX front Laser Range Finder
  - Hokuyo URG-04LX_UG01 back Laser Range Finder
  - SparkFun 9DOF MPU-915 IMU
  - CH Robotics UM7 IMU
- RGB-D datasets from each Kinect for Xbox 360 device, while robot moves;
- One dataset from a hand-held Kinect for Xbox 360 for mapping;
- Ground truth using Qualisys motion tracking system with 12 Oqus cameras
  - 4 high speed Oqus 310+ cameras with 640×512 resolution @ 1764fps
  - 8 high resolution Oqus 400 cameras with 1696×1710 resolution @ 480fps

6.4.1 Model Point Clouds

A point cloud can be sampled from 3D meshes of the AGV platform, to be used as a target model. Examples are shown in Fig. 6.3. These meshes are available with the rest of the dataset.

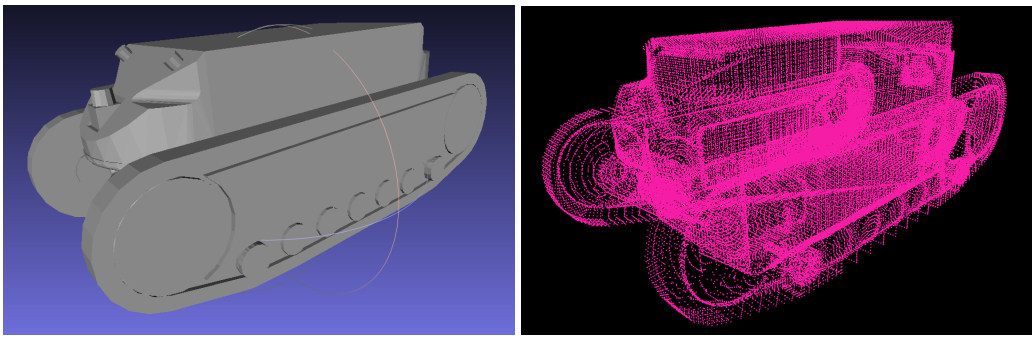

Figure 6.3: Guardian platform mesh (left) and point cloud (right)

6.5 Conclusion

These datasets are available for the CARLoS team for further development and validation of the self-localization and mapping systems and have been made available for future development in 6 DoF tracking at `https://github.com/carlosmccosta/dynamic_robot_localization_tests`.

This dataset shows dynamic movements of a robot in irregular terrain from two different perspectives with RGB-D data and ground truth, as well as the sensors' poses in the global frame, making it very interesting for future work.

Chapter 7

Conclusion and Future Work

Mobile robots and Automated Guided Vehicles (AGVs) operate in diverse environments and require adequate localization. The burden of this task can be taken from the robot by using video tracking for external localization.

Six Degrees of Freedom (DoF) video tracking is a non-trivial problem. There are a number of significant design decisions that impact performance and accuracy, as well as the range of application of a tracker. Current efforts attempt to leverage new technology and are often tailored to specific targets and environments.

A set of relevant software and hardware has been identified, for both development and deployment use, including a flexible robotics middleware platform and Red, Green and Blue, and Depth (RGB-D) library and devices.

The system was developed with and will use Robot Operating System (ROS) and Point Cloud Library (PCL). The hardware used in development was the Microsoft Kinect for Xbox 360 sensor and Microsoft Kinect for Xbox One is recommended for deployment.

The project brought many challenges including familiarization with different technologies and demanding hardware requirements that could not fully be met.

During the development of this project, several approaches were explored, bringing a better understanding of their advantages and limitations and an appropriate path was established.

Simulation results are satisfactory and promising for real-word application. The system is also shown to be robust to random sensor noise and partial occlusion.

The perceived point of failure is a low update rate. This problem can be approached by using better dedicated hardware, as suggested in Section 4.6 that would take advantage of Graphics Processing Unit (GPU) optimized algorithms in PCL.

The collected RGB-D datasets with 6 DoF ground truth are available for the Cooperative Robot for Large Spaces Manufacturing (CARLoS) team for further development and validation of the self-localization and mapping systems and has been made available for future development in 6 DoF tracking.

This dataset shows dynamic movements of a robot in irregular terrain from two different perspectives with RGB-D data and ground truth, as well as the sensors' poses in the global frame,

making it very interesting for future work.

The proposed solution is very cost effective (2570€ for a set-up with 3 powerful sensors and computers at 2015 prices), when compared to other systems, as discussed in Section 4.6.

7.1 Fulfillment of Objectives

As proposed, a system was developed for external robot localization in 6 DoF via video tracking.

The system has the following characteristics demonstrated in simulation with random noise:

- Capable of tracking one rigid object or the main rigid body of an articulated object, indoors;
- Capable of tracking in full 6 DoF without assumptions on the target moving on a floor plane;
- Maximum translation error of 5 cm or 5×10^{-2} m in either axis;
- Maximum rotation error of 5° or 8.7×10^{-2} rad;
- Minimum update frequency of 2 Hz.

