Group Project: Simultaneous localization and mapping with Rollo

Rabbia Asghar, BEng, Ernest Skrzypczyk, BSc

***Abstract*— This document deals with the process of implementation of RGBD based SLAM algorithms in a specific humanoid biped robot setup. It describes the theoretical and practical parts of installation of software, hardware modifications, configuration and troubleshooting.**

I. Introduction and Problem Definition

Simultaneous Localization and Mapping (SLAM) deals with the problem of building a map from surrounding environment while simultaneously performing localization in that environment. The problem of mapping in robotics is the representation of environment by gathering and processing information from robot's sensors. Localisation, on the other hand, involves estimating the pose (position and orientation) of the robot relative to the map.

The two problems mapping and localisation are correlated and cannot be solved independent of each other. In order to map a robot's surrounding environment, it is important to know the pose of the robot, where the observations are made by sensors. Similarly, in order for the robot to estimate its position and orientation, sufficient information about the environment should be present. This is why SLAM is often referred to as chicken and egg problem [1]. In 1990s it was found that the combined problem is convergent once formulated as a single estimation problem.

A solution to the SLAM problem is crucial for autonomous navigation and control of mobile robots and involves implementation of algorithms capable of loop detection within the generated maps, especially in scenarios where bigger, global maps are composed of smaller, local ones. Several approaches are present, however SLAM implementations still face certain problems and have room for improvement. An important aspect of the relation between the environment and robot is the computer vision heavily relying on properly calibrated cameras, that allow precise mapping and localization, as well as any aspects related to those.

II. Experimental Setup

A. Rollo

Rollo [2] is a wheeled biped humanoid robot, which has been already used for extented Kalman filter implementation for the localization problem [3]. Initial scenario of the setup was composed of the Rollo, a biped humanoid robot of type (2, 0). In order to implement online SLAM Kinect and RapsberryPi had to be mounted, which resulted in the hardware setup as portrayed by 1..

This hardware arrangement made it possible for the Raspberry Pi to receive raw data from the Kinect using the libfreenect based ROS driver [4] and to send it over a network using ROS structure. Since the goal was to use Rollo to map large environments, a wireless network was used, evermore so, since the communication between Rollo and control host was done using a specifically configured wireless router. This particular setup allows to offload the computations to a graphical processing unit (GPU) host and record maps from even long sessions.

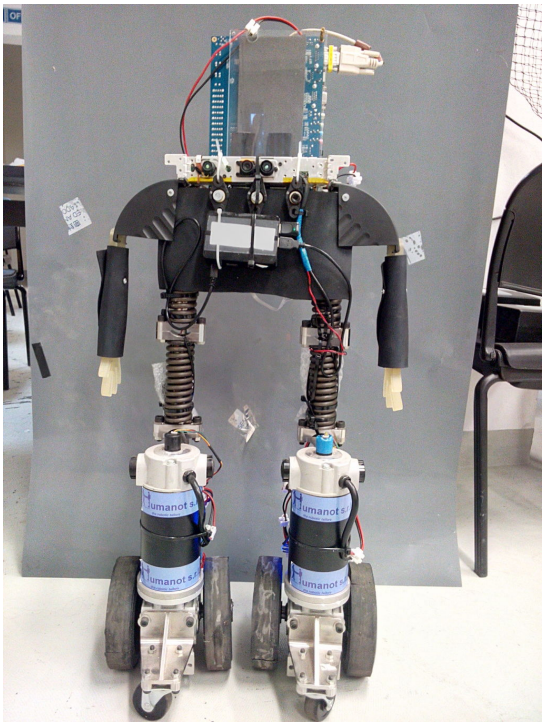


Fig. 1: Installation of Kinect and Raspberry Pi on Rollo.

The host's system needed to prepared to be able to use the GPU for the `rgbd-mapping-and-relocalization` in this case a GeForce GTX970, which is compatible with CUDA [5] compute ability version 5.2 [6]. The host's operating system was `Ubuntu 14.04.4 LTS Trusty Tahr x86_64`ith the kernel version `3.19.0-49`ntel Core i7 4790 with 8 cores running at 3,60 GHz. Additionally an HDD with 50GB free space was provided, which should estimated sufficient for test runs on storing large maps.

Robot Operating System (ROS) [7] was used as the main communication and control framework. The RaspberryPi run the ROS Master [8] node, which then communicated over a network, wireless or cable, with the GPU host as depicted on figure 2. This offload of computation had the primary goal of reducing energy consumption and providing more framerate

Project Supervisor: Prof. F. Mastrogiovanni, PhD C. Recchiuto

for acquisition of data for generated 3D maps. With the performance oriented hardware on the GPU host, processing was basically conducted in realtime, which was tested and achieved when Kinect was directly connected through USB to the host . Futhermore optimization of the system configuration has been undertaken primarily on the RaspberryPi in order to reduce energy consumption and provide more comfortable control over the system as described in IV-C.1.

B. Kinect

Kinect v1 [9] is used as the main sensor for implementation of both mapping and localisation. SLAM relevant components of Kinect are an RGB camera and an infrared projector and sensor to measure depth. In order to not restrict mobility of Rollo, Kinect was mounted onto it and powered by one of the batteries. For this reason, Kinect was disassembled, its USB and power connections were replaced and connected, respectively, to the Raspberry Pi and a voltage regulator set at $12V$. Kinect's cover and base were removed before installing it on Rollo. 4 microphones situated in its cover and the motor in its base were thus excluded from the system as they were not required for the current implementation of SLAM. Kinect's pose was thus fixed with respect to the robot frame.

C. Raspberry Pi

Raspberry Pi was used to establish communication between Kinect and remote GPU host for SLAM using ROS raw and custom messages like *sensor/Msg*. Previously, Kinect driver `libfreenect`nd ROS Indigo were installed and set up on the Raspberry Pi. This setup underwent further modifications to suit the particular implementation and power saving goal. Kinect data was communicated over ROS network with Raspberry Pi configured as master.

Similar to Kinect, Raspberry Pi was powered by the Rollo batteries using a separate voltage converter set at $5V$, even though the FPGA board provided a free USB slot.

III. Algorithms Analysed

For the project already developed ROS based RGB-D SLAM algorithms were studied. The algorithms implemented with Rollo are all available open-source and that use RGB-D sensors.

A. RGBDSLAM v2

RGBDSLAM v2 is developed by team of engineers from University of Freiburg, Germany. Details regarding its installation and source code can be found online. [11].

RGBDSLAM v2 uses a graph-based SLAM system that can be broken up into three modules, as illustrated in figure 4: Frontend, backend and final map representation. In frontend, RGB camera based images are used to extract visual keypoints and depth camera results are used to localise the keypoints in 3D. The transformations between associated keypoints are estimated and false positive matches for keypoints are rejected using RANSAC (RANdom SAmple Consensus) algorithm. The algorithm uses a beam-based environment measurement model to evaluate frame-to-frame estimates and improve the reliability of transformation estimates [12].

RGBDSLAM v2 offers 4 feature extractors methods:

- SIFTGPU,
- ORB,
- SIFT,
- SURF.

Among these, SIRF and SURF are proprietary.

Pose graph is optimised in the backend module using non-linear optimisation to compute a globally consistent trajectory. This trajectory is then used to project the original point measurements into a common coordinate frame, consequently creating point cloud representation of the world. The surrounding environment is then be represented in an alternate and more efficient manner by using 3D occupancy grid maps.

B. RGB-D Mapping and Relocalisation

RGB-D Mapping and Relocalisation algorithm is developed by team of engineers from University of Bristol, UK. Details regarding its installation and source code are available online [13].

