Range Scanning System for an Omnidirectional Mobile Robot: Infrastructure setup

Projektpraktikum Computational Neuro Engineering: Report

Armin Pribbernow

Fachgebiet für
NEUROWISSENSCHAFTLICHE SYSTEMTHEORIE
Technische Universität München

Prof. Dr. Jörg Conradt

Betreuer: M.Sc. Cristian Axenie
Beginn: 14.10.2014
Abgabe: 14.01.2015

In your final hardback copy, replace this page with the signed exercise sheet.

Abstract

This report describes the infrastructure setup process for a laser rangefinder enhancing the hard- and software for an already existing omnidirectional mobile robot. An USB interface for the laser rangefinder is developed, using a provided library with basic functionality from the manufacturer. To synchronize data acquisition over USB from both laser rangefinder and robot an embedded board is needed and the existing Wi-Fi interface of the robot has to be changed to USB. This embedded board also acts as a server to stream the synchronized sensor data to an offboard PC (client) over Wi-Fi. A simple GUI on the offboard PC allows manual control of the robot, extended with the laser rangefinder, visualizes the received sensor data in different plots.

Contents

Chapter 1

Introduction

The motivation for this task arises from the problem of autonomous mobile robot exploration in an unknown environment. Due to no absolute position reference the position has to be estimated from embedded sensors, like inertial, odometry and proprioceptive sensors, or cameras. What remains, despite predefined landmarks, is the object recognition problem. Because our system is a low power embedded system we cannot afford to process frames from a camera due to high computational load (especially memory usage). An alternative for cameras are laser rangefinders as they offer a good scanning coverage as well as high precision and data rates. The final goal of a superior and more complex project is to implement sensor fusion using all available sensors, including laser range data, onboard the mobile robot, therefore a basic infrastructure setup (hard- and software) to interface with the `LRF` is needed.

1.1 Onboard setup

The embedded, or onboard, side of the project is the mobile robot extended with the laser rangefinder. To allow synchronized data acquisition and streaming over Wi-Fi an appropriate embedded board has to be chosen. Further tasks include getting familiar with data acquisition and preprocessing of the laser rangefinder. In the software library provided by the manufacturer the communication protocol with basic functionality is already implemented. Simultaneous streaming over Wi-Fi implies the use of multiple threads: a server functionality has to be implemented to communicate with the offboard PC (send and receive data and commands) while the realtime data acquisition from the sensors remains unaffected.

1.2 Offboard setup

The offboard setup is basically a graphical user interface and acts as the client side of the overall application. It shall visualize all received data and log it in raw format to allow later use in other algorithms, for example sensor fusion or SLAM, and enable

manual control of the mobile robot (data formats, motor control signals, i. a.) via a text-based terminal/commander and buttons for moving the robot.