7.2 Contributions

This work introduces a multiple sensor 6 DoF tracking architecture, for external localization, based on ROS framework using PCL processing and off-the-shelf inexpensive RGB-D sensors.

A 6 DoF tracking dataset with ground truth also results from this work, available at `https://github.com/carlosmccosta/dynamic_robot_localization_tests`.

7.3 Future Work

Future work should take advantage of unexplored features in ROS and PCL, as well as better hardware.

Interesting directions for future development are:

- Using more powerful hardware including a dedicated GPU and optimized PCL features;
- Explore use of Red, Green and Blue color (RGB) data in initial pose estimation;
- Implement a ROS parameter server for dynamic reconfiguration;
- Design a Graphical User Interface (GUI) that would allow a human operator to provide an initial pose estimate;
- Conduct real-word evaluation with collected dataset.

References

[1] Daniel F. Dementhon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2):123–141, 1995. ISSN 09205691. doi: 10.1007/BF01450852. URL http://link.springer.com/article/10.1007/BF01450852. Cited on page 2.

[2] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking. *ACM Computing Surveys*, 38(4):13–es, 2006. ISSN 03600300. doi: 10.1145/1177352.1177355. Cited on pages 5 and 8.

[3] Paul Besl and Neil McKay. A Method for Registration of 3-D Shapes, 1992. ISSN 0162-8828. Cited on pages 7 and 12.

[4] Dirk-Jan Kroon. Finite Iterative Closest Point - File Exchange - MATLAB Central, 2009. URL http://www.mathworks.com/matlabcentral/fileexchange/24301-finite-iterative-closest-point. [Accessed 2015-08-12]. Cited on page 7.

[5] D.a. Simon, M. Hebert, and T. Kanade. Real-time 3-D pose estimation using a high-speed range sensor. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994. doi: 10.1109/ROBOT.1994.350953. Cited on pages 7 and 8.

[6] Shuran Song and Jianxiong Xiao. Tracking revisited using RGBD camera: Unified benchmark and baselines. *Proceedings of the IEEE International Conference on Computer Vision*, pages 233–240, 2013. ISSN 1550-5499. doi: 10.1109/ICCV.2013.36. Cited on pages 7, 15, and 16.

[7] Jihua Zhu, Nanning Zheng, and Zejian Yuan. An improved technique for robot global localization in indoor environments. *International Journal of Advanced Robotic Systems*, 8(1): 21–28, 2011. ISSN 17298806. doi: 10.5772/10525. Cited on pages 7 and 9.

[8] Rudolf Emil Kálmán. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960. ISSN 00219223. doi: 10.1115/1.3662552. URL http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402. Cited on page 7.

[9] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *In Practice*, 7:1–16, 2006. ISSN 10069313. doi: 10.1.1.117.6808. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6578{&}amp;rep=rep1{&}amp;type=pdf. Cited on page 7.

[10] Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. In *Proceedings of the IEEE*, volume 92, pages 401–422, 2004. Cited on page 8.

[11] Wikimedia. File:Kalman.png - Wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/wiki/File:Kalman.png. [Accessed 2015-08-12]. Cited on page 8.

[12] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation, 1993. ISSN 0956375X. Cited on page 8.

[13] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002. Cited on page 8.

[14] Pedram Azad, David Munch, Tamim Asfour, and Rüdiger Dillmann. 6-DoF model-based tracking of arbitrarily shaped 3D objects. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5204–5209, 2011. Cited on pages 9 and 14.

[15] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2:559–572, 1901. URL http://dx.doi.org/10.1080/14786440109462720. Cited on page 9.

[16] H. Hotelling. Analysis of a complex of statistical variables into principal components., 1933. ISSN 0022-0663. Cited on page 9.

[17] Jonathon Shlens. A Tutorial on Principal Component Analysis. *Measurement*, 51:52, 2005. doi: 10.1.1.115.3503. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.3503{&}amp;rep=rep1{&}amp;type=pdf. Cited on pages 9 and 10.

[18] Umar Asif, Mohammed Bennamoun, and Ferdous Sohel. Real-time pose estimation of rigid objects using RGB-D imagery. *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pages 1692–1699, jun 2013. doi: 10.1109/ICIEA.2013.6566641. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6566641. Cited on pages 10 and 15.

[19] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 2, 1999. Cited on pages 10 and 11.

[20] D G Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image, 2004. URL http://www.google.com/patents/US6711293. Cited on page 10.

[21] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3951 LNCS:404–417, 2006. ISSN 03029743. doi: 10.1007/11744023{_}32. Cited on pages 10 and 11.

[22] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011. Cited on pages 11 and 12.

