Indoor navigation with motion tracking and depth perception sensors

Mimozë Lladrovci

Supervisor: Prof. Mariusz Nowostawski

01-06-2016

Master's Thesis
Master of Science in Applied Computer Science
30 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology, 2016

Avdeling for
informatikk og medieteknikk
Norges teknisk-naturvitenskapelige universitet
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Norwegian University of Science and Technology
Box 191
N-2802 Gjøvik
Norway

Abstract

With all the advances in technology, indoor navigation continues to remain an open problem. While we are able to use satellite-based navigation systems such as the Global Positioning System (GPS) and automative navigation systems for navigation outdoors, these systems are not viable for use indoors. Attempts at solving the indoor navigation problem by using approaches such as wireless positioning, RFID tag or beacon placement, among others, have been made, but due to weaknesses of these technologies, their use has not become the standard and widely accepted solution. In this work, we designed an indoor navigation system based on motion tracking and depth perception sensors on board a mobile device, Goolge's Project Tango. The main advantage of a device with motion tracking and depth perception sensors on board over other candidates for indoor navigation is that it is self-contained, which allows the implementation of an entire indoor navigation system with a single device. Other advantages include low cost and high accuracy in obstacle detection and positioning of the user. The navigation system is designed to guide the user through a collision-free path from start to destination while achieving the shortest path and real-time guidance. Parts of such a system have already been implemented and solutions to some of the problems are presented, but the field lacks a good overview of the indoor navigation workflow and it is not clear how solutions to smaller problems help in solving the bigger problem of indoor navigation. Our work examines the necessary components of the indoor navigation system and breaks down each component to smaller elements, leading in a detailed indoor navigation workflow. Furthermore, we map the solutions that have been provided to some of the challenges that indoor navigation presents to the workflow. Ultimately, we present an obstacle avoidance algorithm that combines global and local motion planning to guide the user around obstacles to the destination. Global path planning alone allows planning of the path from the start to the destination but does not allow room for changes if obstacles are presented along the way. On the other hand, local path planning by itself can deal with path planning in the short term but cannot guarantee that the destination is reached. By combining both approaches so that the global approach is used to plan the total path and the local approach to plan only the parts of the global path that are affected by obstacles, we overcome their weaknesses and achieve a collision-free path to the destination.

Preface

This thesis was inspired by the lack of navigation systems that can be used indoors, by blind people in particular, despite the fact that the development of technology has come so far. Having Project Tango, the first fully-fledged mobile device with motion tracking and depth perception sensors at our disposal, we set out to implement a navigation system in order to evaluate the feasibility of the use of motion tracking and depth perception sensors for navigation purposes. Since we had blind people in mind, we knew that the system had to include obstacle avoidance, otherwise it would not serve them. While trying to implement the system, we ran into challenges that new and unexplored technologies such as Project Tango bring. Not having any work that designed or discussed the challenges and the workflow of navigation systems with motion tracking and depth perception in its entirety to build our work on, we had to change our goal of implementation and ended up providing the required navigation workflow with all the challenges we identified while trying to implement the system. Without this workflow, anyone who endeavours to implement an indoor navigation system with motion tracking and depth perception in a limited time will run into the same unexpected challenges and either fail to implement the system and do what we did, or stop to overcome the first challenges they meet. If many researchers stop to solve the same challenges of the beginning, the indoor navigation system will take a very long time to complete. By providing this work, we hope to accelerate the implementation of the indoor navigation system that we have designed so that we and blind people will be able to use it soon.

Working with technology that only sees depth points in space has opened our eyes to how much we perceive with them. Not only do we have a big field of view and do not get completely blinded by sunlight and heat, but we see in shapes, not merely in coordinates. As will become apparent later, while we take these for granted when it comes to us, revealing shape to a zero-one system with limited field of view is anything but easy. Therefore this work shall serve as an ode to evolution in general and the evolution of sight in particular.

This thesis would not have been possible without the unconditional support and love of family, the encouragement and inspiration of extraordinary friends, the discussions with classmates, the help and time of my supervisor Mariusz Nowostawski and ultimately without all the people who were stupefied at the amount of work that passing by a chair without colliding with it using technology takes. More often than not, just talking to others about this thesis helped to formulate and understand the problem we were trying to solve better. I want to thank everyone who played a direct role in this thesis, but also everyone who just by existing was an inspiration to move forward.

Përfundimisht, dëshiroj t'a falenderoj familjen time, pa përkrahjen dhe dashurinë e vazhdueshme të të cilëve nuk do të isha sot këtu duke i realizuar ëndërrat e mija. I falenderoj posaçërisht prindërit e mij, por edhe motrat e vëllaun, që janë gjithmonë të gatshëm të më dëgjojnë, ndihmojnë e shtyjnë përpara. Pa juve as ky punim, as asgjë tjetër e bukur nuk do të ishte e mundur. Falemnderit shumë!

List of Figures

Contents

1 Introduction

Navigation is the act of guiding a person, vehicle, plane, ship or robot from one point to another. Normally, when we hear the word *navigation*, our mind goes to the navigation system in our car, the navigation with Google Maps that we do in all the new cities we are in, but we rarely think of any navigation system that navigates us indoors. That is, most likely, because we have never used or have knowledge of any such system. Nonetheless, navigation in indoor environments is just as useful as navigation outdoors because we find ourselves in new indoor environments where we have difficulties finding things and places we want quite often.

Think about it: how do you find an office in a building full of offices you have never been to? How do you find the section you have to go to in the city hospital? How do you find the product that you want to buy in the supermarket? You probably do so by trial and error, by knocking on a few doors, asking a few people, following signs on the walls of the hospital and probably by checking several sections in a row at the supermarket. This is, understandably, a difficult and time-consuming process. Even more so when you are blind and cannot see people to ask, doors to knock on, signs on the wall or products at the mall. Under such circumstances, having a navigation system to guide you in unfamiliar places would be desirable. In addition, indoor navigation is a long-known problem in robotics [3].

So, why haven't we adopted the satellite-based systems that we use outdoors to solve the problem of indoor navigation yet? That is because walls and the structures of the buildings interfere with the satellite signals, rendering them inaccurate and unreliable for positioning indoors. Besides, that system would not have knowledge about the organization of a building or the ever-changing obstacles we may run into while walking indoor environments. There have, however, been attempts at solving the indoor navigation problem with approaches that use Bluetooth [4] or infrared [5] beacons, RFID tags [6], rely on Wi-Fi positioning [6] or WLAN connections [7], but because of their weaknesses that involve the requirement of placing tags/beacons, mapping of spaces with separate technology while navigating using routers and giving feedback using yet another device, expenses of setting up and limitations of use, none of these solutions have become a standard for indoor navigation.

These weaknesses can be overcome by using motion tracking and depth perception sensors. The depth perception sensor that we use is an infra-red sensor that estimates the distance/depth of objects from the device by projecting waves in the scene. On the other hand, the motion tracking sensors that we make use of are inertial sensors: accelerometers and gyroscopes that track the user's movement relative to his surrounding area in a 3D model of the indoor environment that was previously mapped using the same sensors. Using these sensors on board hand-held devices has the advantage of the system depending only on the sensors/device, costs of the sensors are included in the price of the device itself and the device can be used in any indoor environment, independent on whether there are tags, beacons, Wi-Fi sources or any other technology.

Currently, a device with such sensors on board, which we are using in this work, is a tablet by Google called Project Tango. In addition, Project Tango has on board processing (NVidia Tegra K1 with 192 CUDA cores) and 128GB of internal storage, next to 4GB of RAM. Previously, Microsoft's Kinect has been used for its similar sensors in Simultaneous Localization and Mapping (SLAM) (see Chapter 2). Kinect differs from Project Tango largely in the target audience. Kinect does not incorporate any storage and processing units, and it is treated primarily as auxiliary input device for games. Project Tango is designed to be a general purpose tablet, with enhanced sensing capabilities. In principle, any device with an infra-red depth sensor and inertial sensors – accelerometers, gyroscopes – (preferably with on-board storage and processing) can be used for the purpose of indoor navigation in the same fashion as we suggest in this report.

An indoor navigation system, just like any other navigation system, should be able to complete three tasks:

- localize the users in the space they are in,
- localize the destination chosen by them in that space and
- guide the users to the destination from the current position.

Since we also take into account blind users and robots we add another demand to our system: it should guide users to the destination without colliding with obstacles along the way.

In order to achieve these tasks we need to have:

- A map of the space in which the user and the chosen destination are to be localized,
- A navigation algorithm that takes the user from start to destination,
- An obstacle avoidance algorithm that avoids collision with obstacles during traversal.

The navigation algorithm depends on the map to find the route, but the obstacle avoidance algorithm can also be used as an independent component to avoid collision in unknown environments.

We can use motion tracking and depth perception sensors together with the device's camera to:

- Create a 3D map of the environment in which we want to be able to navigate (by scanning the environment with depth perception + camera + motion tracking),
- Locate the user in the map (depth perception + motion tracking) and
- Avoid collision with obstacles (depth only).

The destination chosen by the user can be found computationally from a set of possible destinations which are labelled in the map, while the navigation algorithm is going to estimate the path based on the map – no input from the sensors is needed to compute the path, but will be to localize the users within the map and track their movement along

the suggested path.

Previous research has focused on small parts of indoor navigation with motion tracking and depth perception. The issue with the related work is that whenever an approach or algorithm is proposed as a solution to one of the smaller problems, a lot of things are assumed as existing and serve as a basis to the new research. For example, in many approaches proposed for achieving obstacle avoidance, obstacle modelling is assumed as completed beforehand. The problem with this is that whoever wants to implement and experiment with an proposed approach will fail to see on what exactly that approach is dependent and all the work they will have to implement and test first in order to be able to come to the implementation of the suggestions. Sometimes the dependencies on other parts are so big that no time will be left for the implementation of the actual proposal. In the example of obstacle avoidance, to be able to actually avoid obstacles in real time we need to find a way to deal with the large amount of data coming from the sensors, with modelling obstacles from these data so that all computations are completed before the next frame to avoid lag, while still having a good representation of the obstacle, and only then move on to obstacle avoidance which again may call for separate implementations of obstacle avoidance for moving obstacles and static ones. Assuming too much – taking too much for granted – makes even excellent works less practical. Therefore, here we intend to look closely at all the details that have to be implemented in order to achieve a navigation system that guides the user in an unknown space with obstacles of arbitrary shape using depth and motion tracking sensors and only then propose our approach to obstacle avoidance, a smaller part of the system. We also look at how the work of others fits into the big picture of indoor navigation with motion tracking and depth perception. Without the detailed workflow that we are providing here anyone who endeavours to create an indoor navigation system with motion tracking and depth perception will have to go through the same steps that we did before being able to implement it or will be stopped at the first set of unexpected issues.

Research questions

The goal of the indoor navigation system that we research and design here is to guide a user through a collision-free path from start to destination while achieving shortest path and real-time guidance. The questions that gain the most focus in this thesis are:

- How well has the problem of indoor navigation with motion tracking and depth perception (either using a depth sensor or computer vision techniques) been solved?
- How can an indoor navigation system based on depth perception and motion tracking be achieved? What are the necessary elements and how they are combined?
- Can real-time obstacle avoidance be achieved from point-clouds provided by the depth sensor?

We aim to:

- Review the related work on localization in 3D space, indoor environment 3D modelling and indoor navigation with depth perception and motion tracking (Chapter 2).
- Dissect indoor navigation with motion tracking and depth perception to its fundamental parts and look into the technical capabilities and inner workings of Project

Tango and its sensors (Chapter 3).

- Present a complete view of the issues and challenges related to each of the core parts of the system and analyze what needs to be achieved in order to implement each component (Chapter 4).
- Propose our approach to indoor obstacle avoidance based on depth perception (Chapter 5).
- Summarize and discuss our findings (Chapter 6).
- Draw conclusions and establish necessary tasks for future work (Chapter 7).

Keywords

Depth Sensors, 3D Mapping, Tracking, Indoor Navigation, SLAM, Project Tango

2 Background

2.1 Related Work

Project Tango is a development kit created by Google's Advanced Technology and Projects (ATAP) group [8] and first released in 2014. It incorporates motion tracking sensors – six degrees of freedom (6DOF) accelerometer, gyroscope and compass. It also contains an infra-red time-of-flight (TOF) 3D depth sensor [9]. Due to the novelty of the device there is limited research on it to date. There is, however, comparable research on Microsoft's Kinect, a device that has already had the infra-red based 3D depth perception sensor and a 3-axis accelerometer incorporated [10]. Researchers have made use of the Kinect to implement the Simultaneous Localization and Mapping (SLAM) algorithm. Since we work with localization and mapping with Project Tango, here we look at similar work that has been done using the sensors of Kinect and some issues and challenges related to their work. Also, we look at integration of mobile device sensors for tracking purposes. The related work on path planning and obstacle avoidance is discussed separately in Chapter 4 since it is closer related to our work in the proposed algorithm.

The Simultaneous Localization and Mapping (SLAM) algorithm works with computer vision techniques using feature matching and tracking algorithms and/or depth perception sensors, to simultaneously map the environment and track the device's position relative to the surrounding environment. Based on the change in depth from frame to frame, it can localize the device in the space it is in.

Previously, expensive laser scanning systems have been used to map indoor environments in order to use them for indoor navigation. Although they are more precise than Kinect in depth sensing [11], have a wider field of view (180 degrees compared to the 58 degrees of Kinect) and operate at a wider range of distances, they are expensive ($2000) and only scan in 2D [12]. With the Kinect becoming available at consumer price ($109.99 [13]), research has taken advantage of its sensors to implement the SLAM algorithm. Kinect was released for gaming purposes; to capture motions of people and objects and use them in gameplay. Using the RGB camera and the depth information gathered from the depth sensor, depth information is extracted and presented in a form of RGB-D data, where RGB represent the individual color component of the perceived pixels, and D represents a 3-dimensional array of coordinates of those pixels.

Most of the research in the field focuses on SLAM for robotics applications. Typically, Kinect is placed on top of a robot together with a computer that will collect the data and the data is then processed by another nearby computer after the data is streamed through a wireless connection. Kinect does not have any storage or processing units of its own, and thus has to stream the data to a host device, typically a PC, and rely on the host to process the data.

El-laithy et al. [11] give insights into the capabilities of Kinect for robotics applications. From experiments with the device they concluded that Kinect generates depth data with tolerance of plus or minus 1 cm for distances 80cm to 4m and does not generate any depth data for smaller or bigger distances; the depth sensor has problems with reflect-

ive surfaces, making them appear in front of the device while in reality they are behind the rest of the objects; it does not work well with transparent surfaces such as glass or transparent plastic, often not registering their presence in the scene at all. The depth sensor also has problems with direct light because infra-red sources interfere easily with the infra-red stream of the sensor, but the authors demonstrate that it can be used also for outdoor environments during sunset. The paper suggest that new libraries need to be programmed to calibrate Kinect to detect still objects and not just moving people as Kinect is calibrated by default to.

Oliver et al. [12] also used Kinect in a similar setting, on top of a robot, but experimented with both 2D and 3D SLAM. In the case of 2D SLAM, Kinect was compared to the performance of the Hokuyo laser scanner and the results show that: The laser scanner has a ten times higher depth resolution, which helps with feature matching; it has three times higher field-of-view, which allows it to map spaces faster, and the minimum operating distance of Kinect is ten times higher, which makes it difficult to work with features that are in a close range. Nonetheless, the problem with laser scanning systems is that they only scan in a plane, which means that instead of registering the entire object or a big part of it, they register only a slice, e.g. the laser scanner only 'sees' the legs of the table, instead of the entire table. On the other hand, for the 3D SLAM, the authors show that Kinect correctly represents most of the environment, but also a lot of noise is included and the main source for the error seem to be the above-mentioned limited field-of-view, range and horizontal resolution.

Manap et al. [14] use Kinect for object detection, for two types of objects: shiny and curve-shaped objects and dull and flat objects. They conclude that the sensors in Kinect work best at distances between 4 and 5 feet (1.2-1.5m) and also that the objects with curved, shiny surfaces are incorrectly detected because the reflection of the light makes the sensors wrongly interpret the position of the object. Tests have been done only with one object of each category and cannot be considered generalizable to all situations. Moreover, other types of objects such as flat with shiny surface and curved with dull surfaces have not been tested.

