Bachelor's thesis

Information and Communications Technology

2018

Matias Tamminen

COMPACT 3D CAMERA SCANNER

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | 45 pages

Matias Tamminen

COMPACT 3D CAMERA SCANNER

This thesis work describes the architecture of a device that can construct a 3D representation of the recorded area and saving it to an easy-to-use form.

The device uses a depth-enabled camera to construct 3D point clouds in real-time. The Point clouds can be used for measuring distances and surface area. The device uses already existing software and software libraries to accomplish the 3D modelling and visualization. The real-time 3D mapping and modelling is done with SLAM software that utilizes cameras. The SLAM software is open-source and can be acquired from the projects GitHub webpage.

This paper presents two possible use cases for the device; The 3D mapping of rooms and the 3D modelling of objects. The mapping can be used for interior design or for drawing floorplans. The object modelling can be used for 3D printing.

The main components of the device consist of Nvidia Jetson processing unit, Intel RealSense depth camera, and a touch screen. The processing unit receives image data from the cameras and uses it in SLAM algorithms to form a point cloud map. The processing unit also visualizes the map for the user with the touchscreen.

The software architecture of the device consists of several independent software processes. The processes can be divided in to the camera driver, visualization software and several SLAM process. The software utilises hardware acceleration of an integrated graphics card that shared some of the processing load with the processor.

KEYWORDS:

Embedded, 3D imaging, RGB-D camera, SLAM

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2018 | 45 sivua

Matias Tamminen

KOMPAKTI 3D-KAMERA SKANNERI

Tämä työ pyrkii toteuttamaan arkkitehtuuri kuvauksen laitteesta, joka kykenee mallintamaan 3D-ympäristöjä ja tallentamaan helposti käytettävään muotoon. Laite käyttää syvyysnäköä hyödyntävää kameraa kolmeulotteisten pistepilvimallien rakentamiseen reaaliajassa. Pistepilvimalleja voidaan käyttää etäisyyksien ja pinta-alojen mittaukseen. Laite käyttää jo olemassa olevia ohjelmistoja ja ohjelmisto kirjastoja 3D-mallinnuksen toteuttamiseen ja visualisointiin. Ympäristöjen reaaliaikainen mallinnus toteutetaan kameroita hyödyntävällä SLAM ohjelmistolla. Kyseisen SLAM ohjelmiston lähdekoodi on avoin ja saatavilla projektin GitHub websivuilta.

Työssä esitellään kaksi laitteelle mahdollista käyttötarkoitusta; huoneistojen 3D kartoitus ja esineiden 3D-mallinnus. Huoneistojen kartoitusta voidaan käyttää toimisto- ja elintilojen huonekaluston sijoittelun suunnittelemiseen tai pohjakarttojen piirtämiseen. Esineiden 3D-mallinnusta voidaan käyttää esimerkiksi 3D-printtaukseen.

Laitteen pääkomponentit koostuvat Nvidia Jetson prosessointiyksiköstä, Intel RealSense syvyysnäkökamerasta ja kosketusnäytöstä. Prosessointiyksikkö vastaanottaa kameroiden syöttämää kuvaa ja käyttää sitä SLAM algoritmeissa kartan muodostamiseen. Prosessointi yksikkö myös visualisoi rakentuvan kartan käyttäjälle kosketusnäytön avulla.

Laitteen ohjelmistorakenne koostuu useasta prosessista, jotka toimivat itsenäisesti. Prosessit voidaan jakaa kameran ajuriin, visualisointi ohjelmaan, ja useaan SLAM prosessiin. Ohjelmisto hyödyntää integroidun näyttökortin rautakiihdytystä, joka jakaa prosessointi kuormaa prosessorin kanssa.

ASIASANAT:

Sulautettu järjestelmä, 3D kuvaus, RGB-D kamera, SLAM

CONTENT

FIGURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

SLAM	Simultaneous Localization and Mapping
CPU	Central processing unit
GPU	Graphics processing unit
USB	Universal Serial Bus
UI	User Interface
HDMI	High-definition multimedia interface
IR	Infrared light
TX2	Jetson Tegra X2 board
Lidar	Light Radar or Light Detection And Ranging
RGB-D	Red Green Blue Depth camera sensor
RTAB-Map	Real-Time Appearance-Based Mapping
GUI	Graphical User Interface
FPS	Frames Per Second
SURF	Speeded Up Robust Features
g2o	General Graph Optimization
CUDA	Compute Unified Device Architecture

1 INTRODUCTION

As camera technology develops, the quality and features of the pictures taken change drastically. In fact, calling the images we take without cameras can no longer be described simply as “pictures”. The images we record may not be static, but a collection image data, enabling movement much like video. The natural progression here is to ascend the 2D plane of the conventional pictures and include the 3rd dimension in our “pictures”.

The viewing of photos does not need to be constrained to a single viewpoint. Instead they can be freed to “move around” the picture, simulating the experience of being actually there. This technology has already been realized by taking pictures from every possible angle and using them to construct a 3D model of the target. However, this requires multiple pictures to be taken during a long period of time and still needs additional information about the position of the camera at each picture. This process can be significantly expedited by utilizing machine vision techniques and position estimation to construct this 3D view in from stream of image data.

This documentation aims to describe the architecture, design, and implementation of a device able to record static models of the environment recorded by the user. These 3D environment models can be viewed from any angle and distance. They can also be used to measure distances in the 3D space.

2 CONCEPT

The user application and purpose of the device is to provide an easy to use method of acquiring 3D data of structures, rooms, surroundings or indoor spaces. This data can be used for architectural design, event planning or entertainment value. Based on this 3D model of the environment, other software can draw floor plans, calculate volume and surface areas or evaluate structural integrity. Provided the sufficient 3D software, the amount of information gathered from this data is almost equal to being at the actual site. In theory this would be the data type from which all useful documents and files are derived from. The simplest and most widely used format for 3D representation is currently what is known as a “point cloud”. A point cloud is basically what it sounds like, a collection of independent points that represent the detected points of a surface or an object. Although there are many formats for point clouds, they all follow the same pattern of having on single data type, a point, with a position in the x-, y- and z-axis.

One use case for this type of data would be a construction site, where the area needs to be mapped and distances need to be estimated. For example, if there is a need to start construction on a specific spot of the site, this area would usually be measured with tools utilising laser technology and traditional measuring devices. The end result would be a black and white 2D illustration of the target area. This would give the designer or architect the needed info to the scale and dimensions of the world, but lack all other information such as colours, texture, lighting and the surrounding landscape. With a coloured 3D model of the environment, most of the information that could be recorded is present in one package. Most CAD software can already process the point cloud format, and this format is well known in 3D modelling.

2.1 Device requirements

If the use case for the generated point cloud map is construction or visual inspection of structures, the end result must meet certain requirements. There are multiple ways to estimate the usefulness of point clouds and 3D models. The first and most prominent quality is the accuracy of the data compared to the dimensions of the source material. If the error in the distance measured from one point to the other is not smaller that the margin of error allowed for the task, the data is not useful and does not meet the

requirements for that specific use case. In point clouds distance can easily be measured between two points as long as the data uses metric values for point coordinates.

Another metric for data usefulness is its visual appearance when viewed. If the data is so sparse that it is not possible to see the desired features, even after proper visualization, the data is not useful. This means the data cannot be used, for example, as a replacement for pictures. The appearance of a point cloud is directly affected by its density and colorization. Not only does the cloud need to have enough points so that details and shape can be discerned, but the points need to be visualized so that three-dimensional perception is possible. The easiest way to visualize a point cloud is to colour the points so that they reflect the colour of the source material.

Since the requirements are so heavily defined by the use case, there is no single requirement for each property of the point cloud. Two use cases can be highlighted to give some basic requirements for the device.

The first use case for the device could be the measurement of distance and scale in large-scale environments. This would be useful in areas like construction, interior design or path planning for machines. This requires the 3D data to be accurate when measuring, for example, the distance between walls or ceiling and the floor. It would also be useful if the current layout of the room can be seen from the map, so that new layouts can be planned. Objects like tables, chairs and lights could counted and measured from the map. The accuracy requirement can be derived from a simple use case where the object is to map a room for interior design and furnishing. This requires that the resulting point cloud map has the correct scale and can be used to measure room dimensions.

Second use case is the 3D modelling of objects that would be problematic for 3D photogrammetry with traditional cameras. The 3D modelling process can be expedited if there is no need to combine several images afterwards and the resulting model could be estimated during the imaging. The main requirement for this use case would be the accuracy of surfaces and small-scale dimensions generated by the device. There is no clear requirement for the accuracy of 3D models, since this is dependent of the purpose of the model. However, one requirement for 3D models is the consistency of the model, meaning that the surfaces are not disjointed or have large holes.