RGB-D relocalisation refers to estimating location and orientation of a sensor with respect to a 3D map using a given RGB-D frame. This algorithm is based on key point matching 5 and consists of two main components: a graph matching algorithm and points selection process[14]. Figure 5 shows that first key points are extracted using input RGB-D frame, which are then matched to global reference map using graph matching algorithm. The graph matching algorithm uses pairwise 3-D geometry amongst the key points to overcome the uncertainties caused by visual characteristics and artifact and thus, in result giving robust relocalisation.

Next, points selection process is used to limit the number of keypoints to achieve scalability with the growing map. For this purpose, a non-maximum suppression algorithm is used to select most matchable key points that are evenly distributed over the scene within a volumetric grid. The results are then coupled with standard RANSAC pose estimation and iterative closest point (ICP) pose refinement 5.

C. RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) algorithm is developed by team of engineers from Universite de Sherbrooke, Canada. Details regarding its installation and source code are available online [15].

RTAB-Map algorithm uses RGB-D graph-based SLAM approach which consists of front-end and back-end modules. Front-end deals with graph construction from raw data from RGB and depth camera while the rear-end deals with the graph optimization [16]. RTAB-Map is based on global loop closure detection approach that can deal with multi-session mapping problem, when the robot is moved to a different environment without it knowing. The algorithm uses a bag-of-words approach to determinate if the new image obtained by RGB-D comes from a new scene or one already visited.

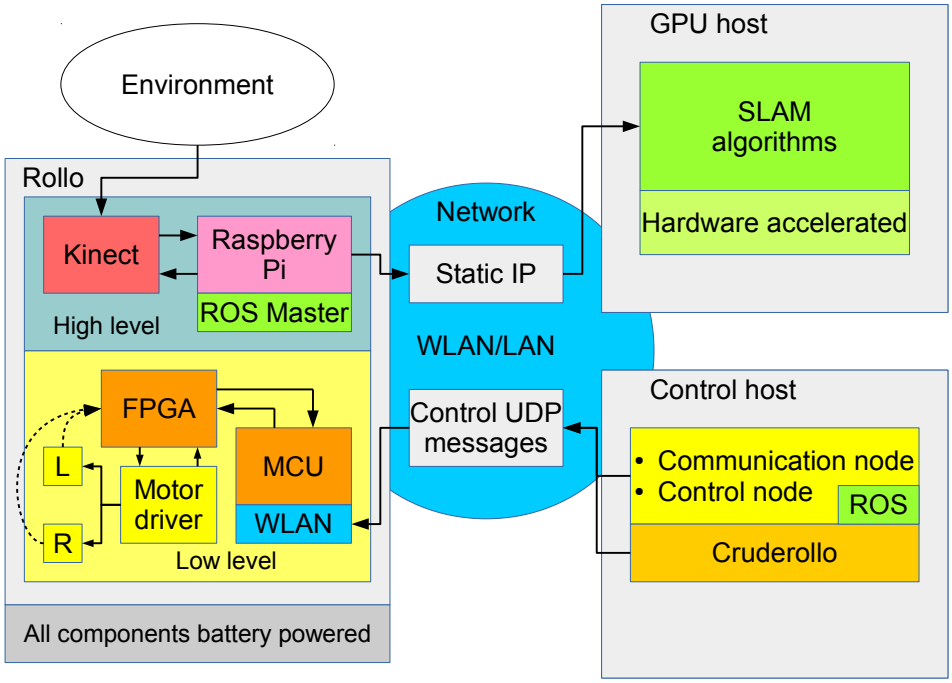


Fig. 2: General network and hardware scheme

Upon determination of loop closure, a new constraint is added to the map's graph, and then the graph is optimized (in the rear-end module) to minimize the errors in the map. In order for the algorithm to run in real time even in large-scale environments, a memory management approach is used by limiting the number of locations used for loop closure detection and graph optimization.

IV. Configuration of the whole system

The initial project goal was to evaluate different SLAM algorithms as described in II. However, because of the limited and a very specific setup, the initial goal was not reached in practice. Based on experience from previous projects and thorough evaluation of benchmarking procedures, only a theoretical comparison was made. The goal shifted towards configuration and calibration of the provided setup.

A. Installation of SLAM algorithms

The experimental setup consisting of the Rollo with the ROS master running on the RaspberryPi needed further configuration and tuning than provided in initial state. This particular setup allows to offload the computations to a GPU host prolonging batteries charge and life. The host's system required further software installation in order to fully utilize the GPU card's capabilities using the CUDA software.

The operating system of the host was a Ubuntu 14.04 LTS x86_64 distribution, which uses the Debian based package manager apt `apt`. The ROS package [13] (rgbd-marl) currently needs specific versions of required dependencies. This resulted in omitting the package management system, which is generally a very risky approach on a system and can render the installation unstable, erratic or even completely unusable. However the decision to test the algorithm was made.

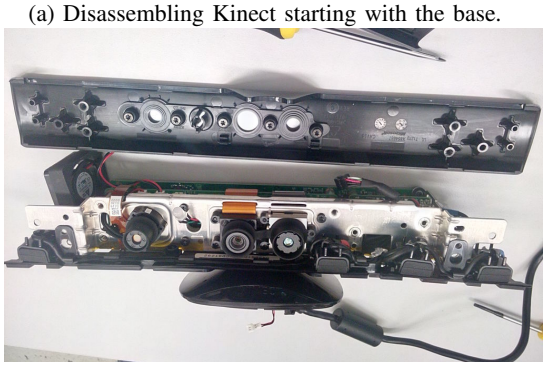
The manual for installation guide for Ubuntu [17] includes several necessary actions that can be dangerous for stability of the system, for example *manual symlinking of system libraries* and *manually forced installation overriding*, possibly already installed, possibly already installed, system libraries. This is particularly bothersome, since most of the dependencies are available directly from the repositories or through 3rd party providers like *pkgs.org*, where even packages from other distributions using tools like alien [21]. Following the instructions following dependencies were installed directly from sources or repositories:

TABLE I: Dependencies of the RGBD mapping and relocalisation algorithm

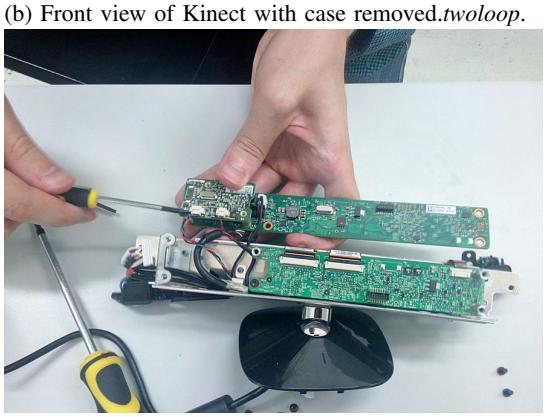
NVidia	CUDA	GLUT	GLEW
zlib	Boost 1.58	OpenNI 1.5	OpenNI 2
Eigen	QT 5.4	QGL Viewer	OpenCV + EM
NIFTI	libsuitesparse		

After setting up the build environment according to the provided example [18] the compilation was failing. Only after manual changes for OpenCV libraries in

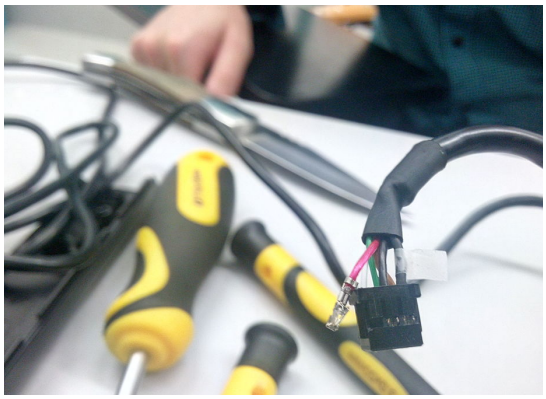
(a) Disassembling Kinect starting with the base.