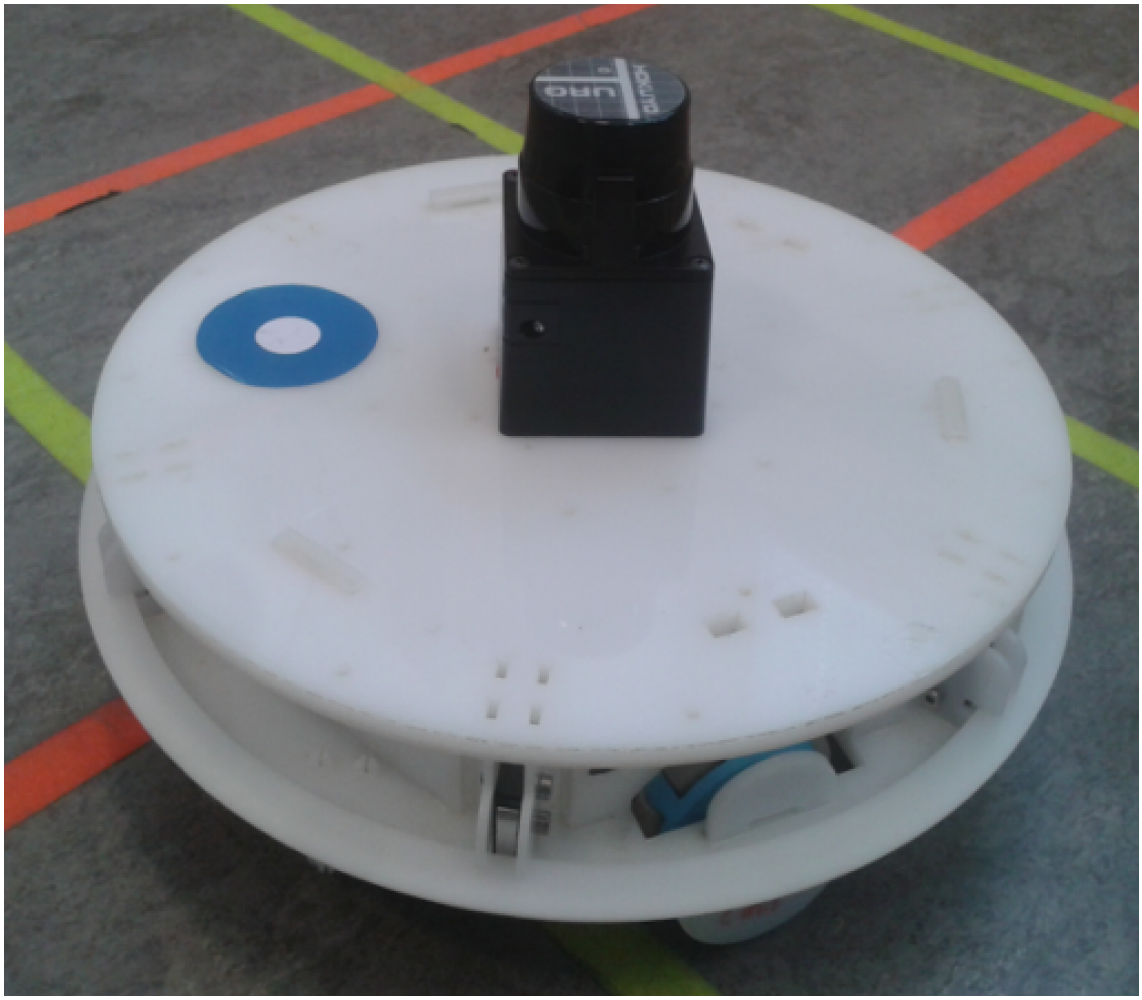

Figure 1.1: The `OmniRob` together with the `LRF`

Chapter 2

Infrastructure Setup

2.1 Hardware

This section briefly describes the used hardware and why it has been chosen. For now the software setup has priority but a future task is to think also about how the embedded board can be hold in place inside the mobile robot and how the laser rangefinder can be attached to the top in a convenient way.

2.1.1 The Omnidirectional Mobile Robot

The `Omnidirectional Mobile Robot`, furthermore called `OmniRob` for convenience, is the basis for the whole project. It streams the following sensor data: 6 bump sensors, computed wheel speed for all 3 wheels, 3-axis gyrometer, 3-axis accelerometer, euler angles (roll/pitch/yaw) of the estimated pose, 3-axis compass (plus estimated heading), pwm signals for all 3 motors, computed wheel angles for all 3 wheels, 6 potentiometer (raw data) and reference velocity (internally used wheel velocities). In fig. 2.1 one can see where some of the sensors are located on the actual robot.
To extend the `OmniRob` the embedded board will be put inside the case, while the laser rangefinder will be put on top of it.
One can find the documentation for the `OmniRob` and how to access it unter the following address: `https://wiki.lsr.ei.tum.de/nst/documentation/omnirob`. The sample program to interface with the robot over Wi-Fi is quite easy to use as it runs in the terminal and understands basically to types of messages: commands beginning with a `!` (for example to set the pwm cycle length,the motor velocity or to start and stop the streaming fo sensor data), and single requests beginning with a `?` (to read the latest data from a specific sensor). Additionally there is a reset, a help and a beep command.