Kamarundin et al. [1], use Kinect to test the feasibility of its use for indoor navigation with a mobile robot. They report the biggest problem with laser scanner robots to be their possibility of misinterpreting the position and distance from the sensor due to the fact that they scan only in 2D. Figure 1 shows how the distance from one object can be misinterpreted by the laser scanner which may cause collision when the robot navigates that space later. For the experimentation, the common setup is to use a 'base station' computer (more powerful), connected to a smaller computer and a Kinect on a mobile robot. The authors report problems with reflectance of light from object surfaces, making the objects appear closer than they really are and that transparent surfaces are not registered because the infra-red waves can travel through them. The work focuses on converting the 3D data gathered from Kinect to a 2D map for indoor SLAM for robots. A robot scans a room and after the data is gathered, it is processed to exclude information 26 centimeters below the sensor and higher than the height of the robot since they are either part of the floor or so high they cannot be considered as obstacles. Next, the nearest objects/obstacles are projected on a 2D map – this decreases the amount of data that needs to be processed and it is efficient since the robot at any given time only needs to know the nearest obstacles and when those are overcome, the next set of nearest

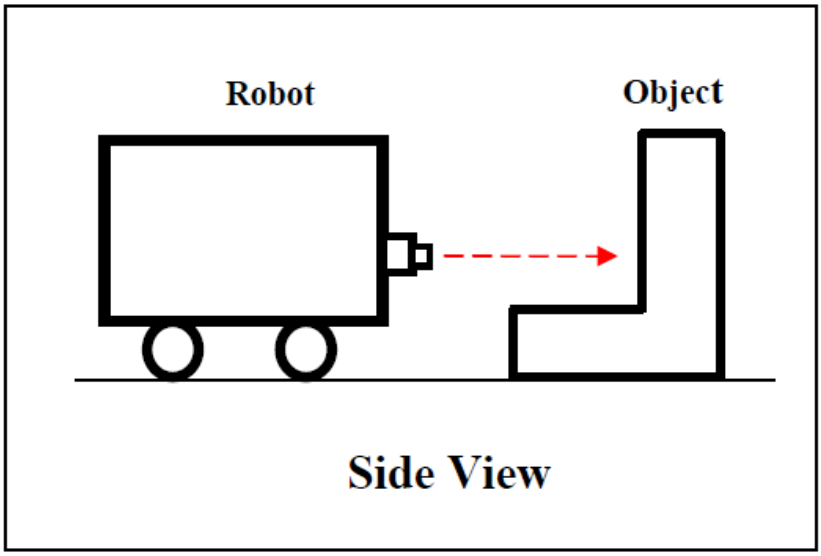

Figure 1: Robot with 2D scanner mistakenly determines the location of the object [1]

obstacles is mapped. The results prove that the conversion method has been able to map objects that were potentially missed by 1D and 2D scanners. The authors also suggest the use of an additional ultrasonic sensor together with Kinect in order to detect the transparent and high reflection surfaces since sound waves do not get distorted from direct light and also provide a better resolution for short-ranged vision.

Similarly, Ghani et al. [15] show that using the Kinect they can construct richer 2D maps for indoor navigation for robots compared to the maps generated by laser scanning systems because of the same reasons as [1] reported. The implemented SLAM algorithm is tested with simulated and real indoor environments and results show that real-environment based maps are much noisier than for the simulated ones. Nonetheless, they are more accurate at including obstacles than the laser scanning system used for comparison. One major problem that is reported is the difficulty of performing SLAM in environments with moving instances, such as people, in front of the robot. They propose the development of an algorithm for removing and updating dynamic objects on the 2D map as future work.

Shashkov et al. [16] make use of Kinect to scan 3D scenes in order to use them for game development. The authors use Microsoft's KinectFusion [17] for this purpose, a SLAM algorithm Microsoft developed to work with Kinect. Although they consider the results to be "generally appealing", they find limitations in using this algorithm. Those are the need to remain in the immediate area (reported optimal distance 1.8 m), the fact that the algorithm prevents you from moving in other areas, that it is viable only indoors because of infra-red interference and that the result is overly complex, representing even the simplest walls and surfaces with a lot of information and edges. As an alternative, the authors used the algorithm to scan different areas close to each other, constructed meshes using the gathered data and stitched the meshes together manually. They also report that other open-source implementations, such as the spatially extended version of KinectFusion, Kintinious [18], do not give satisfactory results for scanning of larger indoor spaces.

Du et al. [19] are concerned with the fact that Kinect sensors are generating too much data, many times representing some areas with too dense information, while others do not get covered at all. They suggest to make use of user input to help the mapping of indoor spaces by finding areas that have not been mapped well, and moving the user away from those that have dense representations already. The results of testing their system with 4 users, although small to draw conclusions from, strongly suggest that interactive mapping makes the mapping process robust and error tolerant.

The work of Pinto et al. [2] is the only work so far that actually implements trajectories to suggest the robots to use when moving in an indoor environment. However, the trajectories are very simple and do not seem to be generated from the same standing point, as can be seen in Figure 2. They tell the robot how to move about without colliding with obstacles. In the Figure, green trajectories are feasible trajectories, while the red ones infeasible because they make the robot run into obstacles.

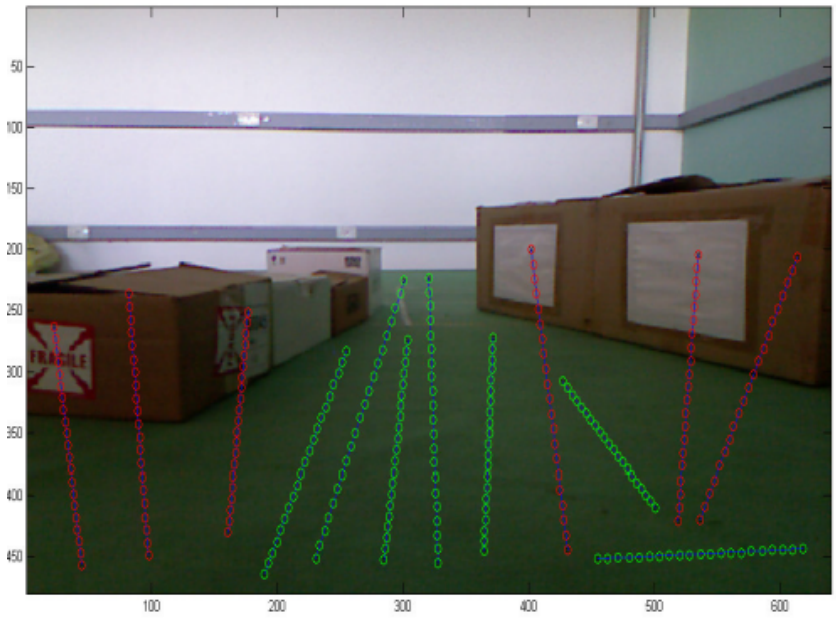

Figure 2: Trajectories suggested to the robot by the algorithm developed by Pinto et al. [2]

Schöps et al. [20] create a system for 3D modelling using Project Tango based on the data from the fish-eye camera of the device. The camera poses are computed with the combination of feature tracking based on the fish-eye images and inertial measurements from the Inertial Measurement Unit (IMU) sensors (accelerometer, gyroscope) in a Kalman filter. From the fish-eye data the depth information is calculated. This allowed the authors to model outdoor environments as well because the system is not dependent on the infra-red sensor that gets affected from direct sunlight and has a limited range. The system allows user input when not enough data about a part of the scene is gathered.

Wagner et al. [21] presents a method for vehicle positioning inside parking spaces where Global Navigation Satellite System (GNSS) is not available, using dead reckoning (geometrical data such as distance and angle) and digital maps of the environments. Since it makes use of both geometrical data and link connectivity, it is considered a

topological map matching algorithm. The map-generation data was gathered by vehicle inertial measurement units (accelerometers, gyroscopes and magnetometers) and then manually extended to include several floors and individual parking spots or the maps were entirely generated by building plans and reference measurements. The algorithm turns out to be quite error prone, giving errors of up to 20 meters at the exit of the garage when navigating inside the parking space for 10 minutes in 5 level changes. This level of error might be acceptable in the case of vehicle navigation when the driver is present, but is not feasible for indoor navigation of humans as an error of 20 meters might take the individual to a completely unintended place. The algorithm does not consider that a parking spot might be taken, and so does not change the map from what it was when the space was mapped. It does not consider navigation without running into obstacles either, probably because the authors seem to assume an always-present driver. The authors had also worked in employing Wireless LAN for positioning inside the garage, but they noted that the availability or absence of cars in the garage affected the accuracy of positioning immensely and the absolute positions of WLAN systems could not be used to improve the accuracy. Another issue with WLAN is that the availability of access points in parking spaces is not expected in a number large enough to allow positioning.

The work of Li et al. [22] is an example of tracking using smartphone sensors and Wi-Fi fingerprints rather than of navigation. While the authors treat tracking as navigation, it is not really correct, because the developed algorithm only registers where the user is walking and does not guide from one position to another. Correctly, it is simply position tracking.

Jayakody et al. [23] proposes a "navigation system" that is based on crowdsourcing data to label elements in a map. The work does not design or implement the navigation algorithm, but gives guidelines on how it could be achieved, using the crowd to label a topological map. The paper focuses on position tracking, rather than navigating a person through the obstacles to a certain position. Authors suggest the use of voice to guide blind people through the map.

Li et al. [24] is yet another example of the use of smartphone sensors, Wi-Fi fingerprints and magnetic matching for tracking with Pedestrian Dead Reckoning (PDR), not navigation in the true sense. Nonetheless, the work offers a few valuable points to our work:

- Point against Wi-Fi positioning: Wi-Fi based systems require creating and maintaining Wi-Fi networks; most methods for building Received Signal Strength Indicator (RSSI) databases require a large number of reference points and extensive recalibration, making the process tedious and labor intensive; non-stationary radio maps reflected as differences in measurements of the signal strength in the same spot during offline and online phases, induced by the dynamics of the environment such as the presence of people, elevators or moving doors.
- Shortcomings of inertial sensors and correction: internal sensors provide short-term accuracy, but in the long term, their accuracy is degraded. Errors of vertical attitudes (pitch and roll) can be controlled by measurements of the accelerometer, but the error in heading will accumulate over time. To solve this problem the magnetometer may be helpful, but its accuracy is affected by the presence of magnetic fields other than

the Earth's.

- Sensor accuracy: 1% of distance walked when mounted on foot, 4-8% when there is no requirement of location and altitude of the device on the body.

Michel et al. [25] give a comparison of six algorithms developed for improved attitude estimation in the case of pedestrian navigation. The work discusses how the gyroscope, magnetometer and accelerometer are affected from the way the devices are carried and the effect of magnetic deviation has on them. Then the authors go about in comparing different solutions offered by six algorithms developed for the purpose. Valuable outcomes:

- Magnetic deviations present in the environment (speakers, magnets, walls, floors, elevators, belts, keys, etc.) heavily affect smartphone's sensors, thus affecting the precision of attitude estimation algorithms in pedestrian navigation. E.g. the speaker on the smartphone has a magnetic field 30 times stronger than Earth's magnetic field.
- The gyroscope is usually not enough for attitude estimation so the accelerometer and magnetometer are used to correct the drift of the gyroscope. The accelerometer corrects pitch and roll angles while the magnetometer corrects the yaw. The combination of data from both inertial and magnetic sensors then becomes a problem for the algorithms that integrate them.
- The paper gives error models for drift and sensor bias for each of the sensors.
- External acceleration of the device is not negligible in attitude estimation.
- Quaternions are used in conversions because the Euler angles have the gimbal lock problem.
- The Android API offers quaternions and a "black-box" calibration algorithm which is shown to be quite efficient in data calibration. It is the algorithm the least affected from a magnetic field 5.5 stronger than the field for free space, created by surrounding the device used with 6 magnets in static position.
- The comparison demonstrates that uncalibrated data is useless for the purpose of attitude estimation since it deviates a lot from the ground truth because of the presence of the strong magnetic field of the speaker close to the IMU.
- Estimation becomes more difficult the more acceleration deviates from g, making attitudes such as walking with a swinging hand and walking with a phone in the back pocket difficult to estimate.
- The authors make a comparison on the performance of each of six algorithms for a set of motions, a comparison between the motions themselves, show the importance of calibration and correction of magnetic deviation as well as give a comparison of processing time for each algorithm as saving battery life is of high importance in smartphones. In case of battery life the Madwick [26] filter outperforms the others.
- Authors conclude that the algorithms by Martin [27] and Fourati [28] work the best algorithms for integrating sensor data for the purpose of attitude estimation.

Klingensmith et al. [29] present a real-time, house-scale, dense indoor mapping al-

gorithm using Project Tango. The algorithm does not rely on GPU. The research was completed as part of Google's Advanced Projects and Technologies (ATAP) for Project Tango.

- Experimentally found that most (93%) of the space in a typical indoor scene is actually empty. Iterating over these spaces wastes computation and storage.
- Compared to other mobile device based solutions for indoor mapping, mapping using Tango does not require the costly extraction of depth data from monocular stereo since the device has an integrated depth sensor. This saves processing and memory resources for the reconstruction only.
- Truncated Signed Distance Field (TSDF) stores a voxelization of the signed distance field of the scene. TSDF is negative outside objects and positive inside them, the surface being the zero isocontour. It takes more memory than occupancy grids but creates higher quality reconstructions by preserving local structure (that the point clouds didn't). "Distance fields provide useful information to robots: the distance to the nearest obstacle, and the gradient direction to take them away from obstacles."
- Point against Kinect Fusion: It is not suitable for the reconstruction of large scenes because of memory issues created as a result of the use of single fixed-size 3D voxel grid (similar to occupancy grid mapping).
- Tango's depth sensor is significantly noisier than the rest of the available depth sensors.
- Limitations: The algorithm cannot guarantee global map consistency and drifts over time. Needs to include SLAM techniques to combine visual inertial odometry and sparse landmark localization to allow real-time relocalization and loop closure on a mobile device.

2.2 Advantages and disadvantages of indoor navigation with motion tracking and depth perception

From the review above we noted that some of the major issues with motion tracking and depth perception sensors are: the limited range of view of the depth sensor, the interference of the depth sensor with infra-red sources of light and heat and problems with sensing reflective surfaces and transparent objects, the need to have a robust fusion approach of motion tracking sensors in order to improve the accuracy of tracking, the negative effect magnetic fields and the presence of people has on the motion tracking sensors and that it requires a great amount of work to achieve mapping with these sensors since the depth sensor of Project Tango is noisier than other depth perception sensors and while it densely senses some parts of the environment, it completely misses out on others. But there are also promising results that show that navigation with motion tracking and depth perception has advantages over other technologies such as laser scanning systems and Wi-Fi positioning. The results of Klingensmith et al. [29] showing that 93% of indoor environments is empty space is promising in regards to obstacle avoidance because obstacle avoidance becomes much simpler when there are few or no obstacles along the path to the destination.

In this section we compare indoor navigation with motion tracking and depth per-

ception to indoor navigation systems that use other technologies such as Wi-Fi/WLAN, Bluetooth or infra-red beacons, RFID tags, laser scanning systems and cameras with computer vision techniques. The major issues with indoor navigation with the latter set of technologies are:

- Wi-Fi/WLAN-based solutions depend on Wi-Fi/WLAN networks which are external to the device used by user; WLAN may not always be present.
- Beacon/RFID-based indoor navigation depends on the placement of beacons, which are as present as we want them to be. But who will place them? The users can certainly not go around placing tags and beacons themselves.
- Wi-Fi/WLAN-based solutions require setting up and maintaining of routers; availability depends on the availability of the network; price depends on setting up/maintaining these routers and the databases of the router fingerprints, as well as the additional device that will be used for navigation based on Wi-Fi signals. Also, in order to map the spaces in which navigation is to take place, another device that achieves localization and mapping is required, increasing the costs of using such system even further.
- Beacon/RFID-based indoor navigation: Cost depends on the sum of the price for each beacon/tag, the reader device and the mapping device.
- With Wi-Fi/WLAN based systems it is not always the case that fingerprints identify correctly the floor in which the user is. The cause for this is the interference with routers from the floor above and below the current floor.
- The presence or absence of people or objects in the scene affect the accuracy of Wi-Fi/WLAN positioning.
- Laser-based indoor navigation: very expensive, only scan in 2D which will result in incorrect perception of obstacles (see Figure 1).

Next we describe some of the important advantages and disadvantages of indoor navigation with motion tracking over the technologies we just discussed.

2.2.1 Advantages

- Depth perception and motion tracking sensors are on board of the device; they are ever-present as long as the device is present; the entire indoor navigation system can be completed with one device and the user can use the same (type of) device for navigation. A device with motion tracking and depth perception sensors is self-contained.
- Availability of the sensors: as long as the device is present and charged; fully dependent on the state of a single device.
- The price for the sensors is included in the price of the device (Kinect and Project Tango are available as consumer products).
- Fusion of sensor data helps the identification of the floor in which the user is in.
- Improved accuracy of obstacle recognition over laser scan systems.
- Obstacle avoidance with the infra-red sensor does not require prior knowledge of the

environment. For the full navigation, however, a 3D map of the indoor needs to be created, just like with the other technologies, but the user that only uses the map does not have to be familiar with the environment to be navigated in it.

- Using depth perception sensors is more feasible than using computer vision techniques with the camera alone because with the depth sensor depth does not have to be calculated from frame to frame, no feature matching and tracking is required, thus saving a lot of processing power for the rest of the computations.