(b) Front view of Kinect with case removed.*twoloop*.



(c) Disassembling PCBs to access the power and USB connector of Kinect.



(d) Original Power and USB connector of Kinect. These wires were replaced so that Kinect could be installed on Rollo and powered by battery.

Fig. 3: Disassembling of Kinect to modify its wiring so it could be powered by battery.

`CMakeLists.txt` it was possible to properly compile the application. However, when run the process exited with a segmentation fault, which has been tracked using the GNU debugger gdb [20] to the `libQtGui.so.4` library. This particular library should not have been called at all, since the software was compiled using Qt version 5.4, which most likely uses `libQt5Gui.so.5` . The undefined symbol in the library is apparent. Trivial solutions of manual preloading using `LD_PRELOAD`r symlinking the library before compilation and during runtime did not correct the error. Several recompilations have been attempted, the configuration checked each time, without success.

The author of the software suggested running the compiled binary `MappingandRelocalization` with with *sudo* rights. This is either a great simplification on the authors side to get direct access to hardware in `/dev` or a gross error and risk. Without a proper documentation and motivation for this step, it is simply unacceptable to run the binary with system privileges and should be avoided. Hardware damage is just one possibly consequence. For these reasons the focus on this particular algorithm has been dropped. Unfortunately, the installation process interfered with proper setup of other software and this needed to be rectified, in particular the OpenCV package and GL family of libraries, especially mesa related, needed to be reinstalled from the repositories. In the environment setup inside bashrc file, the exported variables necessary for rgbd-marl were deleted.

The other algorithms in focus, RGBDSLAM v2 [11] and RTAB-Map [15], were compiled from github sources. Most of current packages have the capability to be compiled directly using `catkin_make`, so this part of the setup is usually straightforward. Dependencies often can be installed using `rosdep` executable, which is usually incorporated into the build sequence.

Additionally the ORB-SLAM algorithm and others were looked into, but because of compilation or runtime problems, were excluded from the project's focus.

In focus for RGBDSLAM v2 and RTAB-Map were the launcher files, which needed editing for the presented setup. Unfortunately even with Kinect directly connected to the host, RTAB-Map did not start running the algorithm although appropriate topics have been subscribed. None of the provided launchers worked, neither the ones from package nor the simple Kinect launcher from online tutorial [10]. Additionally a custom launcher has been tested, which was basically a reduced version of the default `rtabmap.launcher` with most basic options and without additional sensors selected.

This lead to concentration of efforts on running the RGBDSLAM v2 algorithm. The default launcher `rgbdslam.launch` runs without problems when Kinect is connected directly to the host. Since the host hardware is capable of USB 3.0 transmission speed and the rest of the hardware is set up for performance, the full frame rate achievable from the setup was around $20 \div 25$ fps. For the wireless network connection however, the parameter `subscriber_queue_size` had to be increased from de-

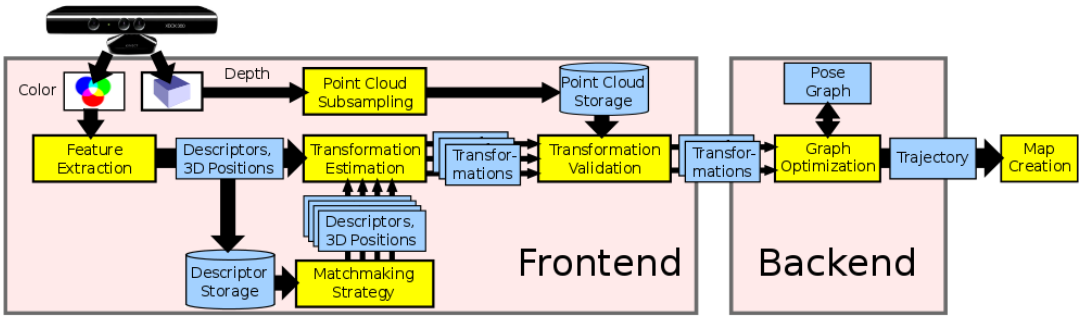


Fig. 4: Schematic representation of the RGBD SLAM v2 algorithm.

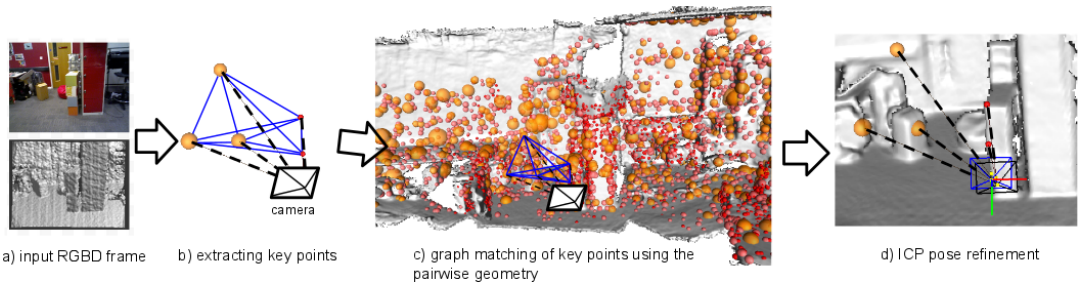


Fig. 5: Overview of RGB-D mapping and relocalisation algorithm.

fault value of 3 to a larger value with maximum tested value 64. The other relevant configuration step was to determine what image and format was supposed be used for the SLAM. The choices provided by freenect driver with default configuration include compression using theora codec for various streams. There are main available data streams are:

- depth – processed ir,
- ir – infrared,
- rgb – mono,
- rgb – color.

The transport variants usually include `image_raw`, but various others can be used as long as the publisher's driver enables those options. All of those are messages of type `sensor_msgs/Image`. Depending on provided operating system and installed libraries as well as codecs, additional ROS software like image_view should be able to display and save it. Right clicking in the view window will save a frame, however for scripting purposes it is recommended to use node `extract_images` or `image_saver`.

An important aspect of the RGBDSLAM v2 algorithm is the capability to use hardware accelerated feature detection and extraction algorithms. Here only the patented SIFT algorithm is hardware accelerated, resulting in SIFTGPU method. Even though in case of RGBD-D Mapping and Relocalisation there was a heavy focus on CUDA, in case of SIFTGPU, as stated by the source, by default GLSL is used and outperforms the optional CUDA for the majority of use scenarios. Because of performance profiled hardware of the GPU host and previous installation of CUDA for rgbd-marl, there was no subjectively noticeable impact between ORB, SURF, SIFT and SIFTGPU, however a proper comparison between the image processing algorithms was not performed.

B. *Setting up the network configuration*

For the provided experimental setup, several network configurations have been taken into consideration and have been tested.

- WLAN
  - Network: Dir510l-4f3c, full mobility, one router for the whole system
  - Network: EmaroLabUnify, reduced mobility, two routers: one for data communication, another for control
- LAN
  - Network: local, heavily restricted mobility, one router for communication between RPi and GPU host

The above network configurations can be combined with the following connection schemes.

- Kinect camera connected to Raspberry Pi
- Kinect camera directly connected to GPU host through USB 3.0

In the desired scenario both Raspberry Pi and Kinect were mounted on Rollo. The robot itself was controlled through a control host and sent acquired data from Kinect using the RPi over ROS within the selected network configuration to the GPU host. The GPU host could also work as control host, as long as connection to the mobile router D-Link router was provided, since a static IP assignment has been set on the MCU that used Rollo's WLAN unit. Raspberry Pi run the ROS Master node and the freenect driver. ROS messages of type *sensor_msgs/Image* from topics `/camera/rgb/image_rect_color` and `/camera/depth_registered/sw_registered/image_rect_ra` were subscribed by the SLAM algorithm.