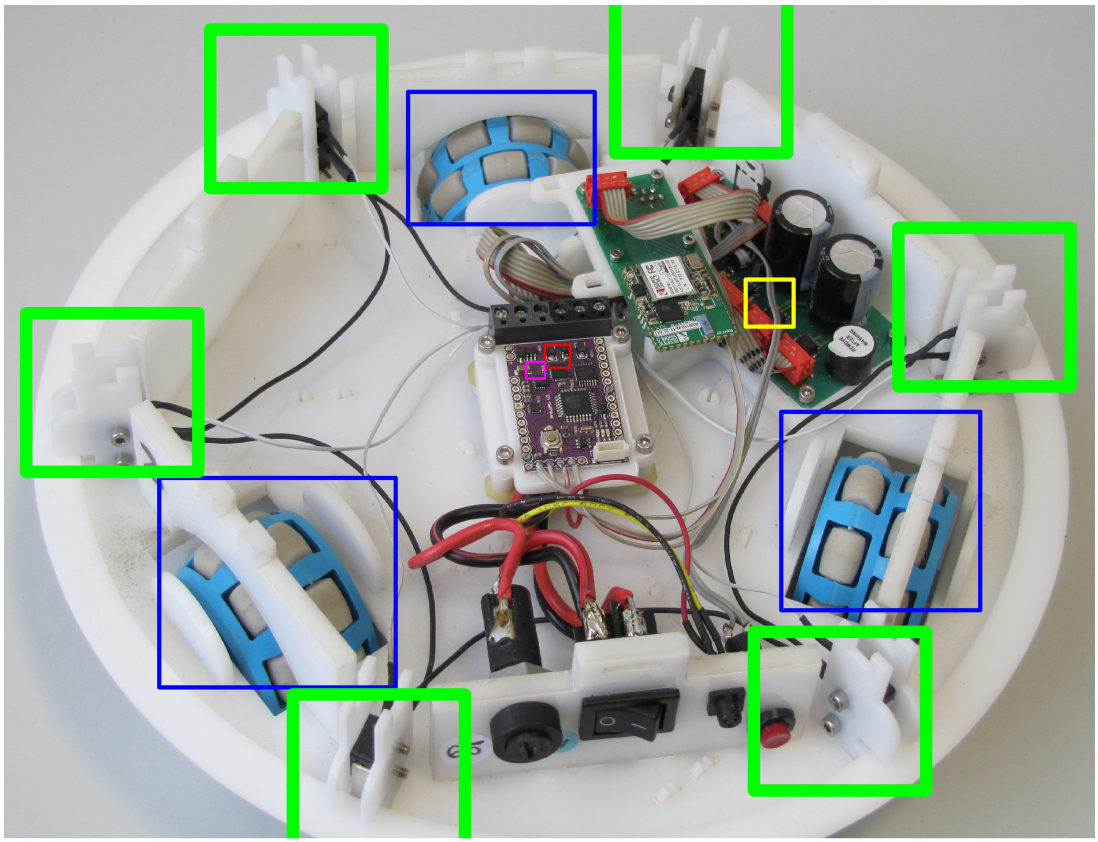

Figure 2.1: `OmniRob` with tagged bump sensors (green), wheel encoders (blue), gyro-/accelerometer (red), compass (purple), pwm controller (yellow)

2.1.2 The HOKUYO URG-04LX-UG01 laser rangefinder

Hokuyo is a japanese manufacturer of automation and sensor technology and known for being the first one to develop laser scanners for the mass market.

The URG-04LX-UG01 laser rangefinder has been chosen because of its fast communication over USB. A library with basic interfacing functionality is provided as well as a simple tool for visualizing the data, which makes it easier to get started. It has a measurement area of 240° with a resolution of 1mm, an angular resolution of 0.36° and a scanning frequency of 10Hz. The data acquisition is customizable in many ways: scanning range (angle), accuracy (3/2-byte) and skipped sensor data.

A picture of the used laser rangefinder is shown in Fig. 2.2.

Figure 2.2: The URG-04LX-UG01 from Hokuyo

2.1.3 The TRONSMART T428 mini PC

Originally the `Tronsmart T428` is intended to be used as a HDMI TV dongle. It is shipped with Android running on it, but the possibility to flash it with custom-roms, including several linux distributions, makes it interesting anyways. Together with the relatively cheap price of currently ca. 70 $ the technical specifications are very good compared to other manufacturers of TV sticks and mini PCs: the core is a `Rockchip RK3188` (quad core Cortex-A9 SoC), additionally there are 2GB RAM, 8GB flash, micro SD slot, Bluetooth 4.0, Wi-Fi (supporting 5GHz frequency), HDMI video output, USB 2.0 host and micro USB for power.