2.2.2 Disadvantages

- The depth sensor has a short range of sight; works best between 0.8 and 4 meters and does not see beyond.
- Motion tracking sensors need to be effectively fused for drift correction, otherwise drift will accumulate and cause the system to suffer from errors in localization and tracking.
- Depth sensor: infra-red is easily influenced by other infra-red sources such as sources of light and heat which disorient the waves and thus the sensor most of the time fails to register the waves that have been casted by the infra-red projector.
- The depth sensor does not work well with transparent and reflective surfaces, not registering the first and wrongly registering the second. Dark surfaces absorb infra-red waves making them less likely to get recognized.
- The depth sensor/receiver must have direct line of sight; the receiver cannot be covered or put in the pocket.
- While computer vision techniques can be used unchanged in outdoor and indoor environments, the infra-red depth sensor works only indoors and possibly outdoors in low levels of sunlight, e.g. during sunsets.
- The depth sensor is noisier than the rest of available depth sensors.

It is clear that besides disadvantages, motion tracking and depth perception sensors make good candidates for use in indoor navigation purposes.

From the review of the related work, we have noted that while there are works that solve problems related to indoor navigation, it is not clear how exactly these solutions contribute in solving the bigger problem of indoor navigation. In the following chapters we identify challenges that indoor navigation with motion tracking and depth perception presents and review more related work that treats these challenges. In this way we create a clear workflow for the indoor navigation system and make it clear how solving each challenge contributes to solving the indoor navigation problem.

3 Indoor Navigation

Let us look a little closer at the indoor navigation problem and the inner workings of the Project Tango device and the depth perception and motion tracking sensors.

In order to be able to achieve indoor navigation, some fundamental parts are needed. First, we need to have a representation of the environment in which we want to navigate. Just like Google Maps can navigate us outdoors only by using the already available maps of the surrounding area, we need a representation of the building in which we want to offer navigation. Second, we need an algorithm that can take the user from the starting point to the chosen destination within the representation. To compare it to Google Maps again, this would be the algorithm that guides you to the chosen destination by using turn-by-turn navigation. Third, taking into account that the user may be a blind person or a robot, we consider path planning without running into obstacles. Since the environment has changed since the representation was created – different objects in the scene may have changed their position (chairs and tables have been moved and suddenly there is a coffee machine in the room) – the navigation algorithm needs to take into account the online data that it receives from the depth sensor during navigation in order to locate these obstacles and plan the path around them. The three components of the designed indoor navigation system are depicted in Figure 3.

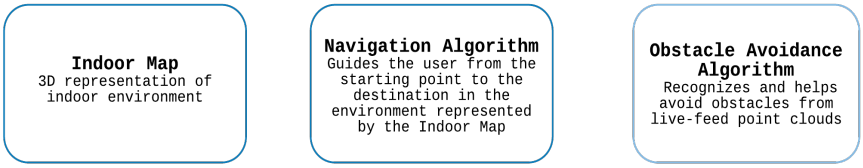

Figure 3: Indoor navigation system components

The Navigation Algorithm uses the Indoor Map to compute the shortest path from start to destination. In order to guide the user along that path, the Navigation Algorithm estimates the position of the user in the Indoor Map by comparing the surroundings of the user (scanned by the device) to the 3D representation of the environment. To achieve this, the user is required to scan the area with the depth sensor and the camera. After the initial position is estimated, the navigation starts and by using motion tracking sensors the position of the user relative to the starting point is tracked within the map and compared to the generated path. The Obstacle Avoidance Algorithm works with the Navigation Algorithm to replan the path when obstacles are presented along the path suggested by the Navigation Algorithm. The task of the navigation system is to produce a collision-free path from starting point to the destination chosen by the user in the map of the indoor environment.

Figure 4 demonstrates a flowchart of the a complete indoor navigation system. First, the user opens the map of the indoor environment in which s/he is about to enter and

chooses a destination where s/he wants to go. In the background, the application will locate the user inside the map and then use the navigation algorithm to compute the shortest path to the chosen destination based on the map. Then, the user gets a chance to decide whether he wants to enable obstacle avoidance and thus incorporate depth sensor data to the navigation algorithm. If the user decides that that is not necessary, the algorithm will offer turn-by-turn navigation, leaving obstacle avoidance to the user. At this point the depth sensor can be turned of to save battery as it will be no longer in use. If, however, the user wants to enable obstacle avoidance, the depth sensor is kept on and the depth live-feed is processed. From the processed data the user is notified of the immediate surroundings, of their distance and their coarse dimensions. The user is told to walk straight until s/he approaches an obstacle or turn when s/he is guided to turn to avoid the obstacle or to follow the computed shortest path. During the entire time, both with or without obstacle avoidance, the user's position is tracked within the map and input to the navigation algorithm. When the user's position matches the destination chosen in the map, the user is notified and the navigation process is terminated.

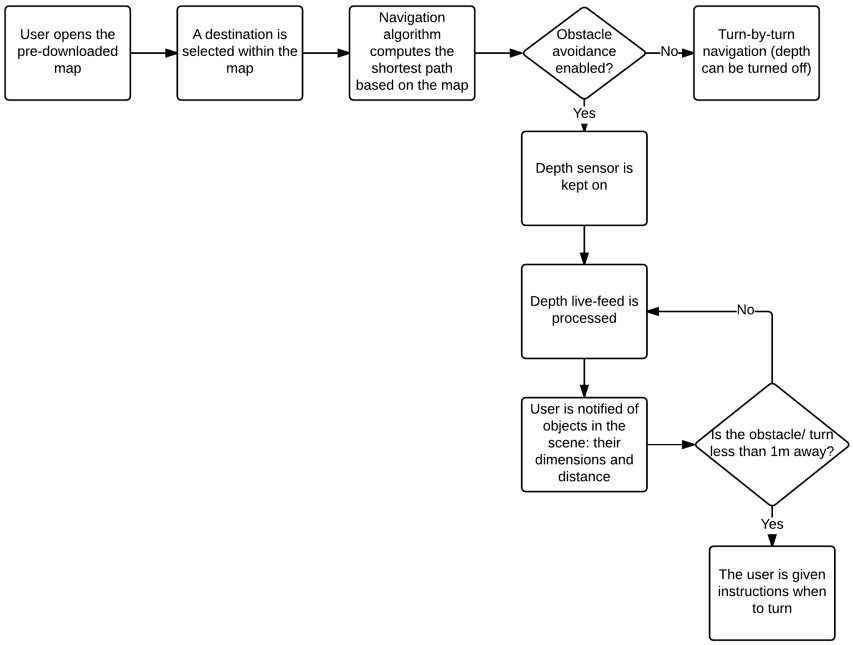

Figure 4: Indoor navigation workflow

Note that if the indoor environment has changed in major ways since the time it was mapped, e.g. new walls have been built or entrances have been permanently closed, and the map has not been updated in the mean time, the navigation algorithm will not be able to compute the true shortest path as it can compute the path only from the outdated map.

3.1 Device technical specifications and inner workings

Project Tango is a device developed by Google's Advanced Technology and Projects (ATAP) group with the intention of adding human-like sensors to technology. This in-

volves the motion tracking and depth perception sensors that allow the device to locate itself in space and compute its distance from objects that are in the range of sight of these sensors [8].

The device offers a 4 megapixel RGB-IR rear-facing camera along with a 170 degrees fish-eye motion tracking camera that updates at 30Hz. The infra-red projector is also positioned in the rear of the device and has a refresh rate of 3Hz. The accelerometer, gyroscope and compass offer six degree-of-freedom positional data.

Different from Kinect which included similar sensors [10], Project Tango, just like other Android devices, offers on-board processing and storage. It has a 2.3 GHz quad-core NVIDIA Tegra K1 CPU and also offers 4 GB of RAM and 128 GB internal storage, which is expandable via microSD [9] [29] [30]. At the time of writing, the devices is running Android 4.4 KitKat. Figure 5 depicts a Project Tango device with its sensors.

Figure 5: Project Tango

Project Tango offers three Application Programming Interfaces (APIs) [31]: The Java API, C API and the Unity API. Unity[1] is a game engine and the Unity API is the only one of the three that allows us to work with and edit meshes of the scanned environments and also has the benefit of offering its own navigation (A* algorithm) and pathfinding algorithms that makes it simple to bring navigation and pathfinding to games. The A* algorithm [32] computes the optimal path between starting point and destination for the generated 3D map divided into cells of a grid. We use Unity 5 in this project.

Our idea is to first scan indoor environments to create meshes as representations of the environment (Indoor Map) and then use these meshes in Unity to compute the shortest A* path (Navigation Algorithm). Obstacle Avoidance is then integrated with the A* path using the live feed gathered from the depth sensor. To achieve each of the components means to successfully complete many complex smaller parts and solve several issues that come up. In an attempt to attain such an indoor navigation system, we were able to identify major difficulties related to each of the components. These issues are described in detail in the next chapter.

[1] https://unity3d.com

3.1.1 Motion tracking

Motion tracking in Project Tango applications is based on visual and inertial odometry. This means that computer vision techniques are combined with measurements from the inertial sensors to achieve mapping of the environment and positioning of the device within an indoor environment relative to the objects in the scene and to track the device in the given space. This process is known as Simultaneous Localization and Mapping. Computer vision techniques involve feature matching – a technique used to match features (objects or part of them) between frames – and feature tracking, a technique which is used to track the matched features in order to determine the position of the camera/device relative to those features. In a similar manner, after the position of the device is established, the speed with which it is moving can be determined based on the change of position between frames. The Tango APIs provide the pose of the device, i.e. position and orientation, in six degrees of freedom (6DOF) from the inertial sensors: accelerometer, gyroscope and compass. The return data is a 3D vector in meters for translation and a quaternion for rotation [33]. Both sources need to be fused in order to achieve tracking, but Google is not offering insights how they achieve data fusion for tracking purposes. Figure 6 shows an illustration of the device being tracked in an indoor environment.

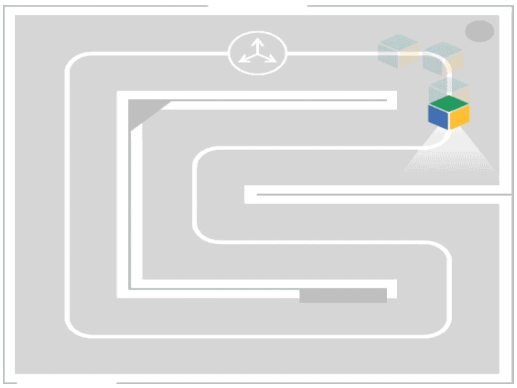

Figure 6: Motion tracking[2]

Motion tracking on Project Tango has some limitations. Two of the most important ones are [33]:

- Drifts: after walking long distances for a period of time error is accumulated and the perceived position from the device will have drifted from the actual position in which the user is in.
- Motion tracking does not understand/register the space around it. Every time a new tracking session starts, it will be based on the latest starting position, not remembering anything from previous tracking sessions.

Both limitations are addressed with Area Learning which is another process that allows the device to "learn" the area around it while keeping track of the user's position in the given scene. The space in which the device is in is *recognized* by matching what the device currently sees with what it has seen before. For area learning it is easier to *re-*

[2]The image is a property of Google and can be found at: `https://www.google.com/atap/project-tango/about-project-tango/`

cognize spaces that are visually interesting – with corners and edges – than empty rooms with white walls. It also depends on the conditions under which the environment is being registered – day/night and furniture changes. Walking around the area with area learning on will allow the device to create an improved model of the area and thus correct drift. Area descriptions are saved in compressed form in Area Description Files (ADF) and can be reloaded whenever area learning is to be continued [34].

Although Tango uses motion tracking with both visual and inertial sensors in their applications, since we are using the depth sensor, in this project we combine depth and inertial motion tracking sensors, leaving out vision. Getting distance with the depth sensor is simpler because the device gets the distances immediately without having to compute them from frame to frame, thus saving a lot of computation power for the other processes. The limitation of using the depth sensor instead of vision is that the application cannot be used for outdoor environments, but since we are concerned only with indoor navigation here, that factor will not restrict our work.

3.1.2 Depth perception

Depth perception allows the device to find the distance from objects in the scene. Project Tango allows three ways of achieving depth perception: Structured Light, Time-of-Flight and Stereo. Structured Light and Time-of-Flight use the infra-red sensor to estimate the depth, while Stereo calculates the distance based on the pictures taken from two cameras (similar to the human eyes). For our purposes, we use the Time-of-Flight depth with the infra-red sensor. In Time-of-Flight the distance from the objects will be estimated from the time it takes the infra-red waves to be reflected back and received at the infra-red sensor [35].

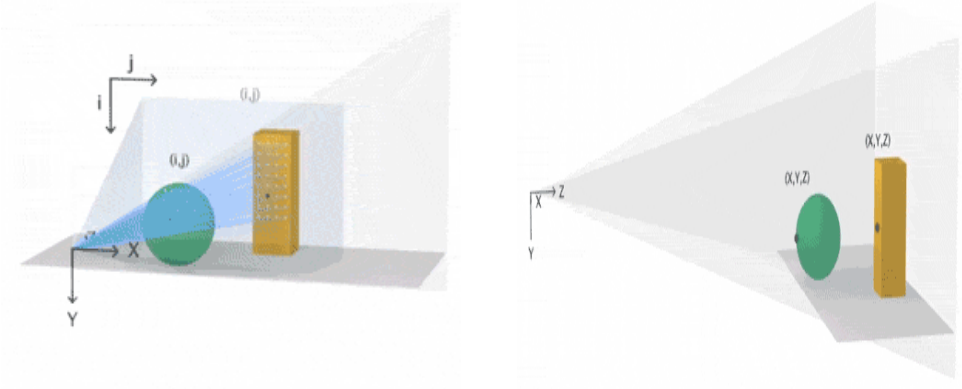

Figure 7: Depth perception[3]

The depth APIs of Project Tango return point clouds from the depth sensor in the form of XYZ coordinates as float values given in meter. If the device is in landscape orientation with the screen facing the user, then +Z points in the direction of the camera's optical axis

[3]The image is a property of Google and can be found at: `https://www.google.com/atap/project-tango/about-project-tango/`

and is perpendicular to the plane of the camera, +Y points to the bottom of the screen and +X points towards the user's right [36]. So, in fact, the API returns coordinates of objects in space relative to the device, not distance directly. The distance can be easily found by computing the Euclidean distance of two points in 3D space (Equation 3.1) knowing that the origin (0,0,0) is the focal center of the sensor.

$$d(P,Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.1}$$

since $P = (x, y, z)$ is any depth point from the point cloud and $Q = O = (0, 0, 0)$ the camera's focal center, we get Equation 3.2. Figure 7 shows how the depth projector casts infra-red waves onto the scene and by sensing the returned waves the depth sensor determines the positions (XYZ coordinate) of each depth point relative to the device.

$$d(P,O) = \sqrt{x^2 + y^2 + z^2} \tag{3.2}$$

We mentioned that sources of light and heat, which are infra-red sources, interfere with the depth sensor and that the sensor fails to detect transparent surfaces since the waves travel through them. Figure 8 shows how the device fails to get depth information on the window because it is transparent and the sunlight interferes with the sensor. The rest of the indoors that is not transparent is detected.

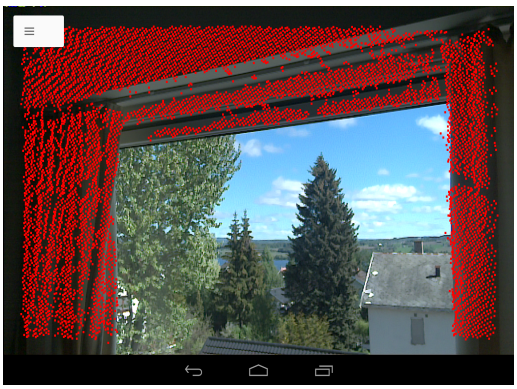

Figure 8: Sunlight interferes with the depth sensor and infra-red waves fail to detect transparent surfaces[4]

3.2 Ideal indoor navigation system with depth perception and motion tracking sensors

Now that we learned of Motion Tracking, Area Learning and Depth Perception, we are inclined to believe that the indoor navigation system that we are looking for can be constructed as following:

1. The 3D representation of the indoor environment is constructed with Area Learning.
2. The representation is used to compute the shortest path for any given starting point to destination within the representation.

[4]Achieved with Tango's Plane Fitting Android app. The application can be found here: `https://github.com/googlesamples/tango-examples-java/tree/master/java_plane_fitting_example`

3. Live Depth Perception is used with Area Learning to recognize the area that the user is in which will make it easy to locate and track the user from start to destination and to help avoid obstacles along the way.