2.2 Main concepts

First problem that needs to be solved is the perception of three-dimensional shapes with a camera that only has a two-dimensional sensor. Traditional cameras lack the ability to measure the distance from the camera to objects in the captured image. Acquiring the third dimension requires additional means of generating more data that could be used to measure distance. This process has existed for a long time by the name of photogrammetry. Making 3D models with photogrammetry is well researched and advanced [1], and is the main method of acquiring models of large environments. However, this method is limited for a number of reasons. Firstly, the process involves taking a large number of individual pictures from multiple angles. The more pictures taken, the more accurate the resulting model or map will be. The imaging process can be time consuming for large environments, especially if a high level of detail is necessary. Next, the pictures are post-processed either manually or automatically. This process can be time-consuming as well, but more importantly there is no guarantee that the data in the images taken previously has enough information for a complete 3D reconstruction. If an angle was missed, or the lighting conditions were off in the imaging phase, the 3D model is flawed and additional images are required. To solve these problems our image capturing needs extended image data to determine the distance of objects in relation to our current position. Furthermore, the result should be viewable either in real-time or right after the imaging process.

The Second problem comes when combining each image frame so that they form a complete map or a model of the desired target. To do that, we need to know the change in camera position and pose between the frames. Since camera sensor is being moved by a human user, we must derive this information after the fact while keeping the process seemingly real-time. These complex problems have been combined in one concept called SLAM, Simultaneous Localization And Mapping [2]. The Localization here refers to the spatial estimation of position, relative to the current surroundings. The mapping refers to the fitting of sensor data in reference to the position provided by the localization.

2.2.1 Distance measurement and motion estimation

In order to solve our 3D perception problem, additional sensors are needed to get more information. Distance measurement tools that use laser technology are widely used in construction, but they usually only provide a one-dimensional, point-to-point measurements. Distance measurement for 3D sensing requires more dimensions and higher resolution distance data. There are many types of sensors that output three-dimensional distance data. Such sensors include Lidars, RGB-D cameras and stereo cameras.

2.2.2 Available technologies

When the goal is to map large and small areas in a user-friendly way, there are options and possibilities when choosing the 3D sensing sensors. The sensor types can be roughly divided into two groups; cameras, lidars. These can be further divided into sensors of different approaches and technologies.

Cameras offer a broad spectrum of different imaging solutions such as conventional color cameras, thermal cameras, infrared, monochrome, 360-degree cameras or hyper-spectrum cameras. Since the mapping of 3D spaces requires some information about distance and appearance of the mapped area, certain types of cameras should be highlighted.

For example, while RGB cameras do not give any information about distance, they do provide a detailed description about visual features and textures. This is useful when the goal is to have a human-readable map. The image data from the camera can be used to color the points in the point cloud and make the map easier to read. However, it can be hard to estimate movement with a single camera sensor using machine vision. Modern solutions to this problem compromise map density for the sake of accurate movement estimation [3]. A camera-based solution relies heavily on the visual properties of the mapped environment. Detecting movement from the camera image requires that detectable visual features are always visible to the camera. The benefit in using cameras for 3D mapping is the availability and device cost of most color cameras. The actual sensors are small, and current generation lenses allow easy integration to handheld devices.

Lidar sensors rely on the accuracy and speed of laser measurements, multiplied over a large area, usually in an arc up to 360-degrees. The lasers can be rotating or static depending on the Lidar type, but the main benefit is to have multiple points of distance measurements in a span of microseconds. Depending on the lidar, the accuracy of single point can be from a few millimeters [4] to a few centimeters [5]. Lidar odometry can be calculated in several ways, but the common way is usually comparing sensor frames to estimate a shift in position [6]. The draw back of a lidar in 3D mapping is usually the blind spots between the lasers and the lack of color information. The cost of high-end lidars with multiple lasers and long range is substantial compared to cameras of any specification.

Combining cameras with sensors that measure distance can provide a good solution for 3D mapping that compensates for the weaknesses of each individual sensor. Camera sensors that provide distance measurements are known as RGB-D cameras.

2.2.3 RGB-D cameras

RGB-D cameras are sensors that combine a standard color camera with a module that gives per-pixel depth information. The depth information can be derived from several methods, including disparity maps from stereo images [7] or time-of-flight measurements [8]. The resulting camera output usually consists of two separate streams; an RGB image and a depth map. A depth map is a 2D matrix with distance values, usually with the same resolution as the RGB-image. Depth sensing is usually limited by range and sensor noise, depending on the technology used.

Depth enabled cameras are widely used in machine vision [9] and in constructing accurate 3D models [10]. The depth map combined with images from a camera helps in making object detection and feature recognition more reliable, since edges and shapes are easier to find. When used in 3D reconstruction, RGB-D cameras give high resolution models with low noise in measurements.

There are two main ways depth maps can be formed: disparity maps from stereo imaging and time of flight measurements. Disparity maps from stereo images are good in feature rich environments where there are plenty of visual edges to compare to previous frames. Stereo cameras usually have decent range, but can perform poorly in environments

where there only a few visual features visible. For example, walls with a solid color with no recognizable textures offer no way of movement estimation.

Raw distance measurements from time-of-flight sensors can provide quick and reliable way to generate depth maps [8], but are often limited by range and getting depth data for each camera pixel requires a wide-angle infrared projector. The range limitations come from the fact that time-of-flight technology relies on the reflectivity of infrared light. If the surface that the light is projected on is not a good reflector, the distance measurement becomes for unreliable. The longer the distance to the object, the less light is reflected back to the sensor.

It is possible to combine these technologies to compensate for their respective weaknesses. For example, using two infrared cameras and an infrared light projector combines stereo vision and light intensity measurement to provide a more complete depth map with extended range. When doing 3D mapping of open spaces, both of these techniques can be used to estimate camera movement and to model the surrounding objects in to a map.

Figure 1. Example of a depth map, near to far translated as black to white [11].

2.2.4 SLAM, Simultaneous Localization And Mapping

SLAM algorithms solve the mathematical problem of knowing the current position of the view point while simultaneously mapping the area with the sensor data [2]. These algorithms have been developed for various sensors that detect physical objects like Lidars, visual cameras, Infra-red sensors, and sonar devices. There are many examples and variations on SLAM implementations but the basic goals and steps apply to all. The

main challenge is to collect enough information about the surrounding world to build a map and then to localize yourself in that known world map. This can often be a “chicken-and-egg” type situation since you cannot start building a map without a frame of reference (localization) and it is hard to know where you are without surroundings (a map).

The practical applications of SLAM range from robotics automation to visualization of 3D spaces. To create 3D models with a camera sensor, we need a system that can take our raw image data, form 3D snapshots, and build a complete map in real-time. There are various mapping projects for robot navigation that use SLAM methods to define paths and obstacles for autonomously moving robots. Most of these projects aim to map areas in a 2D plane or create a very sparse map, only useful for collision avoidance and pathfinding. To be useful for our application the SLAM system must be able to produce dense maps that give a good visual representation of the environment.

The complete understanding of an implementation of SLAM requires deep understanding about the methods and mathematics behind the concept of SLAM. This paper will focus on inputting data and using the output of the SLAM algorithms, not the actual implementation of the algorithm. Good resource for understanding what components and techniques that a SLAM system includes can be found in the paper “SLAM for Dummies” by Søren Riisgaard and Morten Rufus Blas [2].

In the case of camera-based SLAM, or “visual SLAM”, the motion estimation is calculated from the movement of features in the received image. These features are not the kind that humans see when looking at an image. The features here are unique sections of the image, usually less that ten by ten pixels in size [12].

2.3 Implementation

The fore mentioned technologies and methods are used in the device to produce 3D models from the live camera feed. The sensor output from the RGB-D camera gives the system a three-dimensional snapshot of the current field of view. This snapshot is fed to the SLAM software so it can be compared against previous frames and hopefully have enough common properties that the SLAM can deduce movement in some direction. This dataflow relies on the consistency of camera movement and the similarity of image frames.

The SLAM algorithms handling this camera input are resource expensive by nature, since the image frames contain thousands of pixels that have to be run through matrix comparison functions. If this is to be done in a handheld, compact embedded device, the processing unit must meet certain computational requirements. A good embedded processing unit meeting these requirements would need to have sufficient processing power as well as memory. When doing many large matrix operations in sequence, it is useful to have a dedicated graphical processing unit (GPU) that is designed for such calculations.

Some consideration has to be made to the external interfaces of the board as well. If the plan is to use USB cameras in the system, it is good to have at least one USB 3.0 port on the embedded board to provide sufficient bandwidth for the raw image data. The design also requires a display to show the camera feed and some minimal UI elements, so a HDMI port is also useful.