The default setup provided full mobility to the robot, making it possible to map environments outside of laboratory setting equipped only with a mobile router and PC. The connection to the GPU host would of course have to be provided. Technical difficulties however forced using another available wireless network EmaroLabUnify. This provided mobility restricted to the stationary located network, which was sufficient for the project realization however. A more restricted, but less error prone solution was to connect the Raspberry Pi directly through LAN to the GPU host and exchange data over a cable. For all of the above network schemes, the Kinect device could be connected to the Raspberry Pi or directly to the GPU host, however this introduced obvious mobility restrictions and required for the GPU host to have the freenect driver installed and running.

Several problems were encountered in mentioned scenarios. When using wireless network, either DIR510l-4f3c or EmaroLabUnify, there were drop outs of ROS messages. These occurred so often, that the freenect driver running on the RPi had constant crashes. Since the data throughput was very low, another WLAN USB stick was introduced with double data transmission rate. This increased the frame rate of received images on the GPU host, which with the original hardware was around $4 \div 6$ fps, to approximately $8 \div 10$ fps based on the rgbd-slam-v2 provided frame counter. The drop outs continued however, making it impractical to create maps and infeasible to evaluate SLAM algorithms.

The main issue observed was that `camera/camera_nodelet_manager-2` was killed after a certain amount of time after the SLAM algorithm, in this case only RGBDSLAM v2, subscribed to the rosmaster running on the RPi. First troubleshooting included increasing the `subscriber_queue_size` parameter. This did increase the number of frames initially, that is as long as the value stayed below 32 and the network connectivity was guaranteed. The problem was solved by using a LAN connection between GPU host and RPi directly, ergo using a crossover through a router in bridged mode. This led to the new network configuration scheme. The frame rate increased further to $10 \div 12$ fps since the data transmission rate was higher than in WLAN, but it made the initial goals of mapping the environment and simultaneous localization irrelevant. Unfortunately, even before optimizing the RPi configuration by disabling the logging, after the camera nodelet got killed, the logs contained no significant information why. Based on the experience gathered during the project, it is speculated that the bottleneck in form of low transmission rate, which is indicated by low frame rate, is the initial cause of failure. Once the RPi starts sending data, most probably the freenect driver uses a buffer to store raw images received from Kinect. Since it cannot send them out fast enough, the buffer keeps accumulating the data under it overflows causing the camera nodelet to exit.

The table II shows the data transmission rates comparison between different devices and image modes of Kinect. Kinect supports different image modes, with the low level resolution being almost at VGA 640×480 format and the high resolution SXGA 1280×1024 at half frame rate of $15 fps$. For the computations 5 channels of data were used, which is an overestimation in case of high resolution image mode, since depth was processed and always at lower resolution according to documentation /citefreenect-camera.

This comparison explains the experienced network problems. The default mode in which Kinect runs is the low resolution mode. A 500 Mbps device, which was assumed to be the second USB WLAN stick, would have provided sufficient bandwidth.

The main issue with troubleshooting this issue was the time aspect. It takes a certain amount of for the Kinect driver to get started or restarted on the RPi, which runs at 900 MHz. This made the troubleshooting by changing parameters in order to increase the transmission tedious and lengthy. Once the nodelet dies, in order to get the system running in a robust way, the following procedure needs to be followed:

1) Restart WLAN
2) Kill all ros nodes, nodelets and rosmaster on the RPi
3) Kill the SLAM algorithm on the GPU host
4) Restart rosmaster and/or freenect launcher
5) Wait for the freenect driver to be fully up, then start SLAM algorithm

The restarting wireless network part is unfortunately a necessity for a robust restart procedure. On several occasions the Ubuntu OS did not provide feedback about a lost connection, ergo the wireless network status was ”connected”, even though the connection got terminated or at least so badly corrupted that is was unusable. This aspect only made the whole procedure more difficult to follow and longer, making the major part of troubleshooting waiting. In order to speed up the process several scripts have been written for the GPU host and RPi, like the `roskill` utility for guaranteed exit of all ros nodes and nodelets.

Because of frequent switching between network and connection schemes the ROS environment needed changes in order to guarantee proper communication between publisher and subscribers. The necessary variables were `ROS_MASTER_URI` and `ROS_IP`, therefore aliases `ros-rollo` and `ros-gpu` were added, exporting those with appropriate values.

Another main issue was the inability of the RPi normal user *pi* to publish data without having super-user rights. This behaviour was additionally inconsistent. From the technical point of view, only a direct low level hardware access would have required super-user privileges in this kind of ROS and hardware scenario. This particular problem was solved elegantly using systemd's services.

Systemd is an init system used by some GNU/Linux distributions. It has a high level file structure and meta language used to configure complex application scenarios. A service unit can be run as any user, within a limited environment if necessary. In this particular case, based on the configuration scheme the user running the freenect launcher could be changed with minimal effort. Additionally systemd provided extended logging capabilities and an easy interface of control. After proper setup, a simple

Device	Data rates [Mbit/s]	[MB/s]
Raspberry Pi 2 Wifi Dongle [23]	150	18.75
250 Mbps device	250	31.25
TP-link TL-WN823N [24]	300	37.5
500 Mbps device	500	62.5
LAN 1 Gbit/s connection	1000	125
Kinect $640 \times 480@30fps$, 5 channels	368.64	43.95
Kinect $640 \times 488@30fps$, 5 channels	374.74	46.85
Kinect $1280 \times 1024@15fps$, 5 channels	786.43	93.75

TABLE II: Data transmission rates comparison.

Network	ROS_MASTER_URI	GPU host ROS_IP	RPi ROS_IP
WLAN DIR510l-4f3c	http://192.168.0.155:11311	130.251.13.20	192.168.0.155
WLAN EmaroLabUnify	http://130.251.13.99:11311	130.251.13.20	130.251.13.99
LAN Bridge	http://192.168.1.[23]:11311	130.251.13.[23]	192.168.1.[23]

TABLE III: Network configuration for ROS using environment variables.

command `systemctl restart ros` would restart the whole freenect launcher.

Direct connection of the Kinect to the GPU host worked with the SLAM algorithm without any problems. This confirmed that the device itself was working properly and network issues were the main concern.

A possible solution to the data transmission drop outs could be to use compressed images instead of raw data with the help of additional tools like *compressed_image_transport* or simply by configuring the freenect driver to compress the data and send it to the subscriber. This however goes against the purpose of saving energy, since energy is necessary to compress and decompress the images, where the GPU host is of no concern, but the RPi might significantly increase it's power consumption. In retrospect however it would have been a much better approach to fully explore compression capabilities and try to balance CPU and energy usage versus frame rate and data accuracy. Another possible solution could have been to use the `data_skip` parameter for `freenect_camera` node within freenect launcher application to skip frames. Again this was not desired for accurate map generation and therefore not fully explored.

C. RaspberryPi

The Raspberry Pi 2 Model B V1.1 equipped with an ARM Cortex-A7 CPU operating at 900 MHz with 1GB RAM memory run Raspbian as operating system. Several aspects of the provided system were modified for various reasons.

In accordance to listed network configuration schemes, Raspberry Pi was configured to automatically connect to specified wireless networks DIR510l-4f3c or EmaroLabUnify. These were static configurations, however symlinks have been created to allow for faster switching between them.

Since the performance was of no priority for the project, the RPi was run at native CPU frequency. For optimal performance a more minimalistic GNU/Linux distribution like Gentoo or Arch Linux should be installed and properly configured. Because of time restriction this aspect was skipped fully however.