[23] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3951 LNCS, pages 430–443, 2006. Cited on page 11.

[24] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. *BRIEF: Binary robust independent elementary features*, volume 6314 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15560-4. doi: 10.1007/978-3-642-15561-1. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-78149309478{&}partnerID=tZOtx3y1. Cited on page 11.

[25] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D registration. *2009 IEEE International Conference on Robotics and Automation*, 2009. Cited on pages 12 and 13.

[26] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. *2008 10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008*, (December):643–650, 2008. doi: 10.1109/ICARCV.2008.4795593. Cited on page 12.

[27] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3384–3391, 2008. doi: 10.1109/IROS.2008.4650967. Cited on page 12.

[28] Martin a Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applicatlons to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381 – 395, 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL http://dx.doi.org/10.1145/358669.358692. Cited on page 13.

[29] Pedram Azad, Tamim Asfour, and Rudiger Dillmann. Accurate shape-based 6-DoF pose estimation of single-colored objects. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2690–2695, oct 2009. doi: 10.1109/IROS.2009.5354606. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5354606. Cited on pages 13 and 14.

[30] Pablo Iigo-Blasco, Fernando Diaz-Del-Rio, Ma Carmen Romero-Ternero, Daniel Cagigas-Muiz, and Saturnino Vicente-Diaz. Robotics software frameworks for multi-agent robotic systems development. *Robotics and Autonomous Systems*, 60(6):803–821, 2012. ISSN 09218890. doi: 10.1016/j.robot.2012.02.004. URL http://dx.doi.org/10.1016/j.robot.2012.02.004. Cited on page 17.

[31] Morgan Quigley, Ken Conley, Brian Gerkey, Josh FAust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Mg. ROS: an open-source Robot Operating System. *Icra*, 3(Figure 1):5, 2009. doi: http://www.willowgarage.com/papers/ros-open-source-robot-operating-system. URL http://pub1.willowgarage.com/{~}konolige/cs225B/docs/quigley-icra2009-ros.pdf. Cited on pages 17 and 18.

[32] Open Source Robotics Foundation. tf2 - ROS Wiki, 2015. URL http://wiki.ros.org/tf2. [Accessed 2015-08-20]. Cited on page 18.

[33] Open Source Robotics Foundation. Bags - ROS Wiki, 2015. URL http://wiki.ros.org/Bags. [Accessed 2015-08-20]. Cited on page 19.

[34] Open Perception Foundation. PCL - Point Cloud Library (PCL), 2015. URL http://pointclouds.org/. [Accessed 2015-02-07]. Cited on pages 19 and 33.

[35] Radu Bogdan Rusu and S. Cousins. 3D is here: point cloud library. *IEEE International Conference on Robotics and Automation*, 2011. URL http://pointclouds.org/. Cited on page 19.

[36] Open Source Robotics Foundation. pcl - ROS Wiki, 2015. URL http://wiki.ros.org/pcl. [Accessed 2015-02-07]. Cited on page 19.

[37] Elvis Ruiz, Raúl Acuña, Novel Certad, Angel Terrones, and María Eugenia Cabrera. Development of a control platform for the mobile robot Roomba using ROS and a Kinect sensor. *2013 IEEE Latin American Robotics Symposium*, 2013. Cited on pages 19, 20, and 33.

[38] Zheng-Yang Huang, Jung-Tang Huang, and Chih-Ming Hsu. A Case Study of Object Identification Using a Kinect Sensor. *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1743–1747, 2013. doi: 10.1109/SMC.2013.300. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6722053. Cited on pages 20 and 33.

[39] Jrgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. *IEEE International Conference on Intelligent Robots and Systems*, pages 573–580, 2012. Cited on pages 20 and 33.

[40] Inneke Mayachita, Rizka Widyarini, Hadi Rasyid Sono, Adrianto Ravi Ibrahim, and Widyawardana Adiprawita. Implementation of Entertaining Robot on ROS Framework. *Procedia Technology*, 11(Iceei):380–387, 2013. ISSN 22120173. doi: 10.1016/j.protcy.2013.12.206. URL http://www.sciencedirect.com/science/article/pii/S2212017313003605. Cited on pages 20 and 33.

[41] Microsoft. Kinect for Windows Sensor Components and Specifications, 2015. URL https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed 2015-02-08]. Cited on pages 20 and 33.

[42] Microsoft. DepthImageFormat Enumeration, 2015. URL https://msdn.microsoft.com/en-us/library/microsoft.kinect.depthimageformat.aspx. [Accessed 2015-02-08]. Cited on page 20.

[43] Mircosoft. Coordinate Spaces, 2015. URL https://msdn.microsoft.com/en-us/library/hh973078.aspx{#}Depth{_}Ranges. [Accessed 2015-02-08]. Cited on page 20.