The available Linux distributions are `Ubuntu`, `Xubuntu` and `Lubuntu`, each of them in a 720p and a 1080p version. A USB hub can be used to connect mouse and keyboard meaning that one can work with this TV stick just as with a desktop PC.

Figure 2.3: The `Tronsmart T428 mini PC`

2.2 Software

This section describes the development process for the embedded board application and the offboard PC GUI. Both applications were developed on the same system (`Ubuntu 14.04`) to allow easy testing of USB data acquisition and the streaming functionality (using the localhost domain). Later the embedded software simply has to be compiled on the embedded board (`Lubuntu 13.04`).
The `urg_library-1.1.8` in C programming language is the basis for the laser rangefinder part of the embedded application. For accessing the OmniRob there already exists a sample program which structurally remains unchanged but is ported from Wi-Fi to USB communication. This is done to bundle up the OmniRob and the laser rangefinder software parts on one device (the embedded board) and be able to stream the data synchronized in one application.
A sketch of the whole application and how the embedded and the offboard side co-operate is shown in Fig. 2.4.

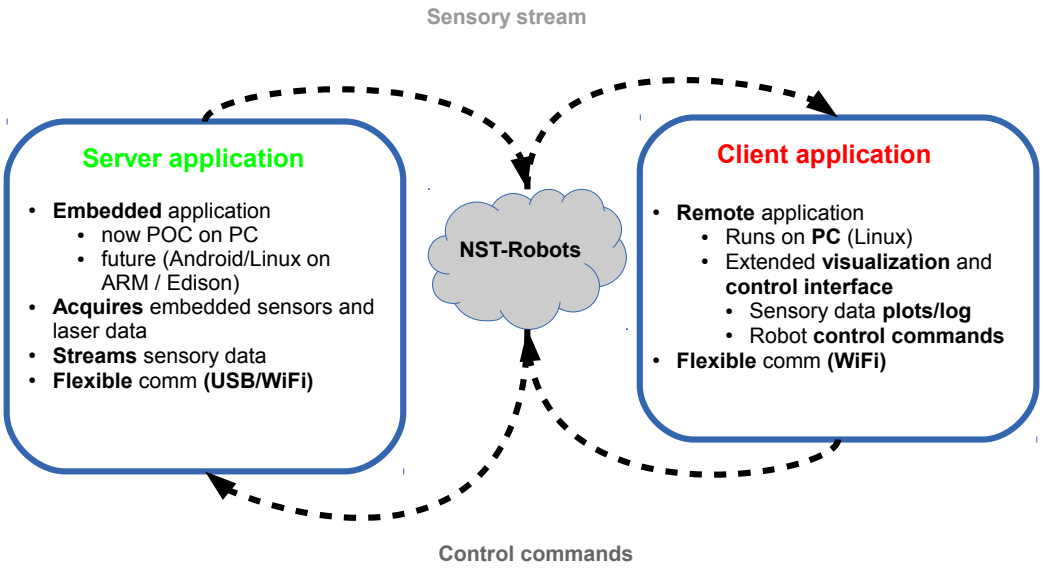

Figure 2.4: Co-operation between embedded and offboard side of the application

2.2.1 Embedded board application

The whole embedded board application is programmed in C.

USB

The low level USB communication for the laser rangefinder is implemented in the urg_library-1.1.8. This library provides functions for opening and closing a USB connection to the LRF, setting measurement parameters and getting the laser range data . Regarding the measurement parameters one can choose between 3-byte and 2-byte sensor data accuracy (affects memory usage and needed bandwidth), set the scanning range (start angle, stop angle) or decide to only acquire every second/third/fourth sensor value to save memory and bandwidth.
The low level USB communication with the OmniRob is realized with termios API, a newer version of the Unix API for terminal I/O. This API uses Unix system calls to open and close a serial device, configure communication parameters and to read from and write to the serial interface.

Server Sockets

For the implementation of network communication the posix socket api is most appropriate. Two server sockets (TCP/IP) are needed, one for the LRF, one for the OmniRob: because the overall application should work even if one of the two devices is not functional (for whatever reason) it is necessary to be able to stream data separately. Creating a server socket is done in four steps: creating the socket, bind it to a specific port, listen for incoming connections from a client and accept a connection. Accepting and handling multiple connections at the same time is possible but implies multithreading.