On the practical software side, the SLAM and visualization software require at least some software overhead. The visualizer would need a lot of graphical libraries and drivers so it is best to choose a platform that can accommodate for many different software libraries. A full Linux kernel and filesystem would give the required software repositories and development environment. Since the hardware is already required to be somewhat powerful, it should be no problem to run an Ubuntu based operating system.

3 DEVICE ARCHITECTURE

Since this paper covers the steps from the concept to the implementation of an embedded device, both software and hardware architectures are required. The high-level architecture will give a general idea of the device operation and components, and the low-level architecture will describe the data flow and interfaces in more detail.

3.1 Hardware

The hardware architecture will describe the used components and devices, and the physical interfaces between them. Some technical specifications are critical to device operation, so they are explicitly defined. Other less important features are left out for the sake of simplification. The main components of the device can be shortly summarized with their main functions.

Table 1. Component list

Component	Description
Nvidia Jetson TX2	The main processor of the device, all other components are connected to this in some way
Connect tech. Orbitty carrier board	Exposes the necessary physical interfaces for other components
7-inch display	For user visualization. Displays the camera feed and gives feedback on device operation.
Intel RealSense D435	Camera module that includes an RGB sensor, two infra-red cameras, and an infrared light projector.

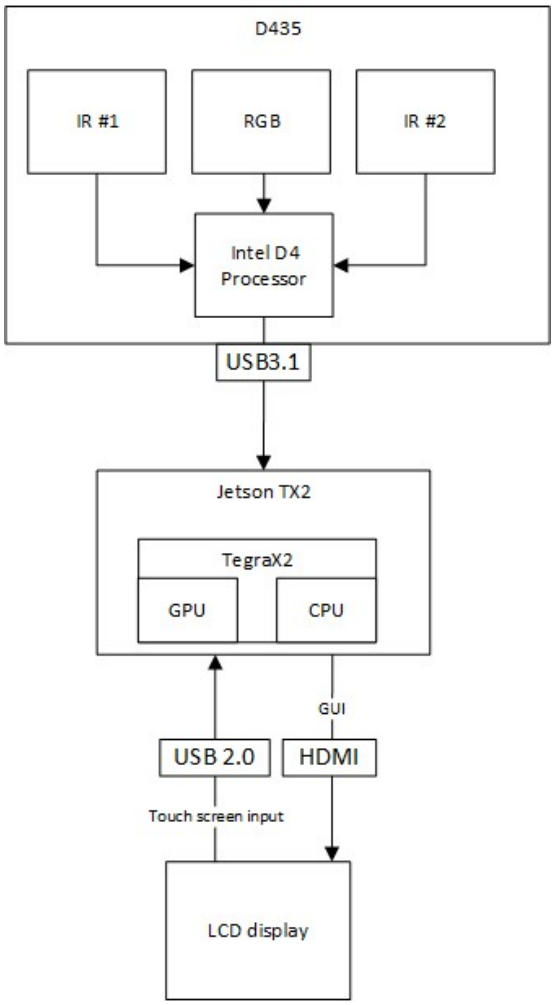

Figure 2. High level hardware architecture

3.1.1 RealSense D435

Intel has developed a line of camera modules designed for machine vision applications called RealSense. In addition to RGB sensors, RealSense cameras have sensors that provide depth data using infrared light projection and stereovision. The latest in the series are the D400 models.

The D435 model includes two infra-red cameras, an infrared light projector and an RGB camera sensor. The stereo IR-cameras and the projector are intended to form a high-

resolution depth map with extended range and good accuracy. The projector emits dots of infrared light that are invisible to the naked eye. The dots form a texture when stretching and contorting with surfaces. When the output from both cameras are compared the distance to surfaces can be estimated for each camera pixel. This technology is referred to by Intel as “unstructured light” [13]. The depth calculation is done on the device, so the output stream is ready to use in machine vision applications. The RGB module provides a video stream up to a resolution of 1920 x 1080 at 30 fps [14]. This is useful for finding visual features with image processing software and provides human viewers with visual feedback. The D435 is well suited for mapping since it gives image and depth data in a wide 90-degree angle.

The total output of the D435 totals to one RGB video stream, two monochrome streams from the stereo-module and the depth map stream calculated on the device. All stream can be taken from the USB 3.1 Superspeed port.

Figure 3. Sensors on the D435 camera

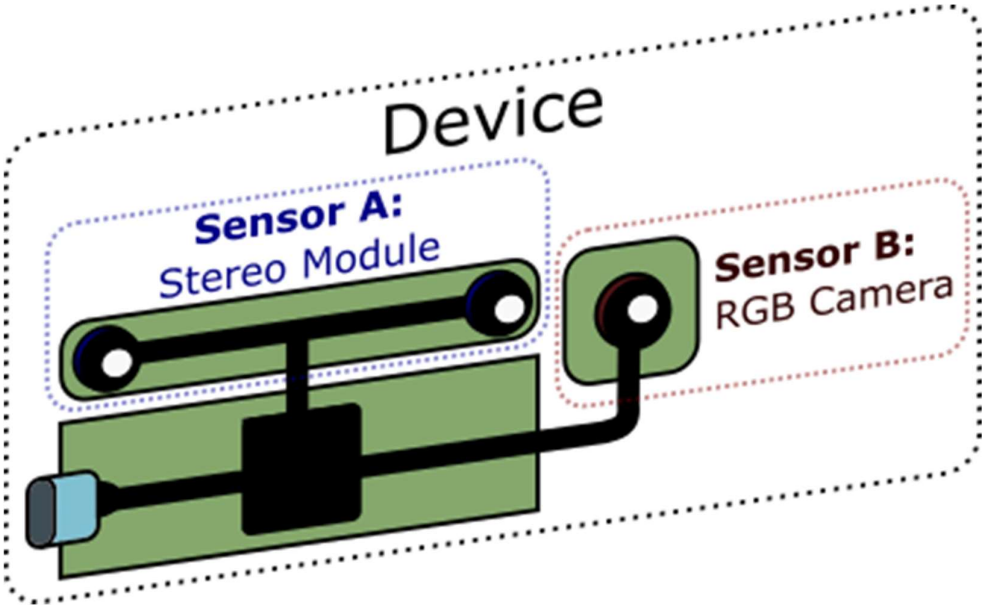

Figure 4. Example of D400 series camera sensors [17]

3.1.2 Nvidia Jetson TX2

The TX2 board can be considered as the main component and the central hub for the data flow. The TX2 has all the required interfaces, data busses and processing power for the device specifications. Most notably the USB and HDMI interfaces are important for most embedded displays.

The TX2 board features the Nvidia Tegra chipset, providing a package of CPU, GPU and memory stacked together. The low profile of the TX2 makes if perfect for small, embedded devices. Without a heat sink or interface ports the board is about the size of a credit card and about thick as a modern mobile phone. What makes the TX2 especially useful for machine vision and image processing is its 256-core pascal-based GPU [15], since the camera-based SLAM performs matrix comparison operations – the prime use case of GPU's.

When we have access to a powerful embedded board like the TX2, it enables us to process much of the operations in real-time. Camera based SLAM algorithms take a heavy toll on any processor, but using the maximum amount of the processing power of

the Nvidia Tegra chipset we can keep up with the required frames-per-second speed of the visual SLAM algorithms.

The TX2 module does not include any interface ports such as USB or HDMI, so an additional carrier board is required. Here we use a board by Connect Tech Inc.

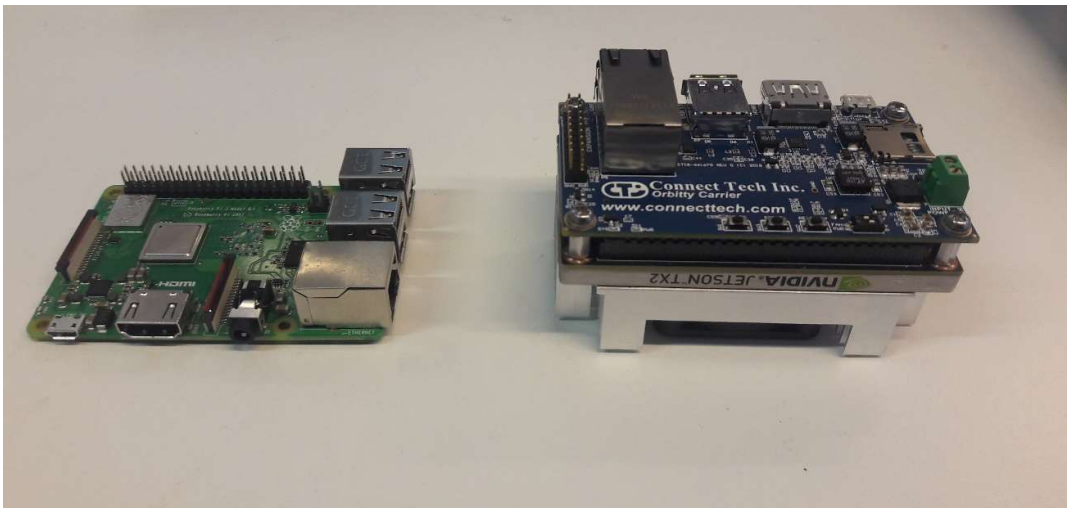

Figure 5. TX2 with Orbitty carrier board (right) size comparison to Raspberry Pi 3B+ (left).