It should be also noted that specific distribution, even though advertised to be optimized for a specific architecture or system, very often because of contradictory goals, like minimalism versus usability and universal solution for majority of users, are not. In case of Raspbian, several system and user level applications are used to autostart programs. Because of the two rosmaster nodes problems described in IV-C.1 all of those possibilities needed to be checked. *The Arch Way* [26] proposes user centrality that puts the user in control *and* responsibility.

1) Energy consumption optimization: One of the goals of the project was to optimize energy consumption of the implemented SLAM solution, in particular the battery life of Rollo was supposed to be prolonged if possible, since the whole setup was isolated and running on Lithium polymer batteries. The provided software and configuration setup already differed from the default Raspbian [25]. Because the Raspberry Pi used GNU/Linux as operating system and since different distribution come with their specific default power profiles, further investigation into the characteristics of Raspbian [25] was needed.

The main aspects of reducing the power consumption on hardware running GNU/Linux include setting, tuning and modification of:

- CPU, RAM and other components power profiles and/or daemons, usually *powersave* governor;
- Specific mount parameters for block or other type mount devices, no *access time updates* for supported filesystems;
- Module attributes related to powersaving options for all relevant hardware devices, for example *power_save* for snd-hda-intel module loaded with today's standard DVI/HDMI devices;
- Medium and low level software and kernel optimization, compiling packages without unnecessary options, similar to *USE flags* in Gentoo Linux, blacklisting unnecessary modules;
- System and ROS configuration optimized for given task performance, setting *vm.swappiness* to $0 \div 10$ for reduced disk usage, saving ROS logs in RAM or disabling them completely;

In the case of Raspberry Pi the following additional modification with respect to the previous state have been made:

- Starting of rosmaster from `~/.bashrc` was removed.
- The root password was set to `rpi`.
- Several modules were disabled:
  - `snd_bcm2835`
  - `snd_hda_intel`
  - `snd`
- Even though the `snd_hda_intel` is not loaded in the headless state, there is no reason for it to be present even when using the GUI over the HDMI connection. The general sound module was additionally blacklisted, preventing any sound modules to load initially.
- Additional entries for the temporary filesystem in `fstab` were added. fstab.diff
- ROS logging was disabled to a certain degree. using environment variables and specific component configuration.
- Set tmpfiles configuration for the log directory to periodically, here every 15 minutes, clean the directory.
- Swappiness in `swappiness` was already set to 1.
- Using the Raspberry Pi configuration tool `raspi-config`
  - Overclocking was disabled, ergo set to *None*.
- Enabled at `multi-user.target` level a systemd service and script for automatic start of the Freenect ROS launcher once the WLAN connection has been established.

Using `~/.bashrc` for starting any application, as it was in the initial state, should be done carefully, since this particular file is called after each login on distribution where Bash is the default shell. This approach resulted in confusion, since there was no checking in the code if a rosmaster node is already running and after using ssh to log into RPi, at least two roscore processes were running, which was speculation for network drop outs and incapability to run ROS properly without super-user rights.

D. Rollo

There was a significant change in the mechanical structure of Rollo. Since the previous state represented by figure 6 the wooden board connecting spring legs was removed. This resulted in increased instability and unpredictability of Rollo. Furthermore the electrical connections to the encoders were removed. Figures 7a and 7b show state of Rollo after on spot rotation in clockwise and counterclockwise directions. The Rollo with mounted Kinect and Raspberry Pi is shown in figure 1 .

From theoretical point of view, those restriction has little significance for SLAM itself. They are just tedious to deal with from the perspective of the user in control.

V. Evaluation of the algorithms

In order to compare the different SLAM algorithms, various benchmark tests were studied. Table IV shows summary of the characteristics of different benchmarks [27].

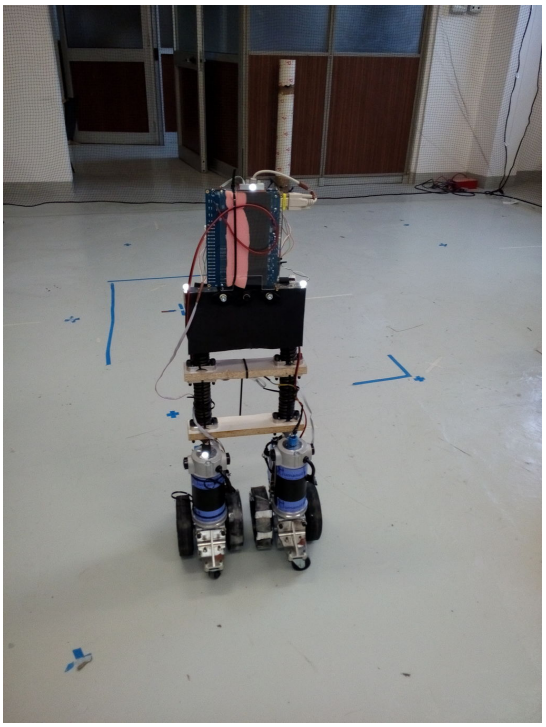


Fig. 6: Rollo with wooden boards.

The most efficient way to analyse the results of algorithm would be to compare both localisation and mapping with ground truth. Obtaining ground truth for the mapping can be difficult and costly. It requires 3D scanner, which was unavailable in the lab facility. However, the ground truth for localisation component can be obtained using the motion capture system. With the assumption that the error in map will be highly correlated to the error of the trajectory for a SLAM algorithm implementation, the ground truth for the localisation alone should also be sufficient for evaluation of algorithm Hence, the benchmark [31] for the evaluation of RGB-D SLAM systems presented by author of III-A was selected, here on, referred as Sturm's benchmark.

The analysis of different SLAM algorithms can be carried out using 3 different methods:

1) Offline evaluation of individual algorithms using public dataset
2) Offline evaluation of individual algorithms using Rollo and its Kinect sensor
3) Online evaluation of individual algorithms

A. Offline evaluation using public datasets

Sturm's benchmark offers a large public dataset containing RGB-D Kinect data and ground-truth (motion capture) data for the evaluation of SLAM algorithms. The advantage of using these public datasets is that different SLAM algorithms can be evaluated without resource restrictions, like access to motion capture system, and then confidently compared.

The public datasets include various groups:

1) Calibration: for the calibration of intrinsic and extrinsic parameters of the Kinect and the motion capture

TABLE IV: Comparison of state-of-the-art RGB-D benchmarks.

Benchmark	Device	Camera Trajectory	Scene Geometry	Global Coordinate System	Number of Trajectories
Meister [28]	Kinect v1	no	ground truth	no	3
Sturm [31]	Kinect v1	ground truth	no	no	39
Zhou [29]	Xtion Pro	computed	computed	no	8
Handa [30]	synthetic	synthetic	synthetic	no	8
CoRBS [27]	Kinect v2	ground truth	ground truth	yes	20

system

2) Testing and Debugging: to facilitate the development of novel algorithms with separated motions along and around the principal axes of the Kinect.
3) Handheld SLAM: to provide a sequence for measuring the long-term drift of (otherwise loop closing) SLAM systems.
4) Robot SLAM: to demonstrate the applicability of SLAM systems to wheeled robots.

The public dataset provided online [32] under the licence Creative Commons Attribution license (CC-BY 3.0) is already calibrated and time synchronised for the Kinect and motion capture data. This dataset can be used as input for any RGB-D SLAM algorithm to obtain estimated trajectory of the Kinect. The estimated trajectory and ground truth are used for evaluation of error and determination of algorithm.