Threads

As USB data acquisition and data streaming over Wi-Fi should happen simultaneously the embedded board application is multithreaded as well. Posix threads, in short pthreads, are the tool of choice when a pthread is created, a pointer to a function is passed. This function usually implements some kind of infinite for/while loop and a stop condition to be able to exit the thread.
In the present case the main-function creates two threads, one for he LRF, one for the OmniRob, and after that just waits until both of them have done their work (exit) and "join" the main-thread again.
The function running in the LRF thread now tries to establish the USB connection to the actual LRF and upon success starts to acquire the sensor data (infinite loop)

and creates a server thread. In this server thread the server socket is created and afterwards waiting for incoming connections. If a client connects to the server socket the infinite loop of the server thread is reached where the socket listens on incoming commands from the client. Finally, to ensure thread-safety, `mutexes` are used to secure the code areas where data is passed from one thread to another (e.g. when passing a command received over Wi-Fi to the `LRF` over USB).
The same happens for the `OmniRob` thread. In both cases the stop condition for the infinifite loops would be a lost connection. Overall there are four threads working if one does not count the `main`-thread, which does basically nothing else than waiting.

2.2.2 Offboard PC GUI

The offboard application is programmed in `C++`, using `Qt`, and basically two-part. The first part is the GUI for visualizing the received sensor data and manual control of the `OmniRob`, the second is the client which tries to connect to the server on the embedded side. Just like for the embedded application more than one thread is needed, mainly to ensure the responsiveness of the GUI to user input.

The signal/slot mechanism in Qt

`Qt` was chosen because of its so-called signal-slot-mechanism. It allows easy communication between objects and classes (in this case client and gui) without the necessity to care about thread-safety. `Signals` and `slots` are an alternative to the callback technique.
`signals` are emitted when particular events occur, or they are emitted manually by the programmer upon programmer-defined events. `Slots` are the opposite part: they are called whenever the signal they are connected to is emitted. Multiple signals can be connected to one slot and multiple slots can be connected to one signal. Almost all `Qt`-classes are `signal/slot` compatible. For example: the `QAbstractSocket`-class emits signals whenever the socket connects or disconnects, a host is found, an error has occurred etc. . This makes it very easy to handle such events as the programmer just has to define a `slot` in which the event shall be handled and to connect this `slot` to the corresponding `signal` (`Qt` offers the `connect(...)`-function for this). Moreover thread-safety is handled internally, thus no `mutexes` or `semaphores` defined by the programmer are needed.

QCustomPlot

`QCustomPlot` is an extension for `Qt` in the form of a `Qt widget`. and mainly intended for data visualization. In this application it is used to plot the streamed and received sensor data in multiple tabs with high performance: one tab for each type of sensor data (lrf/gyro/acc/...). Each tab has its own `QCustomPlot` and a `slot` which is connected to one `signal` in the client-class. The `signal` is emitted whenever the

client receives the corresponding sensor data from the embedded board. Because client and gui run in different threads the gui remains responsive while always the latest data from all sensors is plotted (despite large number of sensors and thus relatively high computational load).
In Fig. 2.5 and Fig. 2.6 one can see plotted gyrometer and `LRF` data.