[44] Microsoft. Purchase the Kinect for Windows sensor, 2015. URL http://www.microsoft.com/en-us/kinectforwindows/purchase/{#}tab=1. [Accessed 2015-02-11]. Cited on pages 20 and 31.

[45] Microsoft. Kinect for Windows features, 2015. URL http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx. [Accessed 2015-02-11]. Cited on page 21.

[46] Tom Kerkhove. Visug: Say Hello to my little friend: a session on Kinect, 2014. URL http://www.slideshare.net/Visug/visug-say-hello-to-my-little-friend-a-session-on-kinect. [Accessed 2015-08-25]. Cited on page 21.

[47] Microsoft. Download Kinect for Windows SDK 2.0 from Official Microsoft Download Center, 2015. URL https://www.microsoft.com/en-us/download/details.aspx?id=44561. [Accessed 2015-08-14]. Cited on page 20.

[48] OpenKinect. OpenKinect, 2012. URL http://openkinect.org/wiki/Main{_}Page. [Accessed 2015-08-14]. Cited on page 21.

[49] ASUSTeK Computer Inc. Multimedia | Xtion PRO LIVE | ASUS USA, 2015. URL https://www.asus.com/us/Multimedia/Xtion{_}PRO{_}LIVE/. Cited on page 21.

[50] ASUSTeK Computer Inc. Multimedia | Xtion PRO LIVE | ASUS USA, 2015. URL https://www.asus.com/us/Multimedia/Xtion{_}PRO{_}LIVE/specifications/. [Accessed 2015-08-25]. Cited on page 21.

[51] Occipital Inc. The Structure Sensor is the first 3D sensor for mobile devices, 2015. URL http://structure.io/. [Accessed 2015-08-25]. Cited on page 21.

[52] Occipital Inc. Get Started with the Structure Sensor, 2015. URL http://structure.io/getstarted{#}connectother. [Accessed 2015-08-25]. Cited on page 21.

[53] LABIOMEP. LABIOMEP | Porto Biomechanics Laboratory, 2015. URL http://www.labiomep.up.pt/. [Accessed 2015-02-12]. Cited on pages 22 and 62.

[54] Qualisys. Oqus camera series – Oqus high-speed camera – Qualisys Motion Capture Systems, 2015. URL http://www.qualisys.com/products/hardware/oqus/. [Accessed 2015-02-11]. Cited on page 22.

[55] Huiyu Zhou, Huiyu Zhou, Huosheng Hu, and Huosheng Hu. A Survey - Human Movement Tracking and Stroke Rehabilitation. *System*, (December), 2004. Cited on page 22.

[56] Microsoft. Kinect for Windows sensor setup, 2015. URL https://www.microsoft.com/en-us/kinectforwindows/purchase/sensor{_}setup.aspx. [Accessed 2015-08-18]. Cited on page 31.

[57] Open Source Robotics Foundation. Documentation - ROS Wiki, 2015. URL http://wiki.ros.org/. [Accessed 2015-08-20]. Cited on page 33.

[58] A Segal, D Haehnel, and S Thrun. Generalized-ICP. *Robotics: Science and Systems*, 2009. doi: 10.1.1.149.3870. Cited on page 33.

[59] Open Perception Foundation. Spatial change detection on unorganized point cloud data - Point Cloud Library (PCL), 2015. URL http://pointclouds.org/documentation/tutorials/octree{_}change.php. [Accessed 2015-08-20]. Cited on page 34.

[60] Open Perception foundation. Euclidean Cluster Extraction - Point Cloud Library (PCL), 2015. URL http://www.pointclouds.org/documentation/tutorials/cluster{_}extraction.php. [Accessed 2015-08-20]. Cited on page 34.

[61] Nicolas Burrus. RGBDemo - Home Page, 2011. URL http://rgbdemo.org/. [Accessed 2015-08-17]. Cited on page 35.

[62] Nicolas Burrus. RGBDemo - Multiple Kinect, 2011. URL http://rgbdemo.org/index.php/Documentation/MultipleKinect. [Accessed 2015-08-17]. Cited on page 35.

[63] Open Perception Foundation. Documentation - Point Cloud Library (PCL), 2015. URL http://pointclouds.org/documentation/tutorials/gpu{_}install.php. [Accessed 2015-09-03]. Cited on page 37.

[64] The CARLoS Project. The CARLoS Project, 2013. URL http://carlosproject.eu/. [Accessed 2015-08-19]. Cited on page 61.

[65] Carlos M Costa, Héber M Sobreira, Armando J Sousa, and Germano M Veiga. Robust and Accurate Localization System for Mobile Manipulators in Cluttered Environments. *2015 IEEE International Conference on Industrial Technology, At Sevilla*, (October):3308–3313, 2015. doi: 10.1109/ICIT.2015.7125588. Cited on page 61.