3.1.3 Orbitty carrier board

To make integration and connecting to other devices easier, the TX2 needs connectors for interfaces like USB, HDMI and Ethernet. Carrier boards for the Nvidia TX-series are not that common so there are not many alternatives to choose from. The two main manufacturers are Auvidea and Connect Tech Inc. Connect Tech has a wide selection of interfacing boards and is Nvidia's preferred solution provider, so the board chosen was ordered from Connect Tech. One thing to note when it comes to TX carrier boards is the compatibility for the TX2. Most of the boards on the market were designed for the previous model, the Jetson TX1. While the devices are very similar, there are slight differences in the device tree architecture that can cause problems when trying to use the USB ports. To ensure compatibility, it is important choose the correct carrier board revision. Newer revisions have complete support for all TX2 features, but require a

support software package provided by Connect Tech. The support package usage and board specific support is documented in the Development setup section of this paper.

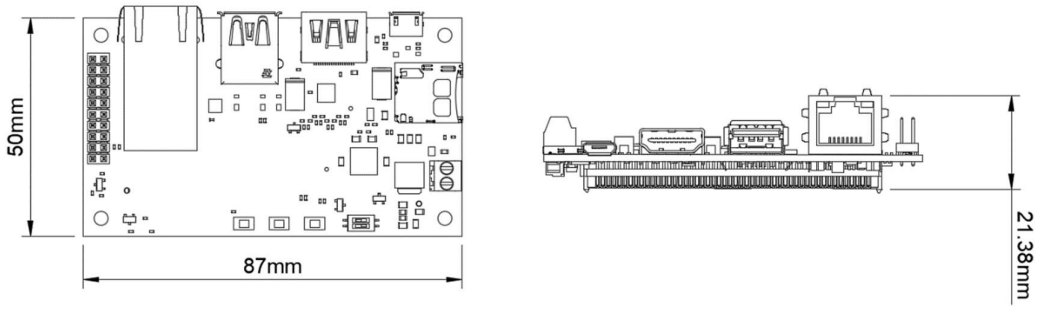

Figure 6. Orbitty carrier board for Jetson TX2

3.2 Software

The main process running on the device is RTAB-Map and its background processes. The three major components are the camera drivers, visual SLAM, and the visualization window. Intel provides their own libraries for their RealSense cameras that are used to grab video feeds from the D435. The latest version of RTAB-Map has built-in support for the RealSense libraries, and uses them if they are installed. The figure below describes the basic data flow and the main software modules.

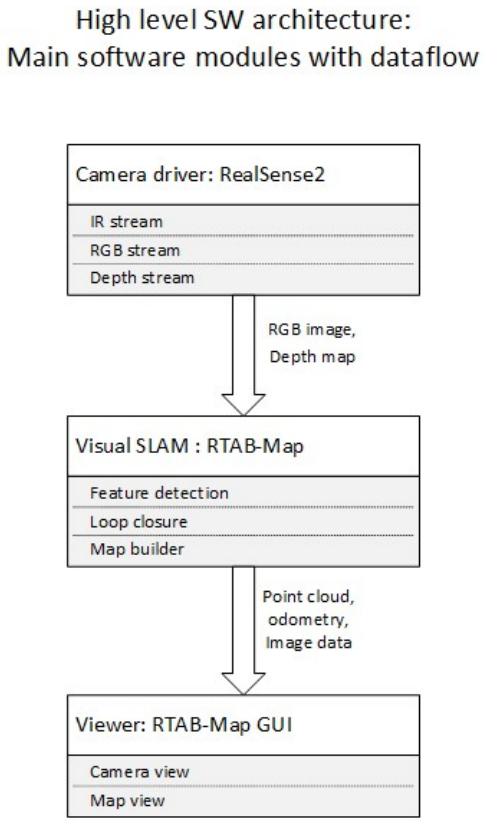

Figure 5. High level software architecture

3.2.1 Camera driver

The camera driver is the RealSense API, and more specifically the librealsense2 library [5]. The API gives access and control to the camera sensors and output streams. The API is divided to two parts: High-Level and Low-Level API. The High-Level API is used to access the three output video streams from the cameras, while the Low-Level API provides the means to change the parameters of each sensor individually. The High-Level architecture was used since the default camera output streams are enough for our purposes.

The librealsense2 library can be found on Intel's GitHub page with installation instructions for the Jetson TX2: https://github.com/IntelRealSense/librealsense.

3.2.2 SLAM process

Finding a suitable SLAM algorithm for modelling objects and mapping indoor spaces takes some searching, since there are numerous open-source projects that aim to solve the visual SLAM problem. All of them have a slightly different approach and varying quality of output. RTAB-Map (Real-Time Appearance-Based Mapping) [16] is an RGB-D, Stereo, and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. The project includes a real-time mapping solution, loop closure detection, visualization tools, and point cloud exporting tools. Everything you need to create 3D point cloud maps is in one package. The SLAM software can take input from various sensors, including stereo cameras, RGB-D sensors and lidars. The software is free and open-source. The examples from the RTAB-Map homepage show a dense and accurate point cloud generation with RGB-D cameras with minimal errors in odometry and accuracy.

Since RTAB-Map is a graph-based SLAM solution, the odometry is constructed as a graph with nodes and edges. The nodes represent the poses of camera at a given time while edges between them represent the sensor data between the pose estimations. This approach to movement estimation allows the process to perform graph optimization, either at run-time or in post-processing. The RTAB-Map SLAM process takes input from sensors like RGB-D cameras and constructs a map based on the calculated odometry. The pipeline here follows the usual visual SLAM architecture.

First, the image received from the camera sensor is divided into features. In the case of RTAB-Map, the feature extraction is done via external library called SURF [18]. These features are the compared to the features in the previous frame and the possible shift in camera position is calculated based on the camera distance to the extracted features. This shift in position translates to a distance and direction of movement, from which odometry can derived. As images are processed by the feature extraction, the unique features are stored in memory to be used in loop closure detection. Loop closure detection is the process of determining if the mapping device has returned to a previously scanned location. This means the device has made a “loop” and the previous path can be optimized to form a closed circuit. The odometry is then corrected by graph optimization, a probabilistic operation implemented by an external library called g2o [19]. Once the odometry graph has been optimized the sensor data is fed to the map generation.

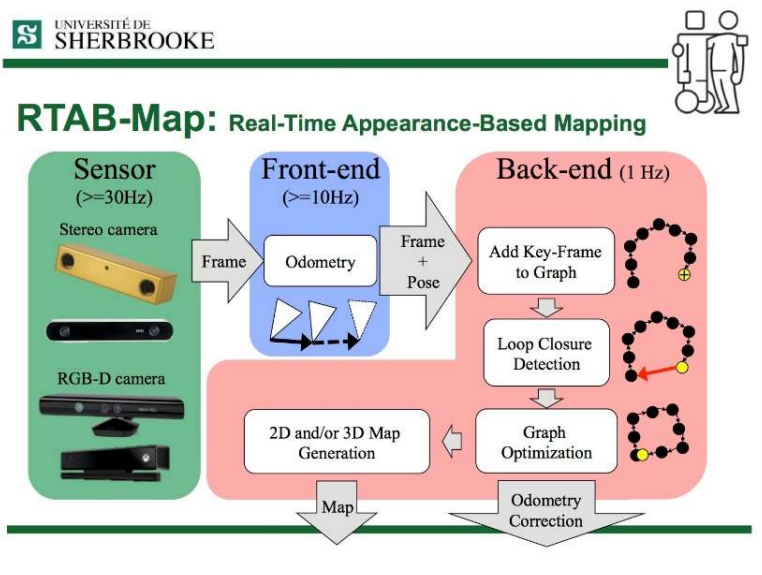

Figure 7. Overview of RTAB-Map pipeline [20]

3.2.3 Visualization GUI

The visualization process is provided by RTAB-Map GUI. All of the code for the GUI is open-source, so it can be modified to have simple controls. The main changes made were removing all menus and buttons that were not necessary for operating the device. Saving files was also made simpler by modifying source code so that pressing the Stop-button automatically saves the point cloud to the storage device.