Sturm's benchmark proposes two evaluation methods: Relative pose error and Absolute trajectory error. Trajectory estimate can be represented as follows

$$\hat{X} = \{\hat{x}_1 ... \hat{x}_n\} \tag{1}$$

while the ground truth for corresponding trajectory can simply be represented as X.

$$X = \{x_1 ... x_n\} \tag{2}$$

It is assumed that the sequence of poses for ground truth and estimated trajectory are time synchronised, equally sampled and have equal length (n).

1) Relative pose error (RPE): The RPE is based on relative differences between the poses recorded by ground truth and the estimate for given time steps [33].

RPE is defined as follows:

$$e_{RPE}(\hat{X}, X) = \frac{1}{N} \sum_{i,j \in \tau} trans(\hat{\delta}_{ij} \ominus \delta_{ij})^2 + rot(\hat{\delta}_{ij} \ominus \delta_{ij})^2 \tag{3}$$

where N is the number of relative transformations evaluated and the ij belongs to set of time steps τ.

$$\hat{\delta}_{ij} \ominus \delta_{ij} = ({x_i}^{-1} x_j)^{-1} (\hat{x}_i^{-1} \hat{x}_j)^{-1} \tag{4}$$

The functions $trans(.)$ and $rot(.)$ compute the rotational and translational components of the pose error. Rotational and translational errors can also be computed and assessed separately.

2) Absolute trajectory error (ATE): ATE offers an alternate error measure for evaluation that measures the difference in terms of the absolute offset between corresponding poses. It is the mean of the squared Euclidean distances between the corresponding poses and is defined as shown in 5.

$$e_{ATE}(\hat{X}, X) = \frac{1}{n} \sum_{i=1}^{n} \parallel trans(\hat{x}_i) - trans(x_i) \parallel^2 \tag{5}$$

The correspondences of poses are established beforehand using the timestamps. ATE and RPE are strongly correlated and RPE is slightly larger than APE. This is because RPE takes into account both translational and rotational error while ATE only takes translational pose error. ATE offers intuitive visualization of the error as it offers results based on translational pose error alone.

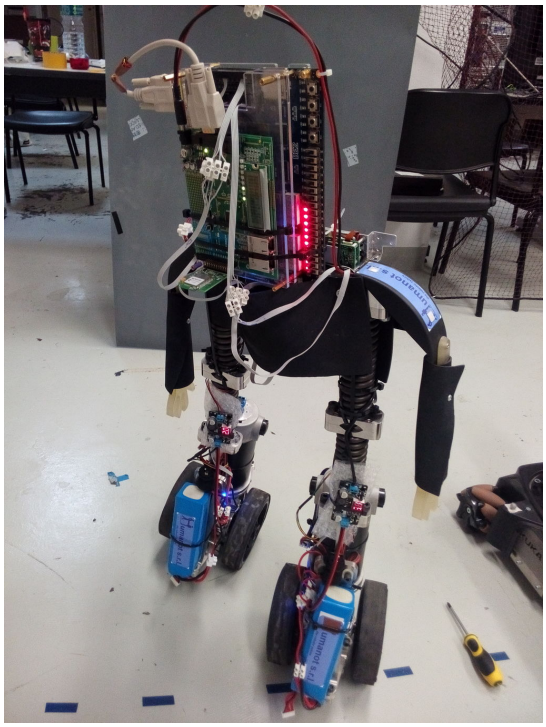
B. Offline evaluation of individual algorithms using Rollo and its Kinect sensor

This requires generation of the Kinect data and recording of ground truth from motion capture by running Rollo. This dataset is more specific to the given system and more appropriate for evaluation of SLAM algorithm in the system. Technical specification of data acquisition include:

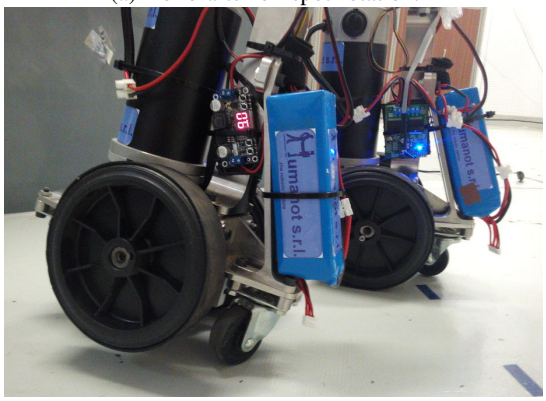
- resolution,
- frame rate,
- update and communication rates with the motion capture and Kinect.

Before the collection of data, intrinsic and extrinsic parameters for Kinect cameras and the motion capture system are calibrated. The calibration of motion capture system is performed using the wand with reflective markers and the software provided with the system. Since no movement of the robot is required, the Kinect was directly connected to the GPU host, allowing for SXGA resolution raw image streaming.

1) Intrinsic calibration of Kinect: The calibration of Kinect cameras is required to determine distortions in the camera input and then rectify the images accordingly. Moreover, calibration also helps to determine the relation between the camera's image pixels and the real world units, for example in metres. This aspect is particularly important for robot environment interaction, which might not be of highest priority for SLAM itself, but also plays a role in it's localization aspect. Kinect delivers 3 image outputs: RGB, IR and depth. The depth image is constructed by triangulation from the IR image and the projector. Before depth calibration of Kinect can be performed, RGB and IR cameras need to be calibrated.



(a) Rollo after on spot rotation.



(b) The left tyre is visibly in air during turn.

Fig. 7: RGB camera calibration.

First, the RGB camera is calibrated. A classical black-white chessboard pattern is chosen with 6 by 9 boxes, each a square of $25mm$ length, referred to as pattern A, for the calibration process. The resolution of camera images is selected as 1280×1024 pixels. Numerous raw RGB images, at least 10, are taken with various positions and orientation of the chessboard pattern in order to generate diverse perspectives. Intrinsic camera parameters of RGB camera are estimated using a developed script written in Python in accordance with the OpenCV camera calibration process . Focal lengths fx, fy, the optical center with coordinates cx, cy and distortion parameters obtained are as shown in table V and VI.Radial distortion parameter k_3 can be skipped in the calibration script and has been skipped during the calibration process. It did get computed and the value seen in the table is very large but its importance is neglibile in the overall camera distortion model.

TABLE V: Intrinsic parameters of RGB and IR camera: focal lengths fx, fy, the optical center cx, cy.

Camera	f_x	f_y	c_x	c_y
RGB	1220.4	1225.5	653.2	481.5
IR	783.3	733.5	348.2	208.1

TABLE VI: Distortion coefficients of RGB and IR camera.

Camera	k_1	k_2	p_1	p_2	k_3
RGB	0.514	-4.48	-0.014	-0.0012	18.9
IR	-5.4	194	0.008	-0.002	-2467

For the distortion, radial and tangential factors are taken into account. The distortion coefficients can be represented in row vector for a camera with 5 coefficients. Distortion coefficient of a camera is represented as D.

$$D = [\ k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3\ ] \tag{6}$$

Once the distortion coefficients are determined for a camera, the radiant and tangential distortions can be corrected. For a given pixel position x and y, radial distortion can be corrected as shown in 7 and 8.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{7}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{8}$$

Tangential distortions can be corrected using equations as shown in 9 and 10.