Unfortunately, discrepancies between the files produced by Area Learning and indoor meshes that can be used with Unity to compute the shortest path make it impossible. Area Learning produces an Area Description File that contains descriptions of the objects in the scene and is a closed, internal format to which we do not get access. Hence they cannot be used with other software or converted to other file types that we can work with in order to compute the shortest path.

Also, Area Learning does not work well when the same indoor space is scanned in different environmental conditions such as different amounts of light/darkness in the room and it fails to recognize the same room when furniture is moved around. Ensuring the same conditions every time is a difficult task since indoor environments are prone to changes.

Even if the ideal case would work, we would not be able to guarantee that the A* algorithm would give the shortest path if the obstacles in the way of the user have moved since the first scan. This is due to the fact that the infra-red sensor has short range and is unable to see the whole scene at once and know the dimensions of the obstacles, in order to plan for a better path if the one based on the map is blocked.

In reality, each of the components from Figure 3 brings along its challenges and none can be implemented straight away, however promising and straightforward Depth Perception, Motion Tracking and Area Learning with Project Tango appear. Issues and questions that arise include:

1. Indoor Mapping:
   - Scanning of the area with the applications provided by Project Tango gives an .obj mesh file of the scanned environment. From experience we know that this file cannot be imported in Unity as is because it will not be recognized as a .obj file since standard components of the files seem to be missing when output from the applications.
   - Post-processing is required. Not only because the .obj meshes are not recognized by Unity in the previous step, but also because the models are incomplete, bumpy and with a lot of holes. This steps is time-consuming and requires 3D modelling skills. It involves hole-filling, smoothing, aligning and segmentation.
   - Big files are produced in the end. Some parts of the environment will be densely scanned while others scarcely so. How is the information from the densely scanned parts reduced and holes in the other parts filled?
   - Labelling of generated maps with additional information to create topological maps is necessary. Otherwise, the map by itself will not convey much information to the user.
   - Distribution of the end product to the user.
2. Navigation Algorithm
   - A major weakness of the A* algorithm is the big consumption of processing power whenever the path has to be recalculated. How do we overcome this issue when

the proposed path is blocked by an obstacle that is not in the map or the user does not follow the suggested path until the end and the map needs to be recalculated?

- How to integrate sensors in order to provide accurate positioning and tracking knowing that inertial sensors are prone to error accumulation and noise?
- How is information conveyed to the blind user? How often does an update need to be given?

3. Obstacle avoidance
   - This component registers depth point clouds while the user is navigating. How does the algorithm know what these point clouds represent? Objects in the scene need to be modelled from the depth points, which raises the question of what criteria we use to decide which point belongs to which object.
   - Since we want to keep obstacle avoidance real-time, how can we do object segmentation and obstacle avoidance efficiently?
   - The depth sensor has a limited range of view and because of that, when objects are not completely inside that range, they will appear to extend infinitely. How are "infinite" obstacles treated compared to finite ones?
   - Same as above: How are instructions given to the user and with what update rate? Should the update rate increase with decreasing distance from obstacles or turns or should it be kept constant?

These and other issues and questions that come up are discussed in depth in the upcoming chapter.

4 System Architecture

4.1 3D modelling of indoor environments

As has been previously mentioned, in order to be able to navigate in any environment, we need a representation or a map of that environment. Since our problem deals with navigation without running into obstacles, and not only point-to-point navigation, we need to have a 3D representation of the environment from which we can see the inner structure of the buildings in which we want to navigate. To create these representations, the environments need to be scanned beforehand and a mesh thereof needs to be constructed. Applications that are available on Project Tango (Project Tango Constructor [37] developed by the Project Tango team, Room Scanner[1] version 9.5 and Voxxlr[2] version 2.4) can be used to construct such models, but unfortunately, so far, none of these applications offers an ultimate end product that can be used as a model right away. Figure 9 shows an indoor environment mapped with the designated Project Tango Constructor Android application (current version: 1.1.2) [37]. The application uses depth perception and color data from the camera to construct 3D meshes (.obj files) [35]. We can see from the mesh that there are a lot of holes present, that the data is quite noisy and not always accurate (notice the change in the yellow line in the lower left corner of the picture which in reality is a straight line, but here it is not depicted as such).

Figure 9: 3D represention of a room mapped using Project Tango Constructor

We have extensively used these applications in mapping several indoor environments, but in our experience, the final models that are produced by them need a lot of post-processing that is a time-consuming process and requires experience with available 3D

[1] https://play.google.com/store/apps/details?id=com.BlueRing
[2] https://play.google.com/store/apps/details?id=com.voxxlr

modelling software such as MeshLab, Blender, Maya or Autodesk. The post-processing of generated 3D models involves mesh simplification, cleaning of the meshes, hole-filling (holes are presented in the model when particular parts have not been scanned or when the sensors registered invalid data when they were unable to extract information about them due to environmental factors) and smoothing. Furthermore, in order for the representation to be used with Unity's Navigation and Pathfinding algorithms, the mesh needs to be separated in several parts: the main part, that is considered to be the walkable surface, to which the NavMesh will be assigned in Unity and the rest will be each of the objects that we want to consider as obstacles, to which we will assign the NavMesh Obstacle component. Therefore, a further step or segmenting the floor and obstacles from the single mesh is required.

Commercial applications that offer 3D mesh *fixing* are available online, but they do not always work or are only black boxes to us, meaning that they do not offer any information on what will be fixed in the model or do not give the possibility of manipulating their settings. Two of such tools have been considered and tried in this project: Microsoft 3D Model Repair[3] and Sculpteo[4]. Both tools are designed to repair 3D models to be used for 3D printing.

Microsoft's Model Repair Service is a black box 3D model fixing tool; it works in three steps: 1. A 3D model needs to be uploaded, 2. The model will be (magically) fixed by the service and 3. The fixed model can then be downloaded, if the fixing step has not failed. With the meshes generated by Project Tango Constructor this service either completely fails to repair them, or gives a representation that seems to *package* the mesh. For reference, see Figure 10 where we tried to fix the mesh from Figure 9 with the Model Repair Service from Microsoft. We suspect that the latter case happens because the service expects us to print the model and thus creates a package that includes the model inside of it so that it can be more robust. For our purpose, however, this will not work as we do not get access to the contents of the package in Unity. On the other hand, Sculpteo seems to do some (unspecified) repairs but the model can only be ordered as a 3D print and not downloaded after the fix, so, again, it does not fulfills the need of our project. Sculpteo also works as a black box, only the thickness of parts in the model can be changed (so that the 3D print does not break).

[3] https://modelrepair.azurewebsites.net/
[4] http://www.sculpteo.com/en/

Figure 10: Microsoft's Model Repair Service either a) fails to repair models created with Tango Constructor or b) *packages* them

As discussed in the Background chapter, [20] and [29] are great examples of the amount of work and detail it requires to generate accurate and usable 3D models of indoor environment with, precisely, Project Tango. These works show that 3D modelling cannot be completed straight away with the current tools offered by Project Tango for indoor mapping and that instead sophisticated algorithms that do the mapping have to be implemented.

When the maps are generated, however, we also need to mark desired features on them in order to create topological maps. The model by itself will not be so useful as long as it is not labelled since the device will only have static and limited knowledge about it; it will not known that a set of steps represents a stair in which the user should be more careful when walking than on a flat surface or that a certain door serves as an entrance to someone's office. If the indoor environment is a building with offices, each of the offices need to be labeled so that whenever the user wants to go to a certain office, they will be able to chose from a list of offices. Similarly, products or sections can be marked in a indoor shop map or different offices and departments in a hospital map. Doors, stairs and similar should also be marked on these maps.

After the model is completed, we have to make it available to the users. This issue can be easily solved by offering the map as a downloadable file through an indoor navigation mobile application that integrates the other two components: the Navigation Algorithm and Obstacle Avoidance for indoor navigation purposes.

4.2 Navigation and Path planning

When the Indoor Map is completed, we move on to the Navigation Algorithm component. Navigation is based on planning the path that has to be taken to reach the destination and guiding the user through that path. The indoor map from the previous section will be necessary to compute the path.

The approaches for path planning can be classified in two general techniques: what [38] calls the global technique or [39] calls "path planning with complete knowledge" and the local technique [38] or "path planning with incomplete information" ("path planning with uncertainty") [39]. In the case of the global technique, full information about the environment and the objects in it is assumed and the algorithm can compute the entire path from starting point to the destination before the robot/user has even started

to traverse it. This technique is generally computed offline and then the robot takes the instructions to reach the destination. This assumes static environments and can be compared to our case when we compute the ideal A* path based on the 3D representation of the environment that we have created. The local technique assumes a dynamic environment where obstacles have changed their positions prior to the user's journey, or do so while the user is also moving. This technique has to be online and needs to deal with the data that is gathered from sensors all the time, segmenting obstacles from them and replanning the initial path. The technique does not assume prior knowledge of the environment. Going back to our project again, this method represents the Obstacle Avoidance component working independently from the Navigation Algorithm component.

The idea of this project is to combine both techniques to achieve indoor navigation with obstacle avoidance. First we use the global technique to calculate the shortest path with A* and then, considering that the indoor environments are dynamical and change from the static representation with time, we use the local technique to avoid obstacles along the ideal path.

4.2.1 Navigation and Pathfinding with Unity

After the indoor maps are generated and labeled, they can be used in Unity as the basis for the A* algorithm. The A* algorithm [32] computes the optimal path between starting point and destination for the generated 3D map divided into cells of a grid. The Euclidean distance as the shortest distance between two points in space, cannot be used to compute the distance in these cases because of the composition of the scene and the obstacles that may be positioned between two points.

Unity allows us to assign the NavMesh component to walkable surfaces and the NavMesh Obstacle component to obstacles that can be either static or dynamic and that are objects that the user (who is an assigned NavMesh Agent) should avoid when navigating. Through the NavMesh, the A* algorithm will compute the optimal path between two points in that mesh. While the obstacles are moving, the agent avoids them, but once they become static a hole is carved into the NavMesh so that the agent can steer away from them or recompute the path if the obstacle is blocking the way. Also, Off-Mesh Links can be created for the shortcuts that the user can take but that are not part of the walkable surface such as jumping over a fence or opening a door without walking through it [40]. These components assigned to elements of a static game scene can be seen in Figure 11.

The navigation system in Unity works as following [41]: the navigation mesh is stored as a surface of convex polygons. Both polygon boundaries and information about neighbouring polygons is stored. When the A* calculation starts, the starting point and destination are mapped to the closest polygons and then the path from the start polygon to the destination polygon is searched by visiting all neighbouring polygons. The sequence of polygons from start to finish is called a corridor and during the navigation the agent will be guided to move to the next visible corner of that corridor. A visualization this can be seen in Figure 12. Since there is only one agent present, the user, in our case, we can compute all the corners of the corridor at once and ask the user to walk along the segments that connect those corners. Otherwise, when there are more agents (other people or objects that move), Unity allows correction of these segments to make dynamic obstacle avoidance possible by finding the next visible corner of the corridor when the

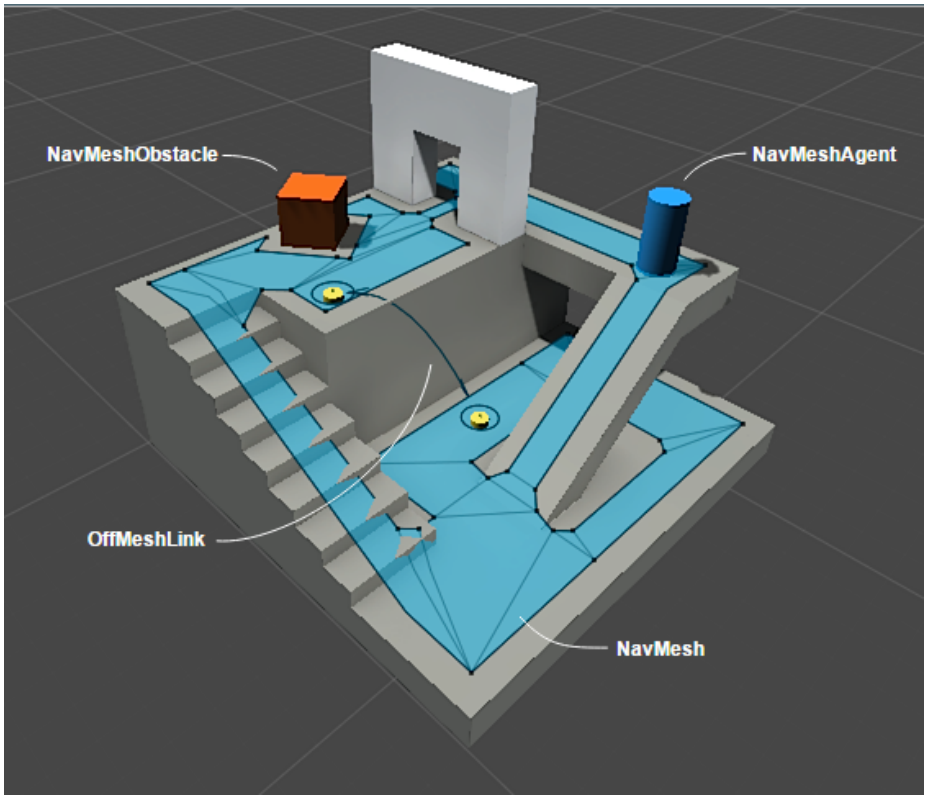

Figure 11: Navigation Mesh in Unity[5]

first one is blocked from sight. The navigation system in Unity also uses Reciprocal Velocity Obstacles (RVO) [42] to predict and avoid collisions with other objects based on the velocity of their movement and collision with the edges of the NavMesh based on the velocity of the agent only. Unity also includes Off-Mesh links in the A* calculations, but we do not use them in this project because we do not expect the users to be jumping between surfaces that are not walkable in the real environment like they would in a video game.

In the global navigation sense, the navigation system in Unity tries to find the shortest path through the entire game world, while in the local navigation sense it tries to efficiently find the next corner of the polygon without colliding with the surroundings. When an obstacle is so big that it cannot be conquered, it is considered a path blocker and it will be carved into the navigation mesh and then A* needs to be recalculated to find the other best path based on the new information and the new position.

[5]The image is owned by Unity and can be found at: `http://docs.unity3d.com/Manual/nav-NavigationSystem.html`

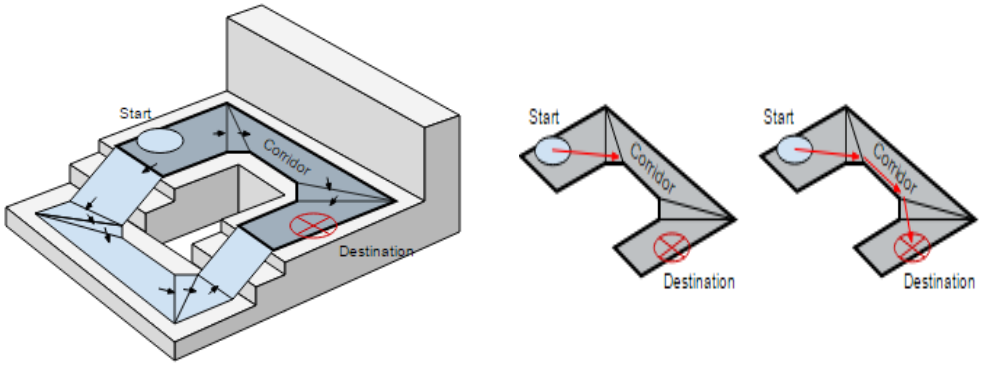

Figure 12: Pathfinding[6]

Since Unity is a game engine and the navigation system is made with navigation in games in mind, it is easy to compute the shortest path for an environment where all elements, how they behave and their physics are known at all times. Whenever an obstacle that should be avoided approaches the agent or blocks the way, the navigation mechanism will work in avoiding it or computing the next best path. In our case, however, achieving the same thing is not easy at all, if not impossible, since we deal with unknown elements that may change their position all the time without us being able to keep track of them. That is because we are able to offer Unity only the map that we have generated in the previous step and in this map everything is static and shortest paths can only be true shortest paths if the environments have not changed in major ways since the mapping has been done, which is quite impossible to assure in dynamic environments. Also, when the live depth data is incorporated, it only offers a partial view of the scene – the device will see only what is in the range of the depth sensor – making it impossible to compute the shortest path based on the current composition of the scene. Therefore, computing the A* path with Unity does not necessarily guarantee the shortest path, but if environments do not change in ways that would completely block paths with walls or other obstacles that cannot be conquered in any way, the computation of the shortest path with A* will be nonetheless valuable as it would be the only component that makes navigation in the true sense possible. Otherwise, obstacle avoidance by itself cannot take the users to a desired place, it can only navigates them around obstacles.