After the modifications, the GUI displays two views: live camera feed and the colored 3D point cloud. The camera feed helps to aim the device towards the target and keep objects in frame. The 3D view shows how the data is gathered and a sparse representation of the final 3D model.

4 DESIGN

The main design goal was to make the device lightweight and easy to use. Also, to get visual feedback from the 3D algorithms, a small display is needed. This would be mainly for knowing what the camera sees, but also to know if the user is close enough to objects.

4.1 Software design

Most of the device software is kept as is from the project repositories. Some of the source code was modified to disable unneeded dialog options when saving the point cloud. The configuration files of the GUI were modified to make the device easier to use. In addition, the device flashing was done following a specific development setup.

4.1.1 Development Setup

Since the software was to be deployed on separate board that is not a x86 system, the development environment had to be designed with a deployment strategy in mind. Nvidia, the manufacturer of the TX2 board, recommends using their own JetPack development tool for modifying and deploying the board system image. Nvidia also provides the cross-compiler for the 64-bit ARM environment.

The JetPack tool was used to flash the device with a custom system image. This included the latest version of the L4T kernel, an Ubuntu 16.04 desktop environment and u-boot bootloader. The Ubuntu filesystem used came with the JetPack development environment, along with the kernel sources.

The setup is so that the TX2 has a host machine that is used to flash it with new images and compile large binaries. Some considerations had to be made when flashing a custom image on a board with a 3rd party carrier board. The official Nvidia development carrier board has a voltage regulator that provided the voltage for the USB interfaces on the board. The custom board that was used did not have this, so the TX2 had to supply the voltage on its own. This means that the original device tree differs from the one used with the Connect Tech. carrier board. Fortunately, the board manufacturer provides a software support package that includes the correct device tree binary.

When installing most of the required software components, it was necessary to compile them from source. These include librealsense2, RTAB-Map, OpenCV and g2o. The rest of the dependencies were available from Ubuntu repositories and were installed using the Advanced Package Tool program.

4.1.2 UI design

The default RTAB-Map GUI has a lot of functions and views that are not required for basic device operation. The functions needed are start mapping, stop mapping and save the map. The configuration files were changed to only show the needed views and the source code was modified to make the scanning easier with less option to choose from. When the device starts, RTAB-Map is launches automatically and a new database for scan data is created.

4.2 Hardware design

The keyword when choosing the device housing was “compact”. The dimensions were made according to the component dimensions and there wasn’t much empty space. Even though this device is a prototype, the device operation relied on the housing being small and the device lightweight. The enclosure for the device was chosen so that it could be held in one or both hands, but have enough space for all the components. The LCD-display was attached to the top of the casing with adhesive tape.

All of the components were attached to a custom-made mounting plate. The plate was 3D printed with black PLA-plastic. The mounting plate had a detachable part that held the camera in place. Some room was left under the plate to allow air flow for the TX2 cooling fan and heatsink.

Figure 8. Picture of the assembled device

5 TUNING RTAB-MAP

Even though the TX2 has significant processing power relative to its size, the visual SLAM algorithms and real-time visualization can take a toll on the processor. When the SLAM process does not get all the processing power it needs, it cannot meet the real-time requirements, and sensor data is discarded. This can result in sparse point clouds and errors in odometry. These errors can be seen when looking at the final point cloud map. Some walls and floors might be duplicated and loop closures are not detected. This can sometimes be fixed with post-processing, but good results are not guaranteed.

5.1 SLAM Parameters

RTAB-Map has multiple parameters that can be tuned for higher performance at the cost of maximum map size and visualization quality. To achieve the highest quality point cloud map possible, the SLAM process needs to prioritize map density and accuracy. This means that the first priority is keep the SLAM odometry accurate, and find loop closures as often as possible. The second thing to aim for is that the image data received from the camera is added to the map as often as possible to generate the maximum number of points. Both of these goals are resource expensive by themselves, so the other mapping parameters will have to optimized to take as little time as possible.

One parameter to note is the source input rate. Changing the sensor input rate from “as fast as possible” to somewhere around 20 Hz makes a large difference. Performance is increased since a new image is only processed 20 times per second, instead of the full camera frame rate of 60 Hz. The draw back from this is that the odometry tracking is not as smooth. This can result in the SLAM process to lose the connection between the previously seen features and the current view. This in turn can result in a broken map with holes or skewed surfaces. Source input rate must be tuned to be high as possible without affecting the other SLAM calculations.

The main parameters that were adjusted for the TX2 were the point cloud density in visualization, maximum sensor depth, odometry image resolution, increasing image buffer size and disabling memory management.

5.2 GPU Acceleration

From the first tests with RTAB-Map running on the TX2, it was seen that the CPU was under heavy load even after parameter optimization. This resulted in errors in odometry estimation and ultimately in a poor-quality map. Additional processing power was required to run RTAB-Map smoothly. Luckily the TX2 has an integrated GPU that could be used to accelerate the operations in the SLAM pipeline.

A large part of the processing load came from the features extraction operations that are done for every camera frame that passes through the SLAM pipeline. There are multiple choices for feature extractors in RTAB-MAP, but the best results had been tested with SURF. SURF is packaged with the extra modules of OpenCV, and many of OpenCV's libraries have a GPU acceleration API's. The SURF library has such an implementation for Nvidia GPU's called SURF_CUDA.

CUDA (Compute Unified Device Architecture) is the Nvidia API for accessing Nvidia GPU hardware acceleration. Utilizing the CUDA accelerated SURF resulted in a significant boost in performance, improving the odometry and map generation. When the feature extraction was moved to the GPU, it freed up resources from the CPU so that it could focus on detecting loop closures and optimizing the odometry graph.

6 TEST RESULTS

6.1 Test cases

Once the SLAM parameters had been tuned to give a reasonable odometry and map generation, different tests were performed to evaluate the system in different environments. Theoretically, the best results would be observed in environments with plenty of discernable features and good lighting conditions. The metrics to measure map accuracy were narrowed down to odometry accuracy, depth measurement noise and loop closure accuracy

6.1.1 Office

The first test case was in an environment where the device was estimated to perform well; a well-lit office with many features from both depth and contrast readings. The office had desks with clutter like monitors and keyboards, that would offer good features for the odometry calculation and loop closure detection.

Scanning process:

The mapping was started facing a desk with two monitors and a keyboard. When moving around the office the camera was facing forwards at a 90-degree angle to the floor. Each desk was briefly scanned at 1 meter to get all the details into the map. The odometry tracking was stable for nearly all areas of the test case. When looking at the feature extraction indicators during the mapping it was seen that nearly the entire image was divided into features. Only certain objects, like white cupboards were devoid of visual features. The mapping was ended at the starting point, looking at the same features that were at start. This allowed the loop closure to correct any drift in odometry.

Result:

Inspecting the final point cloud, with the traveled path visualized, showed that the odometry was correctly estimated. This was seen from the straight floor, odometry graph, and intact map. However, the map had a lot of floating points that appear to be noise from the office lights. This makes the resulting map look messy and obstructs some of the interesting sections. Looking at the result point cloud, it was seen that desks that

were only viewed from one side had trailing points behind them. This means that the geometry of the desk edges was not estimated correctly.

Figure 9. Complete map of the office

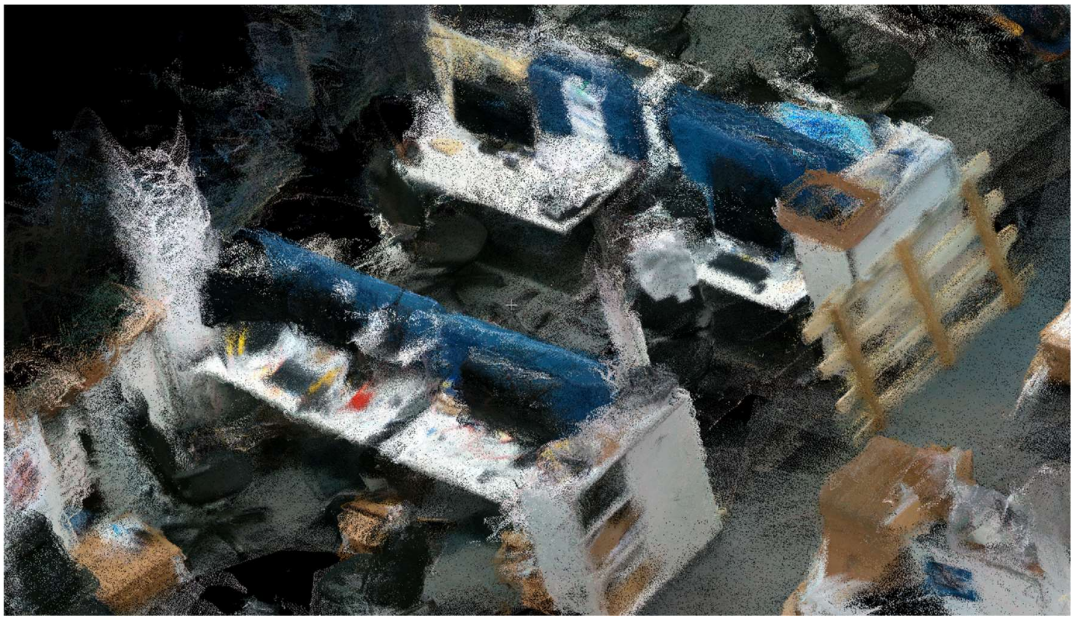

Figure 10. Zoomed in view of the office. Blurry sections can be seen where the depth data was not accurate, or the lighting affected the camera.

6.1.2 Parking garage

Second testing environment was a large parking garage that was dimly lit with fluorescent lights. The area is about 30 meters wide and 60 meters long, and open to sunlight from two sides. The environment did not have a lot of features to use in movement estimation, so the mapping process was estimated to be difficult. The test plan was to use the features available, like cars, parking spots, and pillars, and keep as many features in the frame as possible. The floor and ceiling had no discernable features, and since the walls were so far apart, the path traveled had to be from one cluster of features to the next.

Mapping process:

The mapping was started in an area where the odometry would be easier to calculate. The device had line-of-sight to a nearby car, staircase and far away features like parking spot lines. The mapping started well, but it was noticed that the parking spots did not give any features to the system, despite having a clear contrast difference to the ground. This made the mapping process difficult since it was almost impossible to move the camera away from the car without losing the odometry tracking. The mapping was doable only when keeping one car in the camera frame while looking for the next feature rich target, then moving towards the new features. This limited the mappable areas and the whole parking garage could not mapped.

Result:

The resulting point cloud map had a lot of errors and noise. The map was not useful on its own, and would not be acceptable data if it had to be used in, for example, construction planning. The entire parking garage was not mapped since it was near impossible to map the featureless walls and floor.

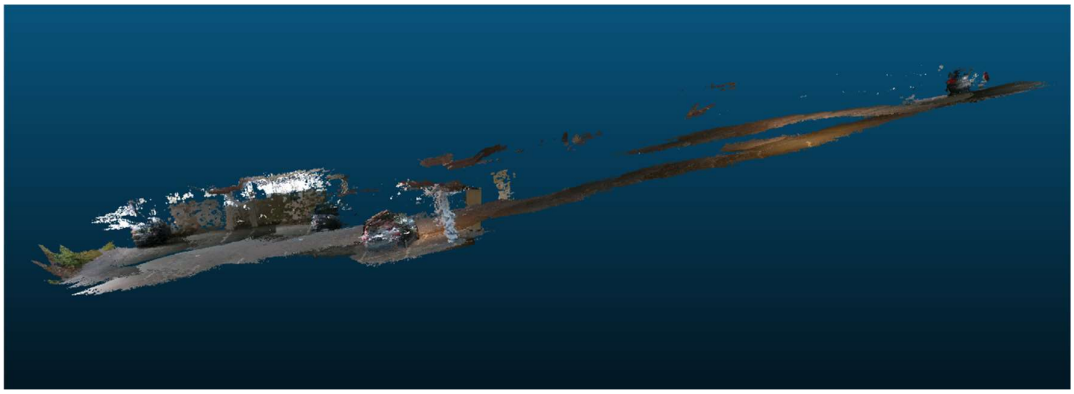

Figure 11. Resulting point cloud map of the parking garage.

6.1.3 Outside

The device was tested for the mapping large structures outside. The plan was to map the walls of large building. The test was expected to be challenging for several reasons. The natural lighting can potentially cause noise in the stereo camera module due to infrared light from the sun. Another concern was the amount of glass on the building ground floor. The north-west wall of the building was almost entirely covered in windows.

Scanning process:

The mapping was started facing three flagpoles. This would be the loop-closure point when finishing the map. The camera was faced towards the walls at a 45-degree angle. The distance from to wall varied during the scan where the wall could not be followed.

Result:

As suspected, the mapping proved difficult. The lighting conditions introduced noise to the map in the form of white points floating in mid-air. Walls with windows had very uneven surfaces and were difficult to map since the odometry would lose tracking. While the point cloud map was not very useful, the odometry and general shape of the building was correct. The odometry path was examined in a 2D graph view and a 2D grid-map

was constructed using RTAB-Map. It was noticed that the loop closure detection and odometry had worked well.

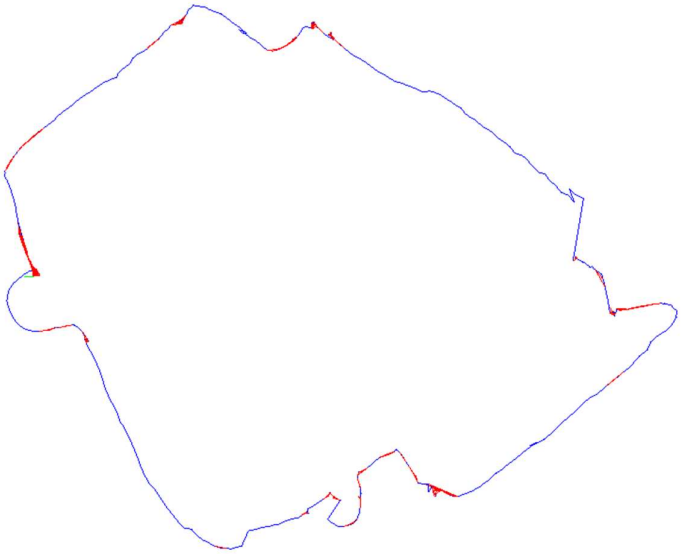

Figure 12. 2D graph view showing the path of the odometry. Red sections mark the parts were loop closures were detected.

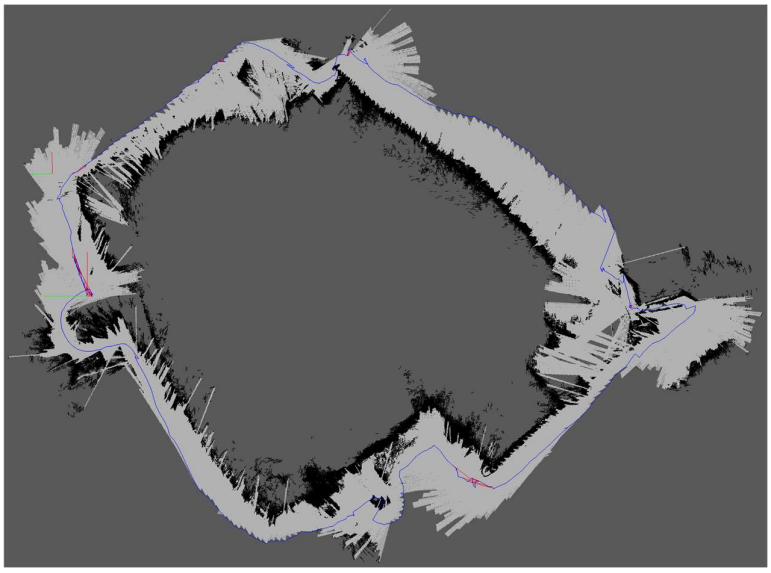

Figure 13. Occupancy grid-map generated with RTAB-Map. Black marks obstacles detected by the depth sensor. White represents empty space.

6.1.4 Staircase

The device was for tested for vertical movement. The purpose was to test how the mapping would work when ascending a flight of stairs. Two types of stairs were tested; a closed staircase and an open spiral staircase. The tests were expected to be challenging for the feature extraction since the areas did not have many features to extract. If sufficient features were found however, the actual odometry was expected to handle the vertical movement well.

Scanning process:

The first test case was a closed staircase with minimal features. The plan was to start from the bottom of the staircase, facing the stairs. The test did not last long, since there was no way map the stairs without losing odometry. The stairs did not offer enough features for the odometry calculation. Surprisingly the steps were not recognized as edges and the white walls were featureless.

The second test case was an open spiral staircase that ascended for three floors. The stairs were climbed to the top with the device aimed at the steps and railing to map the stairs in detail. The odometry estimation was successful since the feature extraction was able to find edges at every step. Because the staircase was quite steep, the camera was close to the stairs, so the depth measurement was accurate.

Results:

The mapping of the spiral staircase was successful, and the result was an accurate 3D model of the steps and railings. The closed staircase was not successfully mapped because there were not enough features for the odometry calculation.

6.1.5 Object modeling

The device was tested for the 3D modeling of small objects. Test were performed on conditions where the system had been observed to perform well. For these tests, the RTAB-Map parameters were tuned for high density point clouds and short duration scanning. The tested objects were an office chair, coffee cup and a keyboard. Small objects were expected to be challenging to map since the odometry needed to be very accurate.

The first test was done on an office chair about 1,2 meters high. The chair was circled twice while keeping the entire chair in the camera frame. The odometry was consistent since there was a lot of features visible. The point cloud was not entirely accurate and had a lot noise. The legs of the chair caused some type of reflection to appear in the depth measurement, which caused the legs to be blurry in the point cloud.

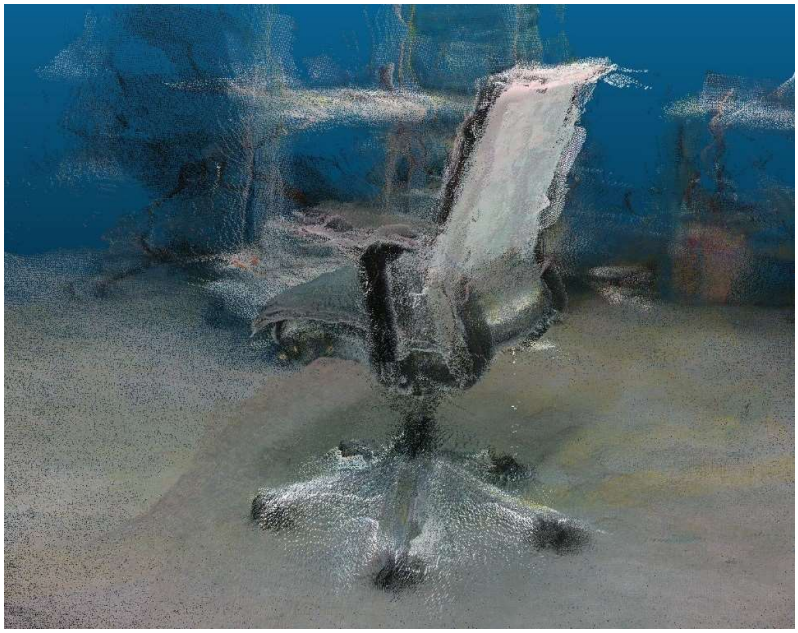

Figure 14. Inaccurate point cloud of a chair

The second test was on a coffee cup. The cup was place on top of a checkerboard pattern to provide additional features and ensure stable odometry. On a small object like a coffee cup, the geometry of the result point cloud was not accurate. This is because of the slight error in SLAM motion estimation.

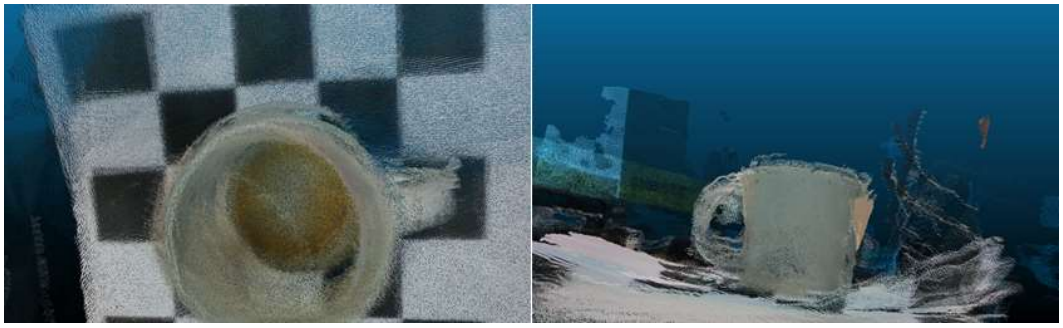

Figure 15. Point cloud of a coffee cup. It can be seen that the model is duplicated.

Another observation that was made when trying to model a keyboard. The depth resolution of the D435 was clearly visible in the point cloud. This resulted in less that smooth surfaces in the 3D model. The resolution could be observed as “steps” in the model. The steps were about 1mm high, this was according to the depth resolution of the camera, which is one percent of the total distance to the object.

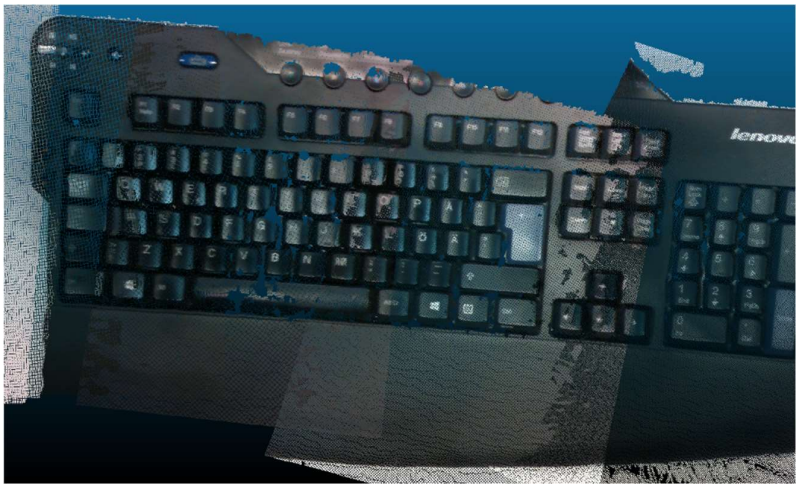

Figure 16. Point cloud of a keyboard. The surface was modelled well except for few holes.

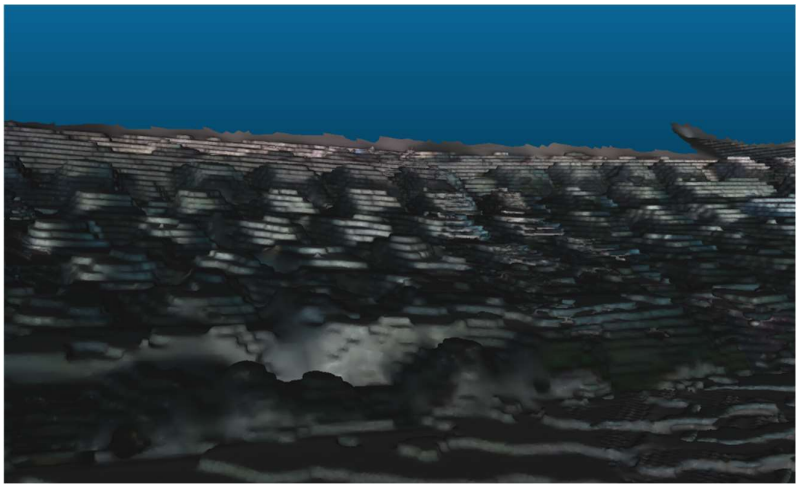

Figure 17. 3D model of keyboard keys from a point cloud. The depth resolution can be seen as "steps" in the model.

6.2 Best conditions

The best conditions for mapping with the device were observed. Environments with plenty of contrast differences and unique visual features were easy to map and gave the loop closure system good reference points. The more features visible to the camera, the more accurate the odometry was.

The system also performed better in smaller environments. The depth measurement noise was significantly lower when the mapped surfaces were closer. The size of the map affected the device performance due to the number of loop closure features in memory and the visualized map size. The device performed best in brightly lit environments where visual features could be easily detected.

6.3 Worst conditions

The environments where mapping was almost impossible were areas where no visual features were present, or where the lighting was not optimal. If the use case was to map an empty parking garage or warehouse, the system would perform poorly. The only way to map the area would be to add features to the environment. This could be done by simply taping checker board patterns to the walls, but there would have to be multiple features visible at all times to achieve accurate odometry.

6.4 Additional notes

Some note-worthy observations were made during the test. Firstly, the camera feed and map visualization were not always rendered smoothly. Especially on longer sessions the camera feed would stutter and freeze. This indicates that the TX2 did not have enough processing power to run RTAB-Map at full speed. Even after the feature extraction calculation was moved to the GPU, the process had trouble keeping the mapping real-time.

At some points during the mapping, the odometry would lose tracking and the mapping was stopped. This would happen when there were not enough features to detect movement. Operation could continue after backtracking so the loop closure could detect

previously mapped areas. This would mean blank walls, the ground, windows and light sources would be challenging to map.

When mapping featureless surfaces was successful, the resulting point cloud map was very uneven. It was observed that the depth data from the camera had a lot of noise when the stereo camera could not detect any features.

7 CONCLUSION

Using the Nvidia Jetson TX2 and Intel RealSense D435 with RTAB-Map to map large areas and small objects is possible, with certain limitations. These can be divided into software and hardware limitations.

7.1 Limitations

Despite the considerable computing capacity of the Jetson TX2, it does not seem to be enough to run RTAB-Map without compromising in map quality. When trying to process images from the camera at rates higher that 10 Hz, the system cannot operate smoothly and camera feed stutters. This means that the feature extraction and odometry calculation done for every frame is slower than the time between inputs. This affects the map accuracy by not adding all the available image data to the map, or by odometry misinterpretation. This is a hardware limitation since the CPU and GPU are not capable of keeping the process real-time.

The D435 camera has limitations of its own. These can be defined as depth quality error and camera image distortion. The depth quality is limited by the unstructured light projection technology. This depth sensing technique poses some problems in areas where infrared interference is present. This causes noise in the depth calculation because the IR cameras can no longer see the intensity of the projected unstructured light dots clearly. The accuracy of this depth sensing technology in also limited in range. The farther the dots are projected, the less light they reflect back. In addition, the farther the dots are projected, the less the images from the two cameras are different. This means that the depth sensing error increases with the distance to the object.

In terms of software used, some limitations are present in the feature detection, odometry calculation, loop closure detection and map construction libraries. For example, the feature detector is not able to find visual features perfectly when the contrast difference is not high enough. A human viewer can clearly see the difference between two shades of grey, but the feature detector does not see any usable features. This is of course affected by the library used and its parameters.

7.2 Indoor mapping

For the indoor mapping use case the main requirement was the accuracy of the room scale and dimensions. The test used to evaluate the results was the office test case. The point cloud was viewed with a software called CloudCompare, which has a tool for measuring distances between individual points.

When mapping an office space 21 meters long and 8 meters wide, the device was able to achieve accuracy of 8 centimetres when measuring the 21-meter distance between the walls from the point cloud. Objects in the room, like desks and cabinets, were in the right place in relation to the walls and each other. The distance accuracy for short distances like these was accurate enough that no error could be measured from the point cloud with this software.

If the goal was to use the point cloud map to design a new layout for the office, the accuracy would be sufficient to make decisions about the placement of furniture, lights and fire alarms. The employee positions could be counted and distances between desks measured. This could be used to determine the space available for more positions, or if the space for employees was too small.

However, at longer distances, the distance and surface area measurement had an error of 0,004 %, which equals to about 8 centimetres on a 21-meter distance. This means that tasks that require precision over long distances would prove more difficult. This would impact surface area calculations that are important when drawing floor maps.

The mapping process itself was straight forward. The mapped area could be mapped by walking around the area, aiming the camera at the desired objects and surfaces. Making sure that all surfaces are mapped is easy as long as the user is familiar with the device operation, and remembers to map all sides of the objects. Simple areas can be mapped by walking normally and facing the camera forwards. Areas with more furniture and clutter need to be mapped more carefully, and require the user to pay attention to the visualization. The acquired map can be seen during the scan from the point cloud visualization UI, so it is easy to know when the map is sufficiently detailed. This makes using the device easy and the eliminates the need to take additional measurements afterwards.

This thesis concludes that this device could be used to map indoor areas for interior design or floor map drawing. If the user is familiar with the device and the area is not too difficult for the feature detection, large environments could be mapped very quickly with tolerable accuracy.

7.3 3D scanning

The second use case for the device was to scan small objects and use the point clouds for 3D modelling. For the device to be used in creating 3D models of real-life objects, the point cloud should be as accurate as possible.

When the device was used to model small objects like a chair or a cup, the point cloud was not as accurate as desired. The odometry was not accurate enough to correctly map the objects and the point cloud was misshapen. The longer the object is scanned, the more errors in the point cloud shape could be observed. Shorter scans showed better results, but moving too fast would cause the odometry to lose tracking.

Using the device for this use case was easy and scanning objects was fast. All that is required to scan a small object is to circle the object with the camera, first horizontally, then vertically. The difficult part is the speed at which to move the camera.

This thesis concludes that, in its current state, this device cannot be used to scan objects into accurate 3D models. The results are not reliable enough to be useful in, for example, 3D printing.

8 FUTURE IMPROVEMENT

8.1 Hardware

If RTAB-Map is to run smoothly and produce the best possible result, additional processing power is required. Both the CPU and GPU should be more capable in order to keep the SLAM process running in real-time while processing as much data as possible. There are many embedded boards with more processing power, but most of them do not have a small size factor like the TX2. One alternative would the new Jetson by Nvidia, the Jetson Xavier.

If a finer resolution is required, a camera with more accurate depth measurement would be needed. Finding a camera with depth resolution higher that 1 percent of total distance would prove difficult, especially a camera with low noise in depth measurements.

8.2 Software

RTAB-Map performs well in visual SLAM using an RGB-D camera, with no prior knowledge of the area or additional sensors. However, the process takes a large amount of processing power to run properly. It could be possible to optimize the CPU load by using multi-threading libraries for the SLAM processes.

In addition, the software does not seem to have a feature to filter the depth data coming from the camera. This causes the map to have uneven surfaces, especially when mapping texture less walls. If the depth data was filtered using some averaging algorithm, the noise could be filtered out.

REFERENCES

[1] McCann, Shawn. "3d reconstruction from multiple images." (2015). Available from: https://pdfs.semanticscholar.org/fc30/577d21df8a360545ef8961ed9235898a9885.pdf

[2] Riisgaard, Søren, and Morten Rufus Blas. "SLAM for Dummies." *A Tutorial Approach to Simultaneous Localization and Mapping* 22.1-127 (2003): 126.

[3] Mur-Artal, Raul, and Juan D. Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras." *IEEE Transactions on Robotics* 33.5 (2017): 1255-1262.

[4] Example of lidar with less than 10 mm accuracy: Leica Geosystems, Leica BLK360 Datasheet. 2017. Available from: https://lasers.leica-geosystems.com/sites/default/files/leica_media/product_documents/blk360/prod_docs_blk360/leica_blk360_spec_sheet.pdf

[5] Example of lidar with 3 cm accuracy: Velodyne LiDAR, VLP-16. Available from: https://www.cadden.fr/wp-content/uploads/2017/02/Velodyne_VLP-16-Puck.pdf

[6] Zhang, Ji, and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time." *Robotics: Science and Systems*. Vol. 2. 2014.

[7] OpenCV Python Tutorials, Depth Map from Stereo Images. 2013 [cited 2018 July 25]. Available from: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html

[8] Y. Song and Y. Ho, "Time-of-flight image enhancement for depth map generation," *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Jeju, 2016, pp. 1-4.

[9] Kerl, Christian, Jürgen Sturm, and Daniel Cremers. "Robust odometry estimation for RGB-D cameras." *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[10] Zollhöfer, Michael, et al. "State of the Art on 3D Reconstruction with RGB-D Cameras." *Computer Graphics Forum*. Vol. 37. No. 2. 2018.

[11] Sean Golden. Intel® RealSense™ Depth Enabled Photography [Image on Internet]. October 2, 2015 [cited June 11 2018]. Available from: https://software.intel.com/en-us/articles/intel-realsense-depth-enabled-photography

[12] docs.opencv.org. OpenCV dev team. Understanding features [Internet]. November 10, 2014 [cited 2018 July 25]. Available from: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning

[13] Leonid Keselman, John Iselin, Woodfill Anders, Grunnet-Jepsen, Achintya Bhowmik, Intel Corporation. Intel RealSense Stereoscopic Depth Cameras. 29 October 2017.

[14] Intel Corporation. Intel Realsense D400 Series Product Family Datasheets. July 2018 [cited September 2018]. Available from: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf

[15] Nvidia Corporation. NVIDIA Jetson TX2 / TX2i System-on-Module Datasheet. November 16 2018, [cited November 25 2018]. Available from: https://developer.nvidia.com/embedded/downloads

[16] Labbé, Mathieu, and François Michaud. "Memory management for real-time appearance-based loop closure detection." *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011.

[17] Intel Corporation. Sensors within device [Image from Internet]. Aug 17, 2017 [Cited July 4 2018]. Available from: https://github.com/IntelRealSense/librealsense/blob/master/doc/img/sensors_within_device.png

[18] Bay, Herbert, et al. "Speeded-up robust features (SURF)." *Computer vision and image understanding* 110.3 (2008): 346-359.

[19] Kümmerle, Rainer, et al. "g 2 o: A general framework for graph optimization." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.

[20] Mathieu Labbé. RTAB-Map pipeline [image from Internet]. Simultaneous Localization and Mapping (SLAM) with RTAB-Map, a presentation by M. Labbe at Université Laval in Québec, 2015. Available from: https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/3/31/Labbe2015ULaval.pdf