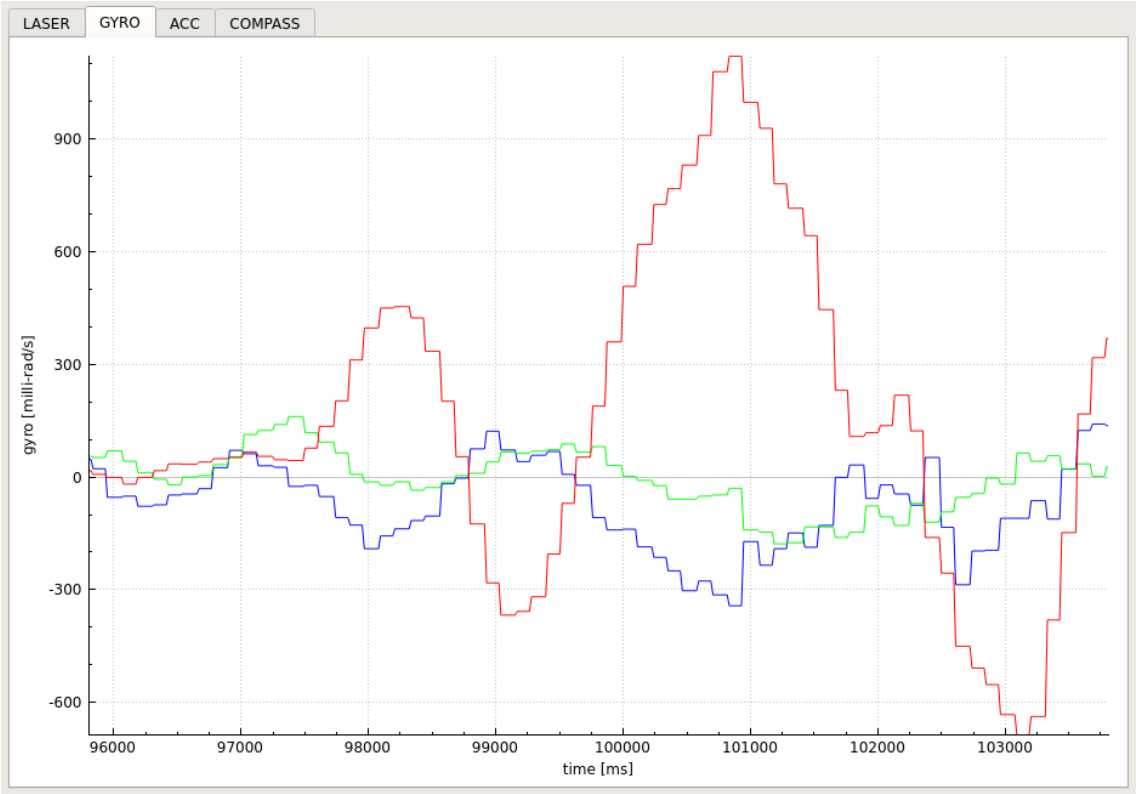

Figure 2.5: Gyrometer data plotted in the GUI

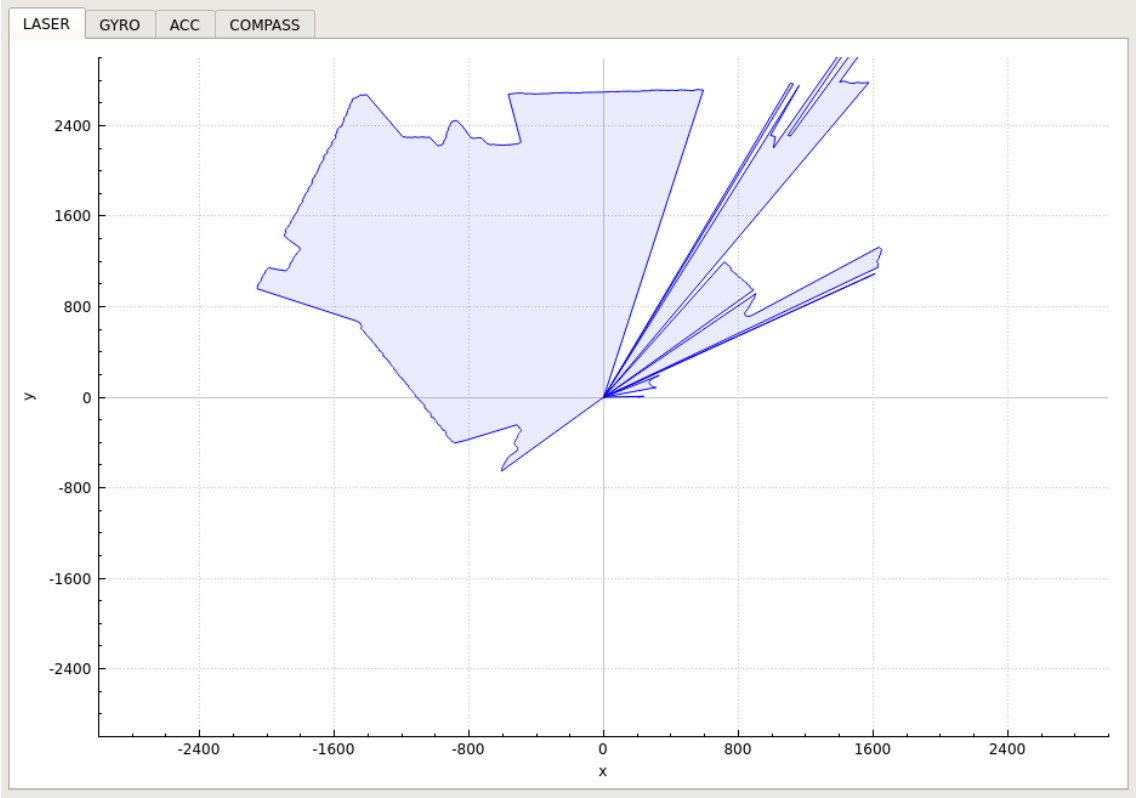

Figure 2.6: LRF data plotted in the GUI

2.2.3 The overall application

The scheme for the overall application is shown in Fig. 2.7. Each rectangle represents a thread, the arrows indicate communication over USB, Wi-Fi or, in case of green, the `signal/slot` mechanism. The status box (Fig. 2.8) inside the GUI allows to connect and disconnect to and from both devices. The manual control box (Fig. 2.9) allows simple steering