The biggest weakness of the A* algorithm is the large amount of computation power that it requires in order to compute the shortest path. This is because it runs through all the neighbouring polygons until the shortest path is found and when the paths need to be recalculated a path through the entire scene needs to be recalculated. These recalculations are done when the path that was calculated first turns out to be completely blocked by an obstacle that cannot be conquered or when the user does not follow the suggested path and makes unexpected turns. In these cases the path needs to be recomputed based on the new knowledge and the new position in regards to the destination. Francis et al. [38] overcome the replanning problems by replanning only the parts that are affected by the movement of other obstacles in the scene. The movement of the other obstacles is predicted based on the speed and direction of movement that is registered from the

[6]The images are owned by Unity and can be found at: `http://docs.unity3d.com/Manual/nav-InnerWorkings.html`

robot. Replanning only affected paths seems trivial when some of the obstacles in the scene only cross and not completely block the chosen path. Also, if the user's position has not drifted so much from the desired path, it is not so difficult to replan the path that would take them back to desired track. This solution cannot be used when there is a new wall built in the middle of the building or when a door that takes you to the destination is locked. In these cases A* will have to be recalculated. However, to save some computation power, cells beyond the first starting position do not need to be checked as they have already once been skipped and would take the users further back than they were in the first place.

4.2.2 Localization and Tracking

Two of the core pillars on which navigation of any kind stands are localization and tracking. In order to be able to navigate users, it is essential to be able to position them and the chosen starting point and destination first, and then track them within the map. It is useless to have a map with the shortest path when the application is not able to keep track of the users' position as they move. In our case, to achieve localization and tracking, we use inertial motion tracking sensors and the depth sensor. First, the sensors are used to position the user within the 3D representation. This is done by using the depth sensor to gather information about the surroundings and comparing the gathered point cloud to the generated 3D map so that the user can be localized in the space given by the map. At this point the users will be required to scan around themselves to make it easier for the sensor to locate in space. The accelerometer and gyroscope are also on in order to mark the starting point, but at this stage cannot help in positioning since they are not expected to have any knowledge of the environment that the user has entered. Since the user chooses the destination from a list of possible destinations that are also labelled in the map, the position of the destination can be easily determined from the position of the label. Then, once the shortest path is estimated and the user starts to walk, the inertial sensors serve to track the user compared to the starting point. The track that the sensors mark, with a small allowance for positioning error, is compared to the A* path. This way of positioning and tracking is known as Simultaneous Localization and Mapping (SLAM).

The biggest issue with positioning and tracking is accuracy. Each of the sources that are used in this process by themselves are prone to noise and then they also need to be fused together to give the ultimate position. In the Related Work section we have provided details on the errors each of the sensor brings: the depth sensor of Project Tango is noisier than other depth sensors [29], heading error is accumulated and after some time the registered values drift from the actual position [24], the yaw rate also needs to be corrected [21] and so on. Usually the error of the heading is corrected with fusion with magnetometer data, but on the other hand the magnetometer is easily affected by magnetic fields around [24], especially from the device's speaker that has a magnetic field 30 times the magnetic field of Earth [25]. Wagner et al. [21] obtain the yaw rate from Kalman Filtering with a single track vehicle model and incorporate a gyroscope to correct the model. The more sensors that are added to the already used sensors for motion tracking, the more difficult it becomes to integrate them.

By far the most used method for integration of sensor data is the Extended Kalman Filter (EKF) and its variations [27]. Although EKF gives good performance when it is properly tuned, Martin et al. [27] report its drawbacks: "it is not easy to choose the

numerous parameters; it is computationally expensive, which is a problem in low-cost embedded systems; it is usually difficult to prove the convergence, and the designer has to rely on extensive simulations".

Michel et al. [25] give a comparison of six algorithms developed for improved attitude estimation in the case of pedestrian navigation. The work discusses how the gyroscope, magnetometer and accelerometer are affected from the way the devices are carried and the effect of magnetic deviation has on them. Then the authors go about in comparing different solutions offered by six algorithms developed for the purpose. Their most important outcomes related to sensor integration are:

- Magnetic deviations present in the environment (speakers, magnets, walls, floors, elevators, belts, keys, etc.) heavily affect smartphone's sensors, thus affecting the precision of attitude estimation algorithms in pedestrian navigation. E.g. the speaker on the smartphone has a magnetic field 30 times stronger than Earth's magnetic field.
- The gyroscope is usually not enough for attitude estimation so the accelerometer and magnetometer are used to correct the drift of the gyroscope. The accelerometer corrects pitch and roll angles while the magnetometer corrects the yaw. The combination of data from both inertial and magnetic sensors then becomes a problem for the algorithms that integrate them.
- The paper gives error models for drift and sensor bias for each of the sensors.
- External acceleration of the device is not negligible in attitude estimation.
- The Android API offers quaternions and a "black-box" calibration algorithm which is shown to be quite efficient in data calibration. It is the algorithm the least affected from a magnetic field 5.5 stronger than the field for free space, created by surrounding the device used with 6 magnets in static position.
- The comparison demonstrates that uncalibrated data is useless for the purpose of attitude estimation since it deviates a lot from the ground truth because of the presence of the strong magnetic field of the speaker close to the IMU.
- Estimation becomes more difficult the more acceleration deviates from g, making attitudes such as walking with a swinging hand and walking with a phone in the back pocket difficult to estimate.
- The authors make a comparison on the performance of each of six algorithms for a set of motions, a comparison between the motions themselves, show the importance of calibration and correction of magnetic deviation as well as give a comparison of processing time for each algorithm as saving battery life is of high importance in smartphones. In case of battery life the Madwick [26] (Gradient Descent based Orientation Filter) filter outperforms the others.
- Authors conclude that the algorithms by Martin [27] (Invariant Nonlinear Observers) and Fourati [28] (Complementary Filter Algorithm) work the best for integrating sensor data for the purpose of attitude estimation. Both of the algorithms are nonlinear filtering approaches and neither are based on the Kalman Filter.

Calibration of the motion tracking sensors is an important measure in the avoidance of drift accumulation. Project Tango can be calibrated with the Project Tango Calibration App and instructions on performing the calibration are given in [43].

4.3 Obstacle avoidance

When the 3D mapping of the environment and the navigation algorithm are completed, for purposes of navigating without running into obstacles, we need to work with the live-feed of the sensor while the navigation from the starting point to the destination is running. This is necessary since the objects that we have first mapped in the 3D model may have changed their position, presenting obstacles in the way of the user. This calls for changes in the planned path so as to avoid the immediate obstacles that are in the way and that the sensor can perceive. Due to the fact that the infra-red depth sensor cannot see further than 4m or closer than 0.8m at its best, we can only deal with obstacles in the field of view and cannot 'prepare' the user for the obstacles that are outside of the sensor's field of view. And since the sensor sees only the faces of objects that are in front of it, the path planning needs to be done after every step - with each change of the view.

When it comes to path planning around obstacles using the live-feed of the depth sensor, there are three major problems we encounter:

1. The data that we receive from the sensor is noisy and objects are over-represented,
2. The gathered data is in the form of a point cloud, which requires us to model objects from the data in order to know the shape, size and distance of the obstacles in sight and,
3. After data is processed and modeled, we need to plan the path around the contours of the obstacles.

Each of these problems, again, involves smaller details that need to be considered. We describe each of them in the following sections.

4.3.1 Reduction of point cloud data

The infra-red depth sensor of Project Tango gathers data with an update rate of 3Hz (worded differently, the sensor has a frame rate of 3 frames per second). An update rate of 3Hz means that the process of the depth projector casting infra-red waves onto the scene and the depth sensor sensing the reflected waves is completed three times for each second that the sensor is on. Although this update rate might seem low, because of the light nature of infra-red that falls all over the scene and returns to the sensor with the speed of light, a big amount of data will be gathered at each frame. There is no actual limit to the amount of data that can be gathered. We have noticed that the amount of data points per frame depends on the environmental conditions that affect how the infra red waves spread and varies from several hundreds to several thousands. This will cause high quantities of data to be gathered from the same objects within seconds (in the range of hundreds of thousands data points), as the user will take several frames to walk to a part of the scene that is new to the sensor.

It is clear that besides the three dimensional data being sensed, the data gathering also depends on a fourth dimension: time. With the passing of time more and more data will be gathered and as it turns out, quite often the data will be repetitive as the slow movements of the user (compared to the update rate) will cause several rays to fall

on the same points of the objects of the scene and thus return the same depth value. Furthermore, due to the effects the environment has on the sensor – light and heat interfere with it – some of the data points that are gathered will be of invalid value, but still registered (as (0,0,0) data points). The depth sensor from Project Tango is also known to be nosier than the rest of the available depth sensors [29]. All these data points – the valid, the invalid/noisy and the repetitive ones – will be looped through when computing shapes and calculating distances from the gathered point cloud. Having large amount of data to work with can make the calculations expensive and slow. In order to avoid noticeable lag, all the processing has to be completed before the next frame renders.

In order to get more insights about the data that is gathered from the sensor, we have saved the point clouds in a text files and later analyzed them with Matlab. After reading the depth data from the registered scene to a 3xn matrix of (X,Y,Z) values, we tried to visualize the data points with Matlab's *scatter3()* function, but after the visualization took more than 10 minutes, we decided that the data needs to be cleaned of invalid and repetitive data points first, in order to make the visualization step less time-consuming. We used the *unique()* function to keep only unique data points (rows in the depth matrix) and tested the entire depth data matrix for (0,0,0) elements and removed them. For the first scene, to take it as an example, from 137408 data points that we had at the beginning, we were left with only 16194 data points after cleaning invalid data points (these comprised 88.2% of the matrix) and 15300 after the repetitive data was cleaned (another 0.7%). Table 1 offers a sample of data points gathered from that scene after it was cleaned from repetitive and invalid data. Figure 13 shows a visualization of the remaining 11.1% of the data. Unsurprisingly, the point cloud looks exactly the same as it did before cleaning the data, only the visualization process was five times faster. After the visualization is done, the Euclidean distance of each point from the sensor is calculated and added to the depth data matrix.

After processing the data from five scenes, these are the three important insights we have gained about the depth data:

- Depth data comes in does not follow a certain structure, meaning that the first points that reach the sensor are written first and this does not necessarily imply that the rays from the points that are closest to the sensor will reach first and will be written first. From the Euclidean distance of each point in Table 1 we can see that this is true. Furthermore, we have noted that most of the time the points that are furthest are written earlier than those that are closer (from the table can be seen how points at 3.85m distance are registered before the 0.6m ones).
- Up to 88.9% of the gathered data is either invalid or repetitive. From this set of point clouds it seems that the invalid data is the most troublesome, the amount of repetitive data is negligible.
- The objects of the scene are densely represented with points (see Figure 13). We can consider the repetitive data negligible, but the problem of data points that differ from each other for only millimeters when they belong to the same object is still a problem that cannot be neglected because computations with so many data points take a lot of processing power and may hinder the system from running in real time.

Table 1: Sample depth data from Project Tango's depth sensor (all data is given in meters)

X	**Y**	**Z**	**Euclidean distance**
-0.85805	-0.55912	3.71362	3.852252
-0.85655	-0.54091	3.684502	3.821234
-0.79084	-0.54609	3.752141	3.873268
-0.75705	-0.46156	3.749837	3.853238
-0.73183	-0.50048	3.794929	3.897118
-0.74584	-0.52915	3.775691	3.884859
-0.6664	-0.61435	3.85941	3.964411
-0.702	-0.57827	3.853181	3.959067
-0.5908	-0.54752	3.858019	3.94121
-0.69041	-0.52597	3.838103	3.935015
-0.06756	-0.07068	0.611407	0.619175
-0.07066	-0.05784	0.598128	0.605057
-0.04312	-0.05748	0.601956	0.606229
-0.03072	-0.06137	0.603032	0.606926
0.8608	-0.61227	4.493241	4.615741

When the data is sensed, we use it to infer information about the scenery – the objects that are in it and their distance – by looping through the depth data and constructing models from them. Since data is sensed continuously in the time dimension while the sensor is on, the data structure that registers the point cloud will increase continuously too, making the process of looping through it never-ending. To keep the navigation real-time and to avoid noticeable lag, all data processing needs to finish before the next frame. On the next frame new data will be added to the point cloud and the processing step has to be repeated because every new frame will add new information to the understanding of the scene. From the insights we have gained by depth data analysis, we can conclude that point clouds should not gather more data than from a few successive frames because of two reasons:

1. It will become more and more difficult to process the data if it keeps growing infinitely and
2. After a few frames, the data from the first frames might become irrelevant to the user's new position.

If the data gathered within a small number of successive frames can still not be processed before the next frame renders – we're dealing with depth point clouds of hundreds of thousands points – then we will have to consider a trade off between time and quality of obstacle reconstruction. Since for online navigation purposes we are interested in keeping the process real-time, we have to trade quality for time, meaning that we need to limit the number of data that will be processed each frame or lower the update rate

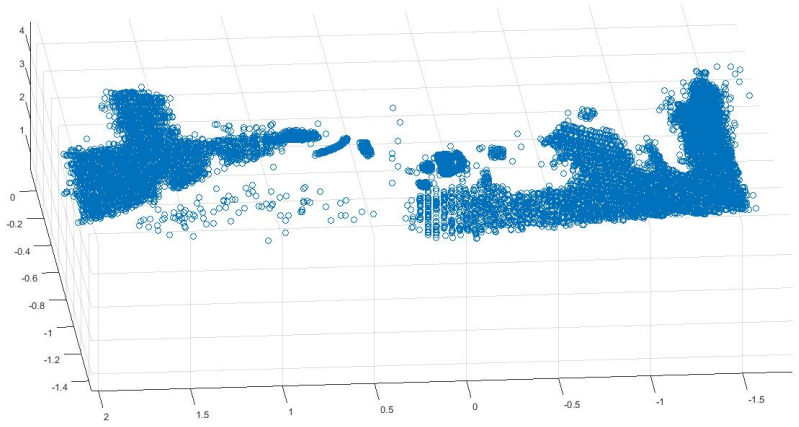

Figure 13: Point cloud gathered during a 3 second scan with Project Tango

of the sensor. The first approach calls for the creation of temporary data structures that save only a certain number of the gathered data and processes that only. Regarding this process, many questions, such as: Which data to select for processing? Process data in a first come, first serve style? Or do a selection of certain points? arise. Since the data comes in unstructured and we cannot limit the sensing to a certain distance, it is difficult to have some robust selection criteria without adding to the computational complexity. If we would want to process the data that is closest first, say within 1.5 meters distance, we would have to loop through the entire point cloud once to select the points that fulfill the criteria and only then proceed with other computations. The simplest that can be done is to limit the incoming data that will be processed to a certain number of data points and process them as first come, first served. Thus the number can be limited to something that allows the device to process the data without noticeable lag. But these approaches to limiting the incoming data has to be experimented with before taking an ultimate decision.

We have to recognize that while we are trying to reduce the data in order to win in computation speed, we degrade the value of system since while we are trying to get instructions for possible collision to the user faster (before collision), we are also risking not representing all the obstacles or not representing obstacles well, which will result in the user colliding anyway.

4.3.2 Obstacle modelling

Although it might not be immediately apparent, even though a computer sees a point cloud of depth data, that does not mean that it can *perceive* objects or the scene from it. Computers are 0, 1 systems that cannot intelligently make sense of anything on their own (until they are programmed to do so). This means that even though the computer may have the perfect point cloud from the scene (which it never does) it will still not know anything about the environment; it will just see a set of 3D points that do not mean anything until some sense is made of them. Thus, in order to make the computer more knowledgeable about the environment and obstacles in it, we need to model objects from the point cloud and let the computer *understand* that they are obstacles for the user when they are on the path which the user is supposed to take. Besides, the computer does not

know that a lack of depth points in the scene means empty space, until it is told so. Just having sensed the point cloud has not solved the object and scene perception issue.

When it comes to obstacle modelling for obstacle avoidance, we need to handle a problem that is different from the modelling of indoor environments that we dealt with in the first part and different from a lot of previous works that focus on how to perfectly represent objects from the gathered depth data (as an example see [44] and references therein). At this stage we need to model obstacles on the go, while the user is walking, and only with the data that is available at that point in time and space. Since the point of this application is in notifying the users of the obstacles in front of them before they run into them, the modelling has to be done fast enough to keep up with the step of the user. Therefore, again, we trade quality of representation for computational speed by using a more coarse-grained representation of the obstacle from the noisy point cloud that we get from the sensor. Note that because of the lack of complete data about an object at any given time, even if we wanted to do a perfect representation of that object we would not be able to. Creating such a representation would be possible only if we scan the object from all sides several times and to do that a lot of time would be spent that would keep the user from moving forward.

