VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

OVLADAČ LASEROVÉHO SKENERU SICK LMS111 PRO RASPBERRY PI 2

DEVICE DRIVER FOR THE SICK LMS111 LASER SCANNER FOR RASPBERRY PI 2

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Vlastimil Lahoda
AUTHOR

VEDOUCÍ PRÁCE Ing. František Burian, Ph.D.
SUPERVISOR

BRNO 2018

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
TECHNICKÉ VYSOKÉ UČENÍ V BRNĚ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Vlastimil Lahoda ***ID:*** 158184
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Ovladač laserového skeneru SICK LMS111 pro Raspberry Pi 2

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s laserovým skenerem SICK LMS111 a platformou Raspberry Pi.
2. Analyzujte všechny funkce skeneru a v jazyce C++ realizujte třídu pro plnohodnotnou obsluhu tohoto zařízení. Mějte na paměti, že rozhraní třídy by do budoucna mělo být použitelné i pro jiné laserové skenery.
3. Doplňte další funkce pro pohodlné použití dané třídy (logování dat, hlášení chyb, přepočty souřadných systémů…). Optimalizujte třídu pro co nejlepší efektivitu.
4. Vypracujte podrobnou dokumentaci k celému projektu, která umožní snadné použití vašeho kódu.
5. Správnost fungování knihovny prezentujte pomocí jednoduché demo aplikace.

DOPORUČENÁ LITERATURA:

Rozumíme C++. Praha: Computer Press, 2003. Programování. ISBN 80-722-6656-X.

PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Přeložil Boris SOKOL. Brno: Computer Press, 2013. Bestseller (Computer Press). ISBN 9788025138281.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. František Burian, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:
Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Práce se věnuje návrhu a realizaci ovladače laserového skeneru pro minipočítač Raspberry Pi 2. V první části seznamuje čtenáře se základními informacemi o laserovém skeneru SICK LMS111-10100 a Raspberry Pi 2. Dále se věnuje rešerši vývojového prostředí a výběru programovacího jazyka. Následuje návrh a realizace třídy pro plnohodnotné ovládání skeneru z Raspberry PI 2.

Klíčová slova

SICK LMS111, Raspberry PI, Raspbian, Qt knihovna, C++.

Abstract

This thesis is focused on design and realization of the laser scanner driver SICK LMS111-1010 for minicomputer Raspberry Pi 2. In the first part the reader is familiar with the basic information about the laser scanner SICK LMS111-10100 and Raspberry Pi 2. It also focuses on the research Integrated Development Environment and the selection of the programming language. Following is the design and implementation of a class for full control of SICK LMS111-10100 skener from Raspberry Pi.

Keywords

SICK LMS111, Raspberry PI, Raspbian, Qt framework, C++.

Bibliografická citace:

LAHODA, V. Ovladač laserového skeneru SICK LMS111 pro Raspberry Pi 2. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 40 s. Vedoucí bakalářské práce Ing. František Burian, Ph.D..

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Ovladač laserového skeneru SICK LMS111 pro Raspberry Pi 2 jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 21. května 2018 ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Františku Burianovi, Ph.D. za odbornou pomoc, trpělivost, podnětné návrhy a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: 21. května 2018 ……

podpis autora

Obsah

1. Úvod ... 12
2. Teoretický rozbor ... 13
2.1 Laserový skener SICK LMS111 ... 13
2.1.1 Měření vzdálenosti a úhlu ... 14
2.1.2 Vliv povrchu objektu na měření ... 14
2.1.3 Komunikace skeneru ... 15
2.2 Raspberry Pi 2 ... 16
2.3 Operační systém Raspberry Pi ... 17
3. Návrh řešení ... 18
3.1 Zapojení ... 18
3.2 Programovací jazyk a vývojové prostředí ... 19
3.3 Vstupy a výstupy ovladače ... 20
3.3.1 Funkce MeasData ... 20
3.3.2 Funkce SaveData ... 21
3.4 Vizualizace naměřených hodnot ... 21
3.5 Realizace knihovny ovladače ... 23
3.5.1 Abstraktní třída Skener ... 23
3.5.2 Třída LMS111 ... 24
3.5.2.1 Metoda Connect ... 24
3.5.2.2 Metoda Disconnect ... 25
3.5.2.3 Metoda WriteTelegram ... 25
3.5.2.4 Metoda SocketRead ... 25
3.5.2.5 Třída ErrorCode a metoda SopasErrorCodes ... 26
3.5.3 Třída Token ... 27
3.6 Příklad použití ovladače ... 29
3.6.1 Vícevláknový příklad použití ovladače ... 29
3.6.1.1 Třída Threads ... 29
3.6.1.2 Funkce main ... 33
3.6.1.3 Ovládání aplikace ... 34

3.6.2 Jednovláknový příklad použití ovladače 34
3.6.3 Prezentace reálných naměřených dat 35
4. Závěr 37

Seznam symbolů a zkratek

Zkratky:

LiDAR	…	Light Detection and Ranging
IDE	…	Integrated Development Environment
TCP/IP	…	Transmission Control Protocol/Internet Protocol
ASCII	…	American Standard Code for Information Interchange
NAS	…	Network Attached Storage
XML	…	Extensible Markup Language
CAN	…	Controller Area Networkt
IOT	…	Internet of Things

Seznam obrázků

Obr. 2-1 Princip měření LMS111 [1] 13
Obr. 2-2 Princip měření doby letu světelného pulsu [1]. 14
Obr. 2-3 Odraz paprsku na povrchu objektu. Většina povrchů odráží paprsek difuzně ve všech směrech [1]. 15
Obr. 2-4 Raspberry Pi 2 [7] 16
Obr. 3-1 Zapojení Skeneru [6], Raspberry Pi [7], routeru[8] a PC[9] 18
Obr. 3-2 Ukázka grafu z Matlabu 22
Obr. 3-3 Vývojový diagram vícevláknového příkladu 30
Obr. 3-4 Znázornění měření skeneru LMS111 [3] 31
Obr. 3-5 Ukázka prostředí aplikace po uložení jednoho měření 34
Obr. 3-6 Fotografie skenovaného prostoru 36
Obr. 3-7 Graf naměřených hodnot 36

Seznam tabulek

Tabulka 2.1 Příklad ASCII telegramu [3] 15
Tabulka 2.2 Příklad binárního telegramu [3] 16

1. ÚVOD

Jako téma Bakalářské práce jsem si vybral Ovladač laserového skeneru SICK LMS111 pro Raspberry Pi 2, jelikož na Ústavu automatizace a měřicí techniky tento skener mají k dispozici a je několik aplikací, kde by se dal tento skener využít. Jedna z možných aplikací je připojit laserový skener k podvozku robota a monitorovat prostor kolem něj. Na ústavu již tyto podvozky mají a jsou ovládány minipočítačem Raspberry PI, proto jsem se rozhodl ovladač pro tento minipočítač naprogramovat. Je to pro mě ideální příležitost osvojit si problematiku programování a snímačů v praxi, což je neocenitelná zkušenost.

Cílem této Bakalářské práce je seznámit se s minipočítačem Raspberry Pi, s funkcemi a vlastnostmi skeneru SICK LMS111 a realizovat třídu v programovacím jazyce C++ pro plnohodnotné ovládání tohoto skeneru. Vytvořit dokumentaci k projektu a funkci ovladače prezentovat jednoduchou demo aplikací.

V první části Bakalářské práce se budu zabývat teoretickým rozborem problematiky laserového skeneru Si LMS111, minipočítače Raspberry Pi a jejich propojení.

V další části práce se budu věnovat návrhu a realizaci třídy ovládající skener. Popis nejdůležitějších metod třídy a ukázkové demo aplikace. Protože k realizaci ovladače jsem požil Qt knihovnu, bylo možné vytvořit multiplatformní ovladač, který běží jak pod Windows, tak pod Linuxem.