of `OmniRob` while all buttons are also shortcutted to the matching `Numpad`-keys on the keyboard. The terminal box will later be used to send commands just like in the original sample program for the `OmniRob`. Finally, all sensor data received by the GUI is logged to streamdump-files in raw binary format.

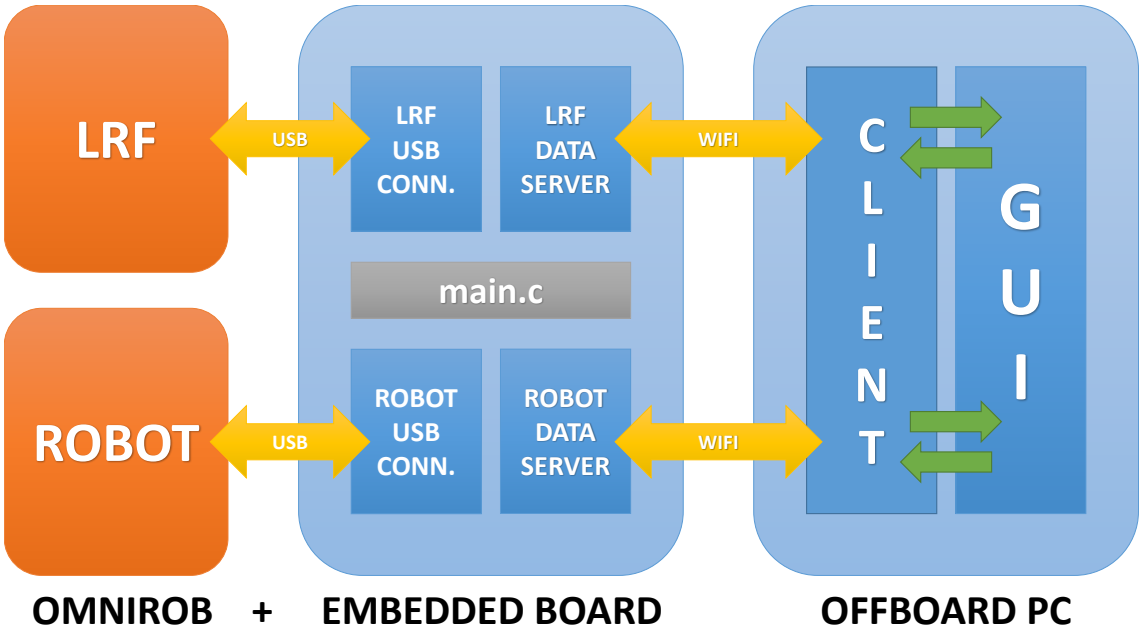

Figure 2.7: Scheme for the overall application

Figure 2.8: The status box in the GUI

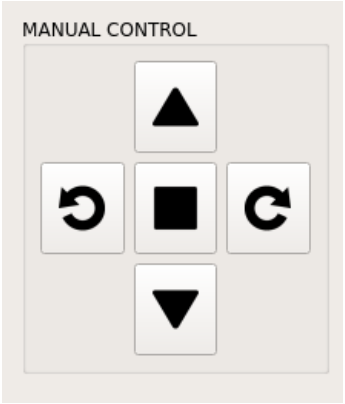

Figure 2.9: The manual control box in the GUI

List of Figures

License