We reason that representing obstacles through their surface planes is good enough for purposes of path planning around obstacles because we are not interested in avoiding details of the objects, but objects as a whole. To achieve this, we need to work on plane segmentation from point cloud data. And in order to achieve plane segmentation, plane fitting must be completed first. We need to test which points belong to a plane and then segment the plane from the scene. Research shows that from plane fitting algorithms, the Random Sample Consensus (RANSAC) algorithm is widely used for purposes of plane fitting (at the time of writing has 14719 references). RANSAC was first presented in 1981 in [45] and it is known as a real-time, efficient algorithm for fitting inliers in their respective planes and eliminating remaining outliers caused by distortion or noise [46] [47]. Three desirable properties of RANSAC are [47]:

- Conceptual simplicity – makes it easily extensible and straightforward to implement;
- Generality – allows its application in a wide range of settings;
- Robustness – deals with data containing more than 50% of outliers;

Holz et al. [46] use a RANSAC-based algorithm to reliably detect obstacles in a scene at high frame rates (30Hz). Schnabel et al. [47] develop a RANSAC-based algorithm that successfully detects planes, spheres, cylinders, cones and tori and is shown to be robust even in high levels of noise and outliers. The proposed algorithm also scales well to the number of data points, processing two millions of points for less than a minute and the number of objects in the scene. Holz et al. [48] also show that the robustness of RANSAC against noise and outliers allows it to be used for segmenting and classifying objects from 3D images of a Time-of-Flight (ToF) camera (we are also using Tango's ToF sensor). Figure 14 shows RANSAC plane fitting achieved with the Plane Fitting Android application from the Project Tango developers[7] [49]. Figure 15 shows that RANSAC successfully

[7]The application can be found here: `https://github.com/googlesamples/tango-examples-java/tree/master/java_plane_fitting_example`

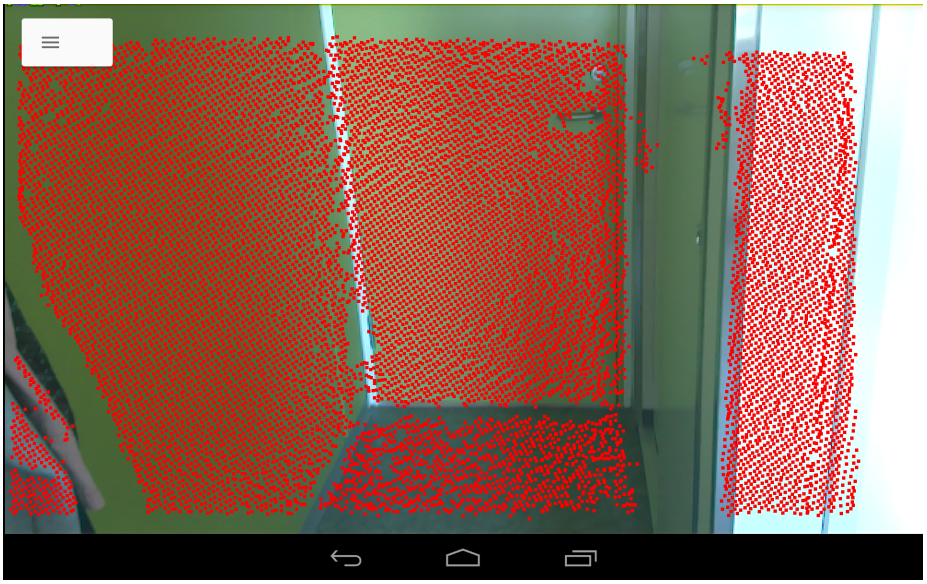

Figure 14: RANSAC Plane fitting achieved with Project Tango's Plane Fitting application

identifies outliers.

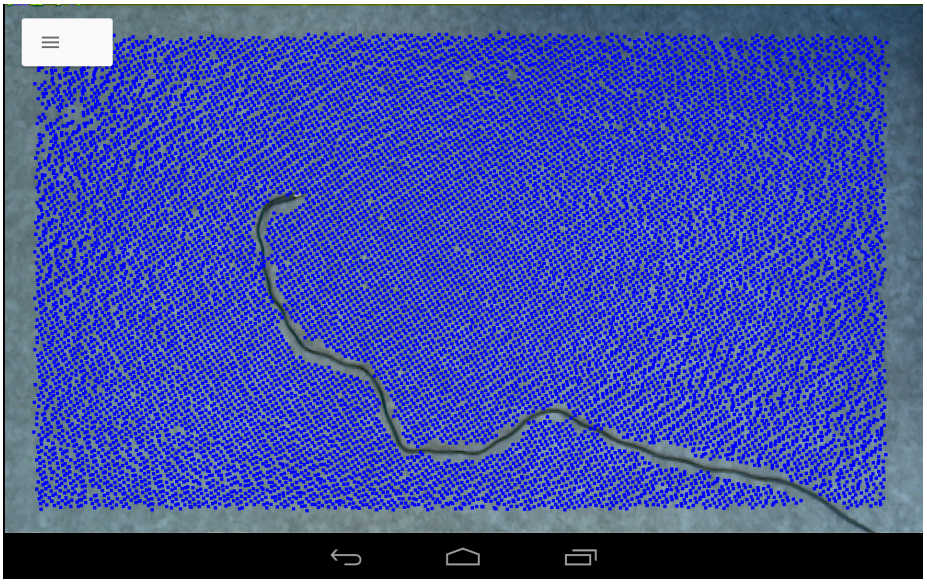

Figure 15: RANSAC correctly identifies outliers

Once data is fitted to planes and the outliers are removed, the Graham Scan [50] algorithm can be used to compute the convex hull of a set of data that belong to a plane [48] in order to segment the object planes from the scene. By computing the convex hull of an obstacle we ensure that all of its edges are "within" the hull and therefore as long as the users do not not collide with the hull they will not collide with any of the edges of the object. Figure 16 illustrates how the algorithm works. So, although the shape of the object might look like the first geometric shape (image a)), after the convex hull is computed we will get the second shape (image b)) and two of the vertexes will

end up within the surface defined by the hull. After the convex hull is computed, the data points within the hull contour can be discarded (image c)). As [51] notes, the "sidedness" of object contours – which side is solid surface and which side is free space – is very important in collision-free motion. Since the Graham scan offers a closed contour, it is easy to keep track of which part is the inner part of the object and which is the outside.

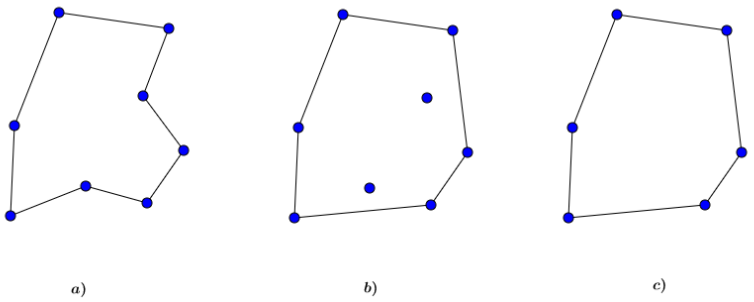

Figure 16: Graham scan work principle

If the distance between two objects in the scene is smaller than a certain distance through which the user can walk, say 0.5m, their points can be merged, as the user won't be able to walk in between. Blum et al. [52] also guarantee a certain "thickness" of the obstacles by assuming that a unit circle can be inscribed in each of the obstacles within the field of view. In that way obstacles that are smaller than the unit circle can be ignored.

An obstacle, just like any other object, can be defined as a closed curve with finite length [39]. However, the finite length of the obstacle curve will not always be observed as such from a range sensor, such as our infra-red depth sensor, when the object extends beyond the sensor's line of sight. This is one of the points that, to the best of our knowledge, has not been discussed in literature before. When the plane fitting algorithm will try to segment a plane from the point cloud data, it might not be able to do so due to the "infinite" boundary on one or more sides of the obstacle. We devise the approximation of these boundaries into temporary "planes" by projecting the data that seems to be prolonging infinitely to a line that will serve as the boundary until the device is able to sense the true boundary. The algorithm should take care not to consider temporary boundaries as chances to take a turn and should not perform computations on how to go past such boundaries until a finite surface of the obstacle is sensed.

4.3.3 Obstacle avoidance

Once the objects have been modeled, we need to find a way to effectively steer the user away from them if they are in the user's path. But, as [38] also noted, because of the lack of the information of the entire surrounding where the user is to be navigated, path planning in a changing environment is quite complex and difficult to achieve.

As was discussed earlier, there are two approaches for path planning: the global technique or "path planning with complete knowledge" and the local technique or "path planning with incomplete information" ("path planning with uncertainty"). The idea of this project is to combine both techniques to achieve indoor navigation with obstacle avoidance. The global technique was investigated thoroughly in Section 4.2. In this section we

discuss the local technique and some of the approaches of solving this problem. We assume that the global technique is completed and that we are following the suggested A* path. So, the obstacle avoidance component works together with the suggested path to reach the destination, but if such suggested path is not available, this component should be possible to be used independently but will only guarantee obstacle avoidance, not arrival to the desired destination.

Francis et al. [38] also combine the two techniques to achieve path planning in an indoor environment. They assume that an initial state of the environment and obstacles in it is known and then go on to follow how these obstacles move and whether their movement affects the path of the robot. The path of the robot is initially planned using the A* algorithm and then the trajectory of the moving objects is predicted based on their speed and direction of movement. If the prediction shows that the trajectories of the objects in the field of view will affect the robot's initially planned path, the path is replanned to avoid possible collision with those obstacles. The algorithm runs the same procedures until the destination is reached. The authors do not give insights into how they deal with obstacle segmentation out of point cloud data; they only assume that the robot will sense the geometry of the moving object and its trajectory of movement based on the objects vertices. Since the A* algorithm takes time in calculating and recalculating the path for the entire map based on incoming data, the authors propose a modified version of A* that replans only the parts that are predicted to affect the robots path, not the entire map. A simulation of the algorithm is done with Matlab, where the predictions of obstacle movements are given with spline functions. The simulations show that the algorithm effectively replans the A* path when possible collisions based on obstacle movement predictions are perceived. The authors plan to implement the proposed algorithm for a robot that uses a 3D time-of-flight camera.

Kunchev et al. [53] give short descriptions of path planning and obstacle avoidance algorithms by summarizing a few of them (seven obstacle avoidance and four path planning algorithms), and present a control model for steering Unmanned Automatic Vehicles based on path planning and obstacle avoidance. They name the local and global path planning techniques as reactive behaviour and planning behaviour, respectively and consider the two techniques as opposite approaches: If the robots rely on path planning alone, they will be prone to collisions with obstacles on the fly; If reactive behaviour is used by itself, it will be impossible for the robot to reach the goal. The authors conclude that both approaches have to be combined in order to overcome the weaknesses of each of them when used by themselves.

Lumelsky et al. [39] address two questions related to the incorporation of range sensing in robot navigation: Is it possible to combine sensing and planning functions to achieve "active sensing" similar to how it is done in nature? and Can richer sensing (such as stereo versus tactile) guarantee shorter paths? Their work shows that it is indeed possible to achieve natural sensing by combining stereo sensing with path planning algorithm, but their developed algorithm does not guarantee the shortest path. The work focuses completely on the local technique of path planning with knowledge on where (and in which direction) the destination is located. The authors conclude that as long as the sensors do not see the complete scene in which they have to navigate in, which is the same for all sensors, the shortest path cannot be guaranteed, even when the quality of sensors is improved or better algorithms are developed. So, the inability to compute

the shortest path is a weakness of the path planning with uncertainty/local path planning, and not of the developed algorithms or the quality of sensors. Papadimitriou and Yannakakis [54] prove mathematically that there is no strategy that achieves a bounded ratio for the shortest path when a map of the environment is not available.

It is interesting to note that a lot of the works that claim to engage on obstacle avoidance in unknown environments always assume a known target/destination. For reference see [39], [52] or [54]. We believe that as long as there is no map of the environment and while the device gets *familiar* with the environment only by visiting it, there is no way of knowing where the target is without having reached it in prior. So, the shortest path to an unknown place cannot be computed and thus the independent obstacle avoidance component can only help in exploration of indoor environments without collision with obstacles.

We believe, however, that by combining the obstacle avoidance component with the navigation algorithm in a given map, this problem can be overcome. Indeed obstacle avoidance by itself will not guarantee the shortest path as long as the environment is unknown. The integration of the live depth feed augments the Indoor Navigation algorithm but also makes the system more complex. At each step the device will have to decide what step to take based on the information gathered about the scene from the depth sensors.

Blum et al. [52] compare the distance walked by a robot to reach a target within an unfamiliar environment with the shortest distance to that target. This works seems limited: it assumes full knowledge of the obstacle when it is in sight (such as size, position and shape) and limits the rooms and obstacles to square rooms and oriented square obstacles. Real environments are a lot more complex and we do not expect such “perfect” rooms and obstacles to be widely common which makes the approach. Another limitation of the algorithm proposed by the authors is that it requires that the robot sweeps back and forth along obstacles and if we were to design an navigation system that requires the same, it would not replace the walking stick of the blind since using the walking stick will take less time than sweeping several times with a tablet.

Kamil et al. [55] reviewed 80 works from the last five years on the motion planning and obstacle avoidance approaches used in dynamic environments. The paper identifies several problems that would provide for better online motion planning algorithms, when taken into consideration. The conclusions are:

1. The relative velocity, which is the velocity of the obstacles relative to the velocity of the robot, is the most important factor in dynamic motion planning. The works that were analyzed did not consider the relative velocity as a constraint in their applications.
2. Some of the works took into account the kinematics of the robot, but others did not. It is important to take into account real-world elements that constrain the movement of the robot because that will make it easier to transform the designed application to a practical one.
3. For successful robot navigation in dynamic environments two problems need to be resolved: real-time recognition of moving obstacles and shortest and safest path has to be produced dynamically.

4. Heuristic approaches do not guarantee the discovery of a solution and to improve the efficiency of the reviewed approaches, those approaches should be combined.
5. Because of the dynamics of the environment and missing of complete data about it the difficulty of motion planning in dynamic environments will increase, which calls for an improved technique for hybrid meta-heuristic motion planning.

In the next chapter we propose a straightforward approach to obstacle avoidance, which combines both global and local motion planning, but when the global path is not available, that approach can be used with small modifications to avoid obstacles without the objective of reaching a goal or the shortest path to the goal.

5 Proposed Algorithm

In the previous chapters we discussed in detail the elements that need to be completed in order to achieve an indoor navigation system based on motion tracking and depth perception sensors. We divided the entire system in three components: the Indoor Map, the Navigation Algorithm and the Obstacle Avoidance Algorithm. We saw that in order to implement the whole system, many details need to be thought of and achieved. In this chapter we consider the steps from the previous chapters as completed and focus only on the implementation of the Obstacle Avoidance component. We assume that we have the map of the environment and the A* generated shortest path from start to destination. In that way we combine global and local motion planning to achieve the shortest collision-free path.

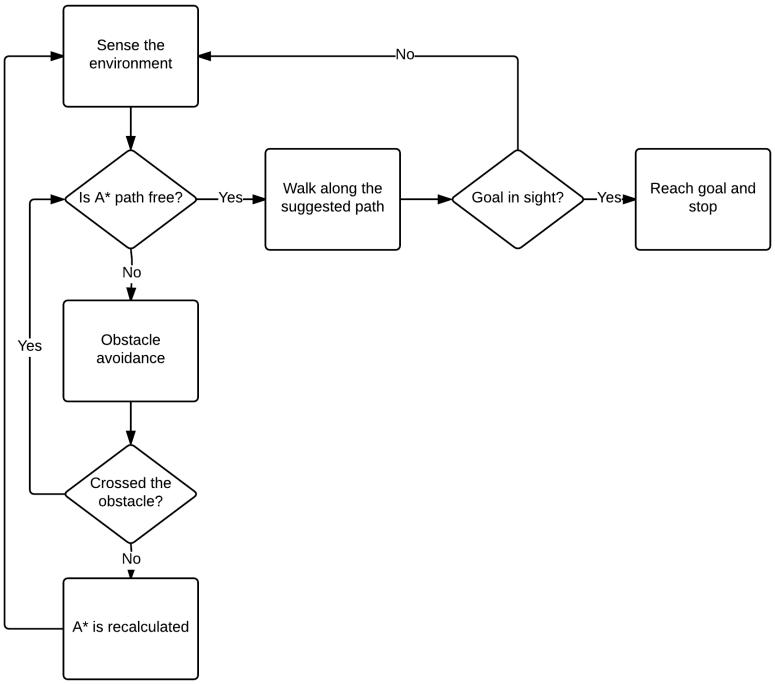

Figure 17: Proposed algorithm

In Figure 17 we can see a flowchart of the algorithm we are proposing. In order to be able to use the Obstacle Avoidance component, the user has to keep the depth sensor on during the entire navigation process. Once depth perception is turned on and the user has chosen a destination in the 3D map, the user's position will be estimated within the map of the environment and the A* algorithm will compute the shortest path to the destination. Although the user should be able to choose a starting point that is different from the current position, to simplify matters, here we consider that the starting point

matches the current position and the user has entered the environment in which s/he is to be navigated. Otherwise, the starting point could be chosen from a list of possible points labelled in the map, just like the destination is chosen. We assume that at all times the motion tracking sensors are used to track the user in the 3D map and to compare the current position to the suggested A* path. The user is instructed to follow the A* path as long as nothing is blocking it. The depth perception sensor will sense the environment and based on its field of view determine whether the A* path is free or not for the length that is visible. If the path is free, the user will be instructed to follow it until the goal is reached. If the path is closed by some obstacle, the Obstacle Avoidance Algorithm instructs the user in avoiding the obstacle while correcting the A* path only in the part affected by the obstacle. If the obstacle cannot be avoided after walking around it and the user does not get back onto the A* track, that means that the obstacle is blocking the way completely and cannot be overcome. If this happens the second best path needs to be calculated with A* considering the (new) current position and the blocking obstacle in the way. As designed in Section 4.2.1, the A* algorithm is modified to not recede the user from the initial starting position, thus its recalculation will take less time.