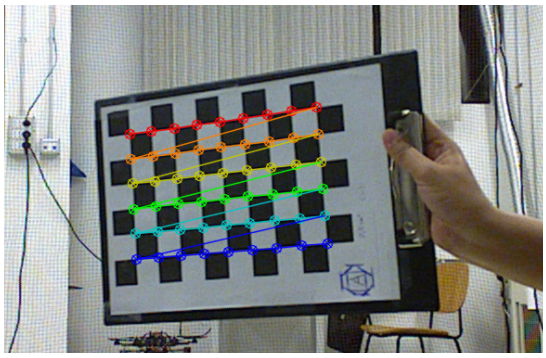
$$x_{corrected} = x[2p_1 xy + p_2(r^2 + 2x^2)] \tag{9}$$

$$y_{corrected} = y[2p_2 xy + p_1(r^2 + 2y^2)] \tag{10}$$

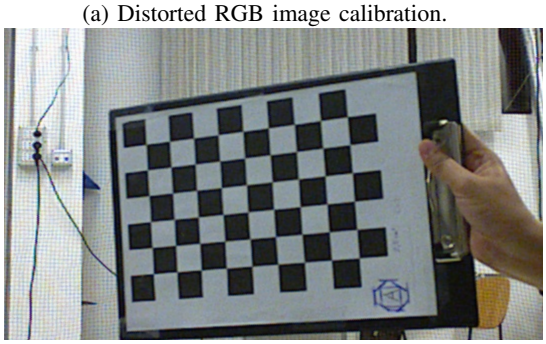
Same procedure needs to be followed for IR camera calibration. However, it was discovered that the images from IR camera were not as easily extractable as with the RGB camera. Raw IR camera images were very dark and an object or motion in front of it was barely distinguishable from the background. In order to extract chessboard pattern from IR input, images were preprocessed by applying various filters before calibration.

Following procedure was devised to process IR images and extract valuable information for calibration. The script for this image processing was also written in Python and called in various Bash scripts for more control and comfortable streamlined process.

1) First the raw IR image is normalised and saved.
2) The script loads the IR image as gray scale and scales it up by factor of 2. Original VGA resolution is 640 x 480 (640 x 488 header). SXGA 1280 x 1024.
3) Canny edge detector is used to detect edges in the image



(a) Distorted RGB image calibration.



(b) Undistorted RGB image.

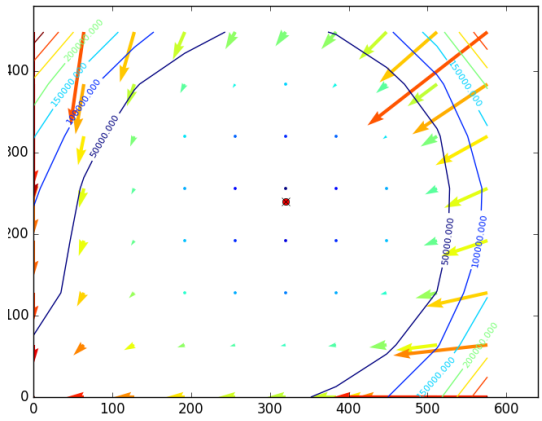
Fig. 8: RGB camera calibration.

4) Median filter with a range of 3 is applied on the image to reduce noise.
5) Next, image is 'closed' by process of dilation of erosion with a kernel of square 3 x 3.
6) The image result of canny edge detector and 'closed' image are summed together and then further processed with gaussian blur.
7) The image is binarized using threshold binary and otsu-threshold.
8) Image is resized to original proportion and saved.
9) Process is repeated for all images

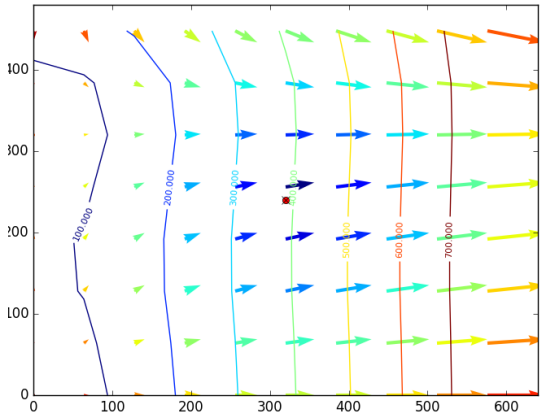
It was observed that the IR projector influences acuity of the image. If the checkerboard is held very close to the Kinect, camera reflection from projector is very high resulting in over saturation and patterns not being recognizable. If the projector is covered, the images get even darker. Additionally there seem to be an distance range from the camera, where the noise level is low enough and contrast high enough resulting in sharp images of the pattern A. For this reason a larger checkerboard pattern was used, consisting of 5 by 7 boxes with each square box of length $48mm$ on an A3 size paper and referred to as pattern G 11.

It was observed that if the checkerboard is held perpendicular to the camera plane, it is not as well recognized as the results when the board is held at an angle. This is also influenced by IR projector and is the result of glare coming from the IR camera. This phenomena is recognizable in a sequence of images.

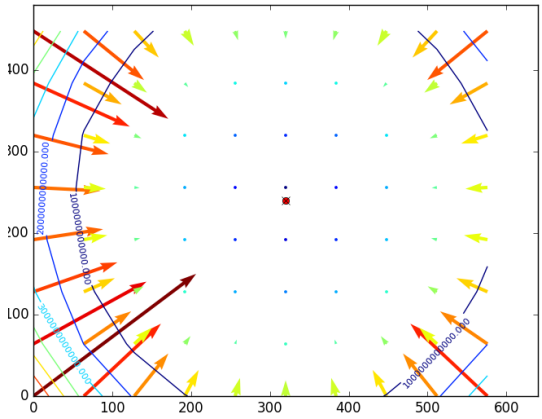
Additionally, it appears that IR sensor needs to be charged with light radiation for relatively long time, which results in



(a) Radial Distortions in RGB images.



(b) Tangential Distortions in RGB images.



(c) Complete Distortions in RGB images.

Fig. 9: RGB camera: radial, tangential and complete distortions.

very noisy and blurred input from movement. Therefore it is advised that in order to capture high contrast and low noise IR images, the calibration pattern should be held in a fixed place for a certain time.

The intrinsic calibration shows results corresponding to gathered experience in this area. The radial distortion are high towards the outer parts of the image and tangential distortions cover the whole image plane, combined representing each cameras unique characteristics that need to be taken into consideration during their operation. ROS allows for reading of YAML calibration files for cameras within the RGBD based SLAM algorithms, therefore it is highly advised to use the results for future work.

Next, extrinsic calibration of both Kinect and the motion capture system is performed.

For this, reflective markers are placed on both the calibration checkerboard and the Kinect. The checkerboard and the Kinect are selected as separate rigid bodies. The pose of checkerboard is determined with respect to the coordinate system of the motion capture system. Using this, average error between point observation and point model is determined and then the estimated orientation error.

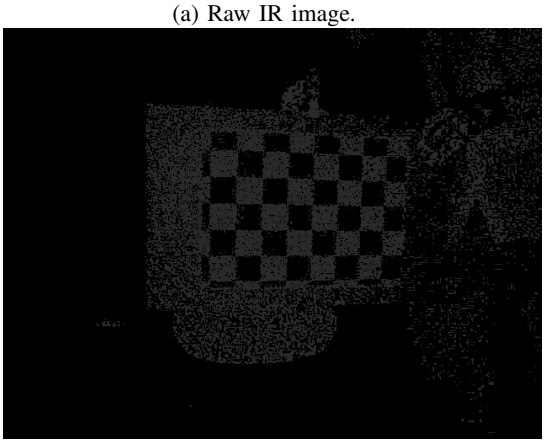
Finally, the transformation between the pose from the motion capture system and the optical frame of the Kinect is determined using the calibration checkerboard.

Once the data is collected, it can be used to compute estimated trajectory from a given SLAM algorithm offline. The ground truth from the motion capture system and the estimated pose of the camera using SLAM needs to be time synchronized before analysis.

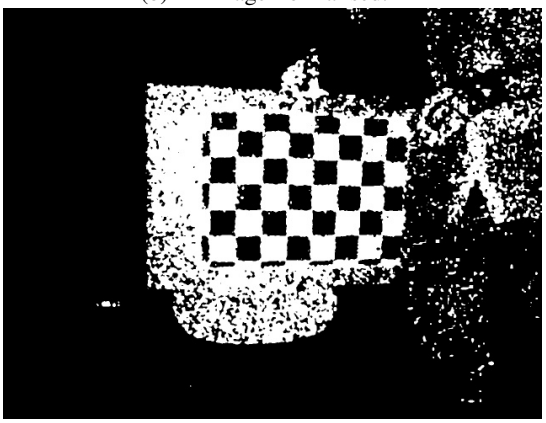
C. Online evaluation of indiviual algorithms

Online evaluation involves determining error and evaluating algorithm while the robot is running and the algorithm is executed. Therefore for different algorithms separate test runs would be required for evaluation making it difficult to sustain identical conditions. This aspect makes offline evaluation a more efficient and more precise approach from the point of view of evaluation.

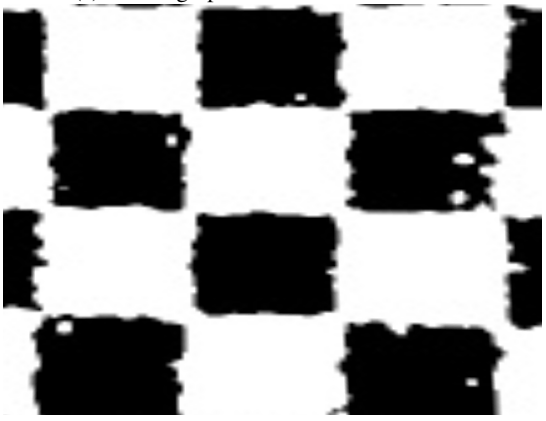
(a) Raw IR image.



(b) IR image normalised.



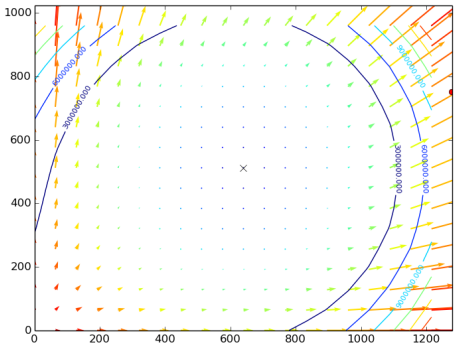
(c) IR image processed for calibration.



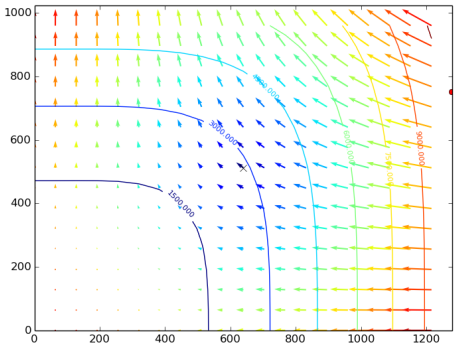
(d) Undistorted IR image.

Fig. 10: IR camera calibration.

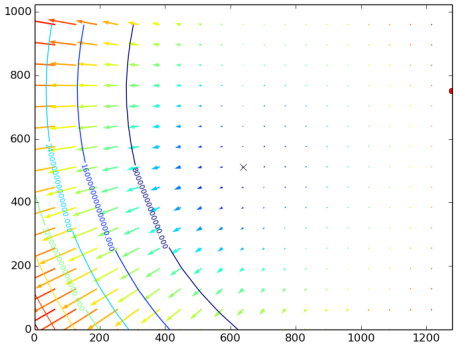
Fig. 11: Checkerboard pattern G: 5 by 7 square tiles of size 48 mm each.



(a) Radial Distortions in IR images.



(b) Tangential Distortions in IR images.



(c) Complete Distortions in IR images.

Fig. 12: IR camera: radial, tangential and complete distortions.

VI. CONCLUSIONS

The document discusses the work involved in implementation of RGBD based SLAM algorithms for a specific setup of Rollo and many challenges faced during that process. Based on gathered experience further work on the problem statement should include proper setup of operating systems for involved devices, from ground up if necessary to minimize error prone software components and allow for robust and performance oriented working conditions. Additionally the calibration process of Kinect should be extended with offline datasets and online calibration techniques, especially in light of the heavy development work undertaken to streamline the whole process, in particular the developed scripts. Care needs to be taken with calibration using depth images, since it involves another resolution. In order to enable full mobility of the robot, configuration of the ROS implementation needs take care of reduced frame rate and network bandwidth. For this the goals should be defined anew, focusing less on energy efficiency of the mobile robot as well as the accuracy of generated maps and more on the balance between them.

REFERENCES

[1] https://openslam.org/slam.html
[2] http://www.humanot.it/products/humanoids
[3] https://github.com/em-er-es/rollo
[4] https://github.com/ros-drivers/freenect_stack
[5] https://developer.nvidia.com/cuda-zone
[6] https://developer.nvidia.com/cuda-gpus
[7] http://www.ros.org
[8] http://wiki.ros.org/Master
[9] https://openkinect.org/wiki/Hardware_info
[10] http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot#Kinect
[11] http://felixendres.github.io/rgbdslam_v2
[12] F. Endres, J. Hess, J. Sturm, D. Cremers, W. Burgard, "3D Mapping with an RGB-D Camera", in IEEE Transactions on Robotics, 2014.
[13] https://github.com/Mavericklsd/ rgbd_mapping_and_relocalisation
[14] Li, S., & Calway, A. "RGBD Relocalisation Using Pairwise Geometry and Concise Key Point Sets" in IEEE Intl. Conf. on Robotics and Automation (ICRA), 2015.
[15] http://introlab.github.io/rtabmap
[16] Grisetti, G. and Kuemmerle, R. and Stachniss, C. and Burgard, W., "A Tutorial on Graph-Based SLAM," in *Intelligent Transportation Systems Magazine, IEEE*, Vol. 2, pp. 31-43, 2010
[17] Ubuntu installation manual
[18] rgbd-marl environment setup file
[19] http://docs.ros.org/api/sensor_msgs/html/msg/Image.html
[20] https://www.gnu.org/software/gdb/
[21] http://joeyh.name/code/alien/
[22] http://wiki.ros.org/freenect_camera
[23] http://tinkersphere.com/raspberry-pi-accessories /321-raspberry-pi-2-wifi-dongle.html
[24] http://www.tp-link.com/en/products/details/TL-WN823N.html
[25] https://www.raspbian.org/
[26] https://wiki.archlinux.org/index.php/Arch_Linux#Principles
[27] Wasenmuller, Oliver and Meyer, Marcel and Stricker, Didier, "CoRBS: Comprehensive RGB-D Benchmark for SLAM using Kinect v2" in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016
[28] Stephan Meister and Izadi, S. and Kohli, P. and Hämmerle, M. and Rother, C. and Daniel Kondermann, "When Can We Use KinectFusion for Ground Truth Acquisition?" in *Workshop on Color-Depth Camera Fusion in Robotics, IEEE International Conference on Intelligent Robots and Systems*, 2012
[29] Zhou, Qian-Yi and Koltun, Vladlen, "Dense Scene Reconstruction with Points of Interest", in ACM Trans. Graph., Vol. 32, 2013
[30] Ankur H and Thomas Whelan and John Mcdonald and Andrew J. Davison "A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM"
[31] J. Sturm and N. Engelhard and F. Endres and W. Burgard and D. remers, "A Benchmark for the Evaluation of RGB-D SLAM Systems", in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012
[32] http://vision.in.tum.de/data/datasets/rgbd-dataset
[33] Kmmerle, R., Steder, B., Dornhege, C., "On Measuring the Accuracy of SLAM Algorithms" in Autonomous Robots (2009)