Let's see how exactly the avoidance of obstacles is done. In order to achieve obstacle avoidance, the immediate environment is registered through the depth sensor and then the RANSAC algorithm is run in order to achieve plane fitting from the dense point cloud. After the data is fitted to their respective planes, the convex hulls of objects are computed from the planes with the Graham scan algorithm and the unnecessary data is discarded, keeping only the hull. The sensor plane is the plane that goes through the (0,0,0) point (the focal center of the sensor) and is perpendicular to the floor. We consider four possibilities for the view in front of the device:

1. The view is open; no obstacles are in sight.
2. There are one or more obstacles right in front of the device, stretching in both sides of the sensor plane.
3. There are obstacles in both sides of the sensor plane but the plane does not go through them.
4. There are one or more obstacles on one side of the sensor plane.

The first case is already treated; if the space is empty and it matches the A* track then the user is instructed to walk along it. In the second case, when the obstacle is on the suggested A* track, the user has to walk around one of the visible corners of the obstacle. Because of lack of knowledge of the complete obstacle, the turn we take – left or right – has equal weight; neither direction can be judged better than the other in a global performance point of view [39]. We decide to take the turn that keeps the user closest to the A* path, although chances are that that will not be the fastest way to cross the obstacle, but a better decision cannot be taken as long as the obstacle remains largely unknown. If the A* path seems to go right through the middle of the visible obstacle plane, then the path that takes the user to the closest obstacle corner will be taken.

At this stage, there are only two cases left: either there are objects on both sides of the sensor plane, or there are objects only on one side. If the obstacles are located only on one side, we first find the vertexes that are closest to the sensor plane (one vertex from each hull). Then we find the one vertex that is the closest of all. This is easy to do

because we just need to find the vertex with the lowest x-value if the obstacles are on the right side of the sensor plane and the highest x-value if the obstacles are on the left from a finite and small set of data points. After that, a plane is constructed through the selected vertex and parallel to the sensor plane. We consider the user to be enclosed in a bounding box collider of variable size (depends on the user but can be initially set to a standard size) which makes it easier to test for collision of the box with the obstacle plane.

In the last case, when there are obstacles in both sides of the sensor plane but the sensor plane does not go through them, again the vertexes closest to the sensor plane are found on both sides in the same manner as above and again the planes through chosen points from both sides and parallel to the sensor plane are computed. Next, the distance between the two virtual parallel planes is computed. The planes are parallel to each other because they are both parallel to the sensor plane. If the distance is bigger than the width of the collider box of the user, the user is instructed to walk in the middle of both obstacles, but if the distance is smaller then the obstacles can be merged to represent a single obstacle. This case then reduces to the second case when the user needs to walk around on one side of the obstacle set.

To test for possible collisions with the obstacles, the distance between the sides of the virtual bounding box of the user and the constructed planes is measured. If the distance is a positive number, that will mean that the user will not collide with the obstacle if s/he keeps walking in the same direction as s/he is at that point in space and time, but if the distance is negative, that means that the planes are closer to the user than the bounding box and hence the user will have to move around the obstacle and not towards it.

So far, we implicitly considered the obstacles to be static objects in the user's way, but since the algorithm is updated every 0.33s (3Hz is the update rate of the depth sensor) and the planned path is reevaluated every frame, it can also detect moving obstacles that are moving closer to the device in the field of view of the sensor. Collision with moving obstacles can occur when the objects move so fast that they will not be seen within the field of view of the device (the field of view has to be crossed in less than 0.33s) or collide with the user from the sides s/he is not scanning the environment in.

5.1 Plane construction

In order to construct the planes and to find the distances between them and specific points, we use plane equations. These can easily be derived. Consider two planar points: P with coordinates (x, y, z) that could be any point in the plane and P_0 with coordinates (x_0, y_0, z_0) which is a determined point, similar to the ones in Figure 18. Their respective vectors will be $\mathbf{r}$ and $\mathbf{r_0}$. The difference between these two vectors, $\mathbf{r} - \mathbf{r_0}$, will lie completely on the plane. Consider also a vector $\mathbf{n}$ with coordinates (a, b, c) perpendicular to the plane. This is the normal vector. Since the normal vector is perpendicular to the plane it will also be perpendicular to any vector that lies on the plane. This means that the Dot product between the normal and the $\mathbf{r} - \mathbf{r_0}$ vector is going to be equal to zero. We use Equation 5.1 to derive the equation of the plane.

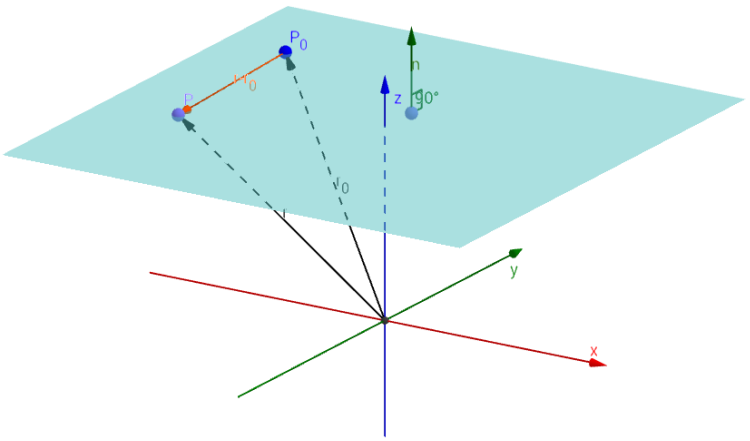

Figure 18: Derivation of the plane equation

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{r_0}) = 0 \tag{5.1}$$

$$(a, b, c) \cdot [(x, y, z) - (x_0, y_0, z_0)] = 0$$

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$$

$$ax + by + cz = ax_0 + by_0 + cz_0$$

$$ax + by + cz = d \tag{5.2}$$

Equation 5.2, where $d = ax_0 + by_0 + cz_0$, represents the Cartesian equation of a plane. From this equation we can see that when we know the vertex closest to the sensor plane, we will immediately find the plane that goes through it.

If instead of the one point and the normal we have three non-collinear points (points that do not belong to a line), we construct two vectors from them and use their cross product to compute the normal to the plane. After the normal is determined, we use it to determine the equation of the plane by replacing the (a, b, c) values of the normal in the Cartesian equation. Once we do that, we will notice that d in the equation is unsolved. We solve it by substituting the values of one of the points in the equation and thus we get the final equation of the plane.

In our case, we need to compute a plane that is tangential to a point and parallel to the sensor plane. Clearly, the equation of the sensor plane is $x = 0$. One normal of this plane is $(1, 0, 0)$ (derived from Equation 5.2). Since the two planes will be parallel, their normals are either parallel or the same. For simplicity, we can use the same normal for both planes. We derive the new plane equation from the Cartesian equation of a plane:

$$ax + by + cz = ax_0 + by_0 + cz_0$$

$$1x + 0y + 0z = 1x_0 + 0y_0 + 0z_0$$

$$x = x_0 \tag{5.3}$$

The equation of the new plane that goes through the obstacle vertex and is parallel to the sensor plane is given in Equation 5.3.

After we know how to construct planes, we move on to finding the distance between two parallel planes. Mathematically, the problem of finding the distance between two parallel planes reduces to the problem of finding the distance between a point that belongs to one plane and the other plane. The formula for calculating the distance between a plane and a point is given in Equation 5.4 where (x_1, y_1, z_1) are the coordinates of the point from the first plane, while a, b,c and d come from the equation of the second plane.

$$D = \frac{|ax_1 + by_1 + cz_1 - d|}{\sqrt{a^2 + b^2 + c^2}} \tag{5.4}$$

5.2 Evaluation of the algorithm

The obstacle avoidance algorithm above is designed to combine global and local motion planning to reach the destination. We consider that the user is following the suggested A* path to the destination and hence this algorithm cannot be tested until the entire navigation system as designed in the previous chapters is implemented. In order to test the Obstacle Avoidance as an independent component, the algorithm has to be implemented independently from the A* algorithm and in that case will not guarantee neither arrival to the destination nor the shortest path and will be reduced to the approaches that use only local planning that were presented in Section 4.3.3.

The efficiency of the algorithm has to be evaluated for these scenarios:

- When there are no obstacles along the planned A* path.
- When there are one or more obstacles in front of the device – in the middle of the suggested path.
- When there are obstacles in the sides of the device and the user can walk between them.
- When there are obstacles in the sides of the device but the distance between them is too small for the user to walk in between.
- When the user is facing static vs moving obstacles.
- When the A* path is completely blocked at some point or the user moves away from the suggested path so that the shortest path based on the new starting point (current position) has to be calculated.

6 Discussion

In the previous chapters we designed an indoor navigation system based on inertial motion tracking and depth perception sensors. We have used these sensors on board Google's Project Tango, but in principle, this system works with any device that incorporates depth and motion tracking sensors and is tied to processing units that allow the system to run in real time. The task of the designed navigation system is to navigate the user through the shortest path from a starting point to a destination without colliding with obstacles during traversal of the path and providing real-time guidance. Like any other navigation system, the indoor navigation system we designed is expected to:

- localize the user in the space where s/he is to be navigated,
- localize the chosen destination within the same space and
- guide the user through (the shortest) path between the two.

We added an additional requirement to our system: guide the user to the destination without running into obstacles, because we take into consideration visually impaired users and robots.

We have divided this system in three main components:

1. The Indoor Map,
2. The Navigation Algorithm and
3. The Obstacle Avoidance Algorithm.

The main advantage that the motion tracking and depth perception indoor navigation system has is that all parts of the system can be achieved with the one device: The 3D mapping is done with the depth sensor and the camera, the shortest path is computed by the processor based on the map, the positioning and tracking is done with inertial motion tracking and depth perception sensors and the Obstacle Avoidance is achieved with the depth sensor alone. This makes the device self-contained and the architecture independent from external technologies.

The main outcomes that we got from the research and design of this navigation system are:

- There is no indoor navigation system with motion tracking and depth perception sensors in use or any work that describes or designs such system.
- The entire system has many complex parts that need to be completed and integrated before we can have a complete system, and that will be a laborious process. To make it easier to implement a navigation system with motion tracking and depth perception, we laid out a road map with the elements that have to be completed in order to achieve such a system and mapped work related to all of those elements along the

way. We achieved the detailed workflow by coming across challenges while trying to implement the system ourselves. Therefore, we can say that we discuss practical challenges in detail, not theoretical ones.

- In order to create the Indoor Map that we need for indoor navigation, we have to, either focus on the implementation of a new algorithm that achieves mapping in one go and the produced map will not need post processing, making the work also usable to other people with the same intentions (like [20] and [29] did), or scan the environment with current tools provided with Project Tango and then focus on post-processing with 3D modelling tools – this will contribute to a sequence of procedures that other parties can use for their purposes of indoor scanning.
- It is straightforward to compute the A* path as Unity will do most of the work once the indoor mesh is ready, but it will be more problematic to achieve accurate positioning and tracking since the sensors are prone to noise and errors that accumulate with time, thus decreasing the accuracy of positioning (and tracking since tracking is simply a track of positions). The A* algorithm divides the indoor mesh in polygons and the position of the user can be approximated to the closest polygon. The comparison of the current position to the position in the map can be done by searching through the immediate neighbor polygons of the previous position once the localization of the user in the start position/starting polygon is done.
- Obstacle Avoidance is the hardest component to achieve because it is required to run strictly in real time (otherwise the user will collide with the surroundings and possibly get hurt). But it also encompasses a lot of computations that have to be performed on large amounts of incoming data, which slow it down. We need to handle the issues of a large point cloud of depth data coming in each frame, the segmentation of obstacles from the point cloud and then obstacle avoidance. It turns out that passing by a chair without colliding with it is problematic when a computer sees only data points in 3D space. When obstacles are present, this component will run all its procedures to avoid obstacles, but if the path is free most of the time, which we expect when environments are 93% free space [29], these procedures do not have to be run many times. This component can also be used independently from the system but it will only help in avoiding collision in indoor environments and will not guarantee that the destination will be reached, not to say guarantee the shortest path to the destination.

The Navigation Algorithm and the Obstacle Avoidance components ensure that the user is guided through the shortest collision-free path to the destination. We are left with the problem of running the system in real time. To keep the system real-time, the following operations have to be completed before the next update of the depth sensor:

1. Data collection
    - from the depth sensor and
    - from the inertial motion tracking sensors.
2. Localization and/or tracking

3. Plane fitting
4. Plane segmentation – Graham's scan
5. Plane construction for obstacle avoidance
6. Collision testing.

Since the update rate of the depth sensor is 3Hz (3 frames per second) that means that these computations have to be finished within 0.33 seconds and repeated each frame.

In addition, every few frames instructions are given to the user based on the incoming data. Obstacle avoidance instructions take priority over navigation instructions because keeping the user from colliding with obstacles is more important than getting to the next step in the navigation process.

If the user is not following the path suggested by the Navigation algorithm or that path turns out to be blocked, the A* needs to be recalculated. Since A*'s weakness is its computational complexity, trying to recompute the A* and the rest of the operations at once might hinder the system from running in real time when those situations occur. In such cases the user will be required to pause until the A* is computed so that the whole computation power is dedicated to computing the new path. Dedicating the computation power to A* will make the computation faster so the user will wait for a shorter amount of time and after it is completed the computation power will be dedicated to the rest of the operations, taking time as usual.

In order to be able to know whether the proposed system is real time or not, the whole system has to be implemented and its efficiency tested. Until then, we can only give suggestions how the system could be simplified if the computations take too much time. We can:

1. Reduce the tracking to every third (or more) frame.
2. Reduce the quality of obstacle representation by reducing the amount of incoming depth data that is processed.
3. Reduce the update rate of the depth sensor.

We recognize that trading the quality of tracking and representation of obstacles for computation time degrades the value of the system since while we are trying to get instructions for possible collision to the user faster, we are also risking not representing all the obstacles or not representing obstacles well, which will result in the user colliding anyway. We have designed the navigation system and obstacle avoidance algorithm with real-time guidance in mind but how fast the system runs and whether it runs in real time or not will depend not only on our software but also largely on the underlying hardware and on other concurrent processes that are running on the same hardware at the same time. Currently, we are using a platform that offers a 2.3 GHz quadcore NVIDIA Tegra K1 CPU with 192 CUDA cores and 4 GB of RAM and the implementations should make the most of the underlying hardware, but on the other hand, we also have a depth sensor that updates at the low rate of 3Hz. Different platforms will provide different answers to the "real-time-ness" question and while in some platforms the system will run in real time, further optimizations might be required for others.

6.1 Limitations

While we have studied the programmatic part of the indoor navigation system that we designed in detail, the usability and human interaction part have been left out. No matter how efficient the software might be, if the users have difficulties interacting with and learning the tool, the solution will not be accepted well and perhaps not used at all. In need of an indoor navigation system are users who can see and user who can not, just as well as robots, and the three have different needs and demands from the system and use the device in different ways. With the exception of a few, most of the works that we have analyzed in order to produce this work, have implemented and tested their solutions in robots. We must remember that there will be a difference between giving instruction to a robot and to humans, there will even be a big difference between giving instructions to a blind user and a user who can see. Therefore, while the computational parts of previous works can be adapted completely for Project Tango, the instruction part cannot. This is because while it is very easy to guide robots to walk straight or turn precisely 45 or 60 degrees, it is quite difficult to get people to achieve the same and even more so when they are blind. While you can give instructions to a robot programmatically every two seconds, with people you always have to ask questions such as:

- How often does the user want a status update?
- How to keep the user deviating too much from the suggested paths?
- How to communicate the message/instruction? By sound, vibration or voice?

7 Conclusion and Future Work

Navigation is the act of guiding a person, vehicle, plane, ship or robot from one point to another. Navigation in indoor environments is an unsolved problem. Compared to other technologies that have been used in the past to solve the problem, such as Wi-Fi/WLAN, Bluetooth or infra-red beacons, RFID tags and laser scanning systems, a device with motion tracking and depth perception sensors on board has the advantages of being self-contained, having lower cost and being more accurate in obstacle recognition and in determining the floor in which the user is in. Despite these advantages over the other candidates for use in indoor navigation, we have demonstrated that the implementation of an indoor navigation system with motion tracking and depth perception has a long way to go. We have designed an indoor navigation system using Project Tango's motion tracking and depth perception sensors that helps the user navigate from a starting position to the destination through the shortest path while avoiding the obstacles along the way and running in real time. By trying to achieve such system ourselves with the objective of testing its feasibility, we have run into challenges that cannot be resolved right away. Those challenges have enriched the design of the system with great amount of detail – practical details derived from experience, not hypothetical ones. We take into consideration 3 categories of users: sighted people, visually impaired people and robots.

By analyzing related work in the field we have noticed that no indoor navigation system based on motion tracking and depth perception has been implemented and tested yet, neither has it been described or designed in its details. A lot of the work that makes use of motion tracking and depth perception focuses on achieving Simultaneous Localization and Mapping (SLAM) by making use of Microsoft's Kinect, but not all these works are real-time since Kinect does not have storage and processing units, hence the data was gathered first and processed later. There are also two works that successfully use Project Tango in creating 3D indoor maps. The field of obstacle avoidance has been well-studied, but generally as an independent component and not in the sense of its use in indoor navigation and thus does not guarantee that the user will reach the destination, let alone offer the possibility of achieving the shortest path between the start and destination. But by putting these works together, and filling in the gaps between them as we have suggested in our work, an indoor navigation system with depth perception and motion tracking that ensures the shortest path between the start and destination while avoiding obstacles and remaining online, can be achieved. We present a workflow for indoor navigation and map the work that has been done in the field to their respective categories in that workflow.

The indoor navigation system we designed has three core components:

1. The Indoor Map: is the 3D representation of the indoor environment in which the user is to be navigated. It is achieved by scanning the indoor environment with the depth and motion tracking sensors as well as the camera. Project Tango has available tools for scanning, but the meshes that are produced require extensive post-processing with 3D modelling tools. [20] and [29] offer two algorithms that successfully use Project

Tango to map indoor environments. The map is used to localize the user, the starting position and the destination as well as to compute the path between the starting position and destination.

2. The Navigation Algorithm: uses the indoor mesh provided by the previous component to compute the A* shortest path between the start and destination, to track the user in the indoor map and to guide the user through the path. To achieve this component we deal with issues such as efficient sensor integration to achieve positioning and tracking since the sensors are prone to error accumulation and noise, and the computational efficiency of the A* algorithm and its use in dynamic environments.
3. The Obstacle Avoidance Algorithm: the A* path computed by the Navigation Algorithm is based on the static map of the indoor environment, but with the passing of time, objects are moved in the scene and those objects will represent obstacles when positioned in the way of the user. Since the Navigation Algorithm does not prepare the user for these obstacles, the blind user or the robot will collide with them. The depth perception sensor is used to sense the changes in the environment and replan the paths of the A* path that are affected by the obstacles. To achieve this component in real time we have to be able to: deal with the large amount of incoming depth data, model obstacles from the gathered data on the go and then use the models to replan the path around the obstacles.

Besides dissecting the designed navigation system to its details, an obstacle avoidance algorithm based on the depth perception sensor is proposed. This algorithm combines global and local path planning to achieve a collision-free path between the starting point and the destination chosen by the user. Used independently, global path planning has the advantage of finding the path between the start and the destination because it assumes full knowledge of the environment, but also the weakness that the path cannot be changed if the scene between the start and destination has changed, presenting obstacles along the way. On the other hand, local path planning does not have full knowledge of the environment and rather sees only what is in the range of the sensor at that point in space and time. Its advantage is that it can plan paths that avoid obstacles that are in the range of sight, but the weakness that without having full knowledge of the environment, arrival at the destination cannot be guaranteed (since the device will not know where the destination is). These weaknesses can be overcome with combining the two approaches: the path between the start and the destination is computed with the global approach, but when obstacles along the path are presented, the local approach is used to replan the affected path.

The proposed algorithm works as following: after the point cloud data is gathered from the depth sensor, it is processed to segment obstacles from the scene. If the path computed with the global approach is free, the user is instructed to move along the free path. If the path is blocked by obstacles, we represent these obstacles through their convex hull and then construct planes that are normal to the floor around their extremities. Considering the user as *enclosed* in in a virtual bounding box, we test for collision between the sides of the box with the constructed planes. Since the constructed planes extend beyond the obstacle, we can test if the user is going to collide with the obstacle extremities if s/he user keeps moving in the same direction from a distance. In the case when the constructed plane is parallel to the front side of the box, which is the side in

front of the device, the user is navigated around the closest corner of the obstacle hull. The algorithm is updated for each frame of the depth sensor and at all times the incoming data from a few successive frames is taken into consideration in obstacle segmentation and path planning around the segmented obstacles.

The number of frames that are taken into consideration and the trade off between quality of representation and processing speed that might be required to be made in order to achieve a real-time system, will be experimented with when the algorithm is implemented. Implementation and testing of the proposed algorithm are the first tasks to be completed as future work. This component is the most computationally expensive of all since when objects are in front of the sensor a large number of points will have to be be processed several times each frame that the sensor is updated in order to achieve: plane fitting, plane segmentation, extremity-finding computation, plane construction, collision detection and communication of instructions, but if the environment is composed of 93% free space as the experimental results from [29] suggest, these procedures will not have to be repeated very often, allowing the system to run in real time.

To test the proposed algorithm independently of the system, only the local path planning approach has to be implemented. As an independent component, its efficiency in avoiding obstacles has to be tested, but independent from the global path planning obstacle avoidance will not help the user reach a destination. We are interested in finding out whether combining local and global motion planning will allow achievement of collision-free paths that are also the shortest paths to the destination. Therefore the next step will be to implement the complete algorithm, including the global path planning approach. In order to have a path to a destination within an environment we will have to have a map of the environment, an algorithm that computes the path to the destination, positioning and tracking and lastly instructions to get to the destination using the computed path. After all these are integrated, we will have the indoor navigation system we have designed here. Next, the usability of the system has to be worked on, since different users (sighted and visually impaired people and robots) will have different requirements from the system and will use it in different ways. All these issues are open for future work.

The navigation system and the obstacle avoidance algorithm we have designed here are designed with real-time guidance in mind, and we have treated challenges such as the trade offs that have to be made in case the implementations based on the designs are not running in real time. But in order to answer the question whether the navigation system with obstacle avoidance we have proposed runs in real time, evaluative tests have to be made after the system is implemented. How fast the system runs and whether it runs in real time or not will depend largely on the underlying hardware and on other concurrent processes that are running on the same hardware at the same time. Different platforms will provide different answers to the “real-time-ness" question and while in some platforms the system will run in real time, further optimizations might be required for others. It is important to achieve real-time guidance because the users of the obstacle avoidance algorithm will want to know when they are about to collide with obstacles without have to find it out themselves. Therefore, real-time guidance is one of the most important elements to work on in the future.

Bibliography

[1] Kamarudin, K., Mamduh, S., Shakaff, A., Saad, S., Zakaria, A., Abdullah, A., & Kamarudin, L. March 2013. Method to convert kinect's 3d depth data to a 2d map for indoor slam. In *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*, 247–251.

[2] Ferrari Pinto, R., Conceicao, A., Farias, P., & Santos, E. May 2014. A cost effective open-source three-dimensional reconstruction system and trajectory analysis for mobile robots. In *Biosignals and Biorobotics Conference (2014): Biosignals and Robotics for Better and Safer Living (BRC), 5th ISSNIP-IEEE*, 1–5.

[3] Batalin, M. A., Sukhatme, G. S., & Hattig, M. April 2004. Mobile robot navigation using a sensor network. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, 636–641 Vol.1.

[4] MIPSoft. Blindsquare indoor navigation. `http://blindsquare.com/indoor/`. [Online; accessed 01-June-2016].

[5] Sonnenblick, Y. 1998. An indoor navigation system for blind individuals. In *Proceedings of the 13th Annual Conference on Technology and Persons with Disabilities*.

[6] Coroama, V. 2003. The chatty environment - a world explorer for the visually impaired. In *Adjunct Proceedings of Ubicomp*.

[7] Hub, A., Diepstraten, J., & Ertl, T. September 2003. Design and development of an indoor navigation and object identification system for the blind. *SIGACCESS Access. Comput.*, (77-78), 147–152.

[8] Google Advanced Technologies and Projects. 2015. Project Tango is here. Join us! `https://www.google.com/atap/project-tango/`. [Online; accessed 16-December-2015].

[9] Google Advanced Technologies and Projects. 2015. Project Tango technical specs. `https://www.google.com/atap/project-tango/hardware/index.html#technical-specs`. [Online; accessed 09-May-2016].

[10] Microsoft. Kinect for windows sensor components and specifications. `https://msdn.microsoft.com/en-us/library/jj131033.aspx?f=255&MSPPError=-2147217396`. [Online; accessed 16-December-2015].

[11] El-laithy, R., Huang, J., & Yeh, M. April 2012. Study on the use of microsoft kinect for robotics applications. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, 1280–1288.

[12] Oliver, A., Kang, S., Wünsche, B. C., & MacDonald, B. 2012. Using the kinect as a navigation sensor for mobile robotics. In *Proceedings of the 27th Conference on*

Image and Vision Computing New Zealand, IVCNZ '12, 509–514, New York, NY, USA. ACM.

[13] Microsoft. Kinect for Xbox360. `http://www.xbox.com/en-US/xbox-360/accessories/kinect`. [Online; accessed 17-December-2015].

[14] Abd Manap, M., Sahak, R., Zabidi, A., Yassin, I., & Tahir, N. March 2015. Object detection using depth information from kinect sensor. In *Signal Processing Its Applications (CSPA), 2015 IEEE 11th International Colloquium on*, 160–163.

[15] Ghani, M., Sahari, K., & Kiong, L. C. Dec 2014. Improvement of the 2d slam system using kinect sensor for indoor mapping. In *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*, 776–781.

[16] Shashkov, M., Nguyen, C., Yepez, M., Hess-Flores, M., & Joy, K. July 2014. Semi-autonomous digitization of real-world environments. In *Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES), 2014*, 1–4.

[17] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., & Fitzgibbon, A. 2011. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, 559–568, New York, NY, USA. ACM.

[18] Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., & McDonald, J. Kintinuous: Spatially extended Kinectfusion. Technical Report MIT-CSAIL-TR-2012-020, Massachusetts Institute of Technology, Cambridge, USA, 2012.

[19] Du, H., Henry, P., Ren, X., Cheng, M., Goldman, D. B., Seitz, S. M., & Fox, D. 2011. Interactive 3d modeling of indoor environments with a consumer depth camera. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, 75–84, New York, NY, USA. ACM.

[20] Schops, T., Sattler, T., Hane, C., & Pollefeys, M. Oct 2015. 3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices. In *3D Vision (3DV), 2015 International Conference on*, 291–299.

[21] Wagner, J., Isert, C., Purschwitz, A., & Kistner, A. Sept 2010. Improved vehicle positioning for indoor navigation in parking garages through commercially available maps. In *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, 1–8.

[22] Li, Y., Zhang, P., Niu, X., Zhuang, Y., Lan, H., & El-Sheimy, N. Oct 2015. Real-time indoor navigation using smartphone sensors. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, 1–10.

[23] Anuradha Jayakody, J., Murray, I., & Herrmann, J. Oct 2015. An algorithm for labeling topological maps to represent point of interest for vision impaired navigation. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, 1–8.

[24] Li, Y., Zhang, P., Lan, H., Zhuang, Y., Niu, X., & El-Sheimy, N. Oct 2015. A modularized real-time indoor navigation algorithm on smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, 1–7.

[25] Michel, T., Fourati, H., Geneves, P., & Layaida, N. Oct 2015. A comparative analysis of attitude estimation for pedestrian navigation with smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, 1–10.

[26] Madgwick, S., Harrison, A., & Vaidyanathan, R. June 2011. Estimation of imu and marg orientation using a gradient descent algorithm. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, 1–7.

[27] Martin, P. & Salaün, E. 2010. Design and implementation of a low-cost observer-based attitude and heading reference system. *Control Engineering Practice*, 18(7), 712 – 722. Special Issue on Aerial Robotics.

[28] Fourati, H., Manamanni, N., Afilal, L., & Handrich, Y. Jan 2011. A nonlinear filtering approach for the attitude and dynamic body acceleration estimation based on inertial and magnetic sensors: Bio-logging application. *Sensors Journal, IEEE*, 11(1), 233–244.

[29] Klingensmith, M., Dryanovski, I., Srinivasa, S., & Xiao, J. July 2015. Chisel: Real time large scale 3d reconstruction onboard a mobile device. In *Robotics Science and Systems 2015*.

[30] iFixit. 2014. Project tango tablet teardown. `https://www.ifixit.com/Teardown/Project+Tango+Tablet+Teardown/28148`. [Online; accessed 09-May-2016].

[31] Google Advanced Technologies and Projects. 2015. API Overview. `https://developers.google.com/project-tango/apis/overview`. [Online; accessed 01-June-2016].

[32] Hart, P. E., Nilsson, N. J., & Raphael, B. July 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.

[33] Google Advanced Technologies and Projects. 2015. Motion Tracking. `https://developers.google.com/project-tango/overview/motion-tracking`. [Online; accessed 05-May-2016].

[34] Google Advanced Technologies and Projects. 2015. Area Learning. `https://developers.google.com/project-tango/overview/area-learning`. [Online; accessed 05-May-2016].

[35] Google Advanced Technologies and Projects. 2015. Depth Perception. `https://developers.google.com/project-tango/overview/depth-perception`. [Online; accessed 05-May-2016].

[36] Project Tango. 2016. Tangoxyzij. `https://developers.google.com/project-tango/apis/c/reference/struct/tango-x-y-zij`. [Online; accessed 18-May-2016].

[37] Project Tango. 2016. Project Tango Constructor. https://play.google.com/store/apps/details?id=com.projecttango.constructor. [Online; accessed 09-May-2016].

[38] Francis, S. L. X., Anavatti, S. G., & Garratt, M. Nov 2011. Online incremental and heuristic path planning for autonomous ground vehicle. In *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 233–239.

[39] Lumelsky, V. J. & Skewis, T. Sep 1990. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(5), 1058–1069.

[40] Unity Technologies. Navigation system in unity. http://docs.unity3d.com/Manual/nav-NavigationSystem.html. [Online; accessed 11-May-2016].

[41] Unity Technologies. Inner workings of the navigation system. http://docs.unity3d.com/Manual/nav-InnerWorkings.html. [Online; accessed 11-May-2016].

[42] van den Berg, J., Lin, M., & Manocha, D. May 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1928–1935.

[43] Google Advanced Technologies and Projects. 2015. Calibrating Your Project Tango Tablet Development Kit. https://developers.google.com/project-tango/hardware/calibration. [Online; accessed 25-May-2016].

[44] Linsen, L. Point cloud representation. Technical report, Universität Karlsruhe, 2001.

[45] Fischler, M. A. & Bolles, R. C. June 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6), 381–395.

[46] Holz, D., Holzer, S., Rusu, R. B., & Behnke, S. *RoboCup 2011: Robot Soccer World Cup XV*, chapter Real-Time Plane Segmentation Using RGB-D Cameras, 306–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[47] Schnabel, R., Wahl, R., & Klein, R. June 2007. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2), 214–226.

[48] Holz, D., Schnabel, R., Droeschel, D., Stückler, J., & Behnke, S. *RoboCup 2010: Robot Soccer World Cup XIV*, chapter Towards Semantic Scene Analysis with Time-of-Flight Cameras, 121–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[49] Google Advanced Technologies and Projects. 2015. Java API Tango Support. https://developers.google.com/project-tango/apis/java/support/reference/TangoSupport#method-detail. [Online; accessed 29-May-2016].

[50] Graham, R. L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4), 132–133.

[51] Blake, A., Brady, M., Cipolla, R., Xie, Z., & Zisserman, A. Apr 1991. Visual navigation around curved obstacles. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 2490–2495 vol.3.

[52] Blum, A., Raghavan, P., & Schieber, B. 1991. Navigating in unfamiliar geometric terrain. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, 494–504, New York, NY, USA. ACM.

[53] Kunchev, V., Jain, L., Ivancevic, V., & Finn, A. *Knowledge-Based Intelligent Information and Engineering Systems: 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006. Proceedings, Part II*, chapter Path Planning and Obstacle Avoidance for Autonomous Mobile Robots: A Review, 537–544. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[54] Papadimitriou, C. H. & Yannakakis, M. 1991. Shortest paths without a map. *Theoretical Computer Science*, 84(1), 127 – 150.

[55] Kamil, F., Tang, S., Khaksar, W., Zulkifli, N., & Ahmad, S. 2015. A review on motion planning and obstacle avoidance approaches in dynamic environments. *Advances in Robotics & Automation*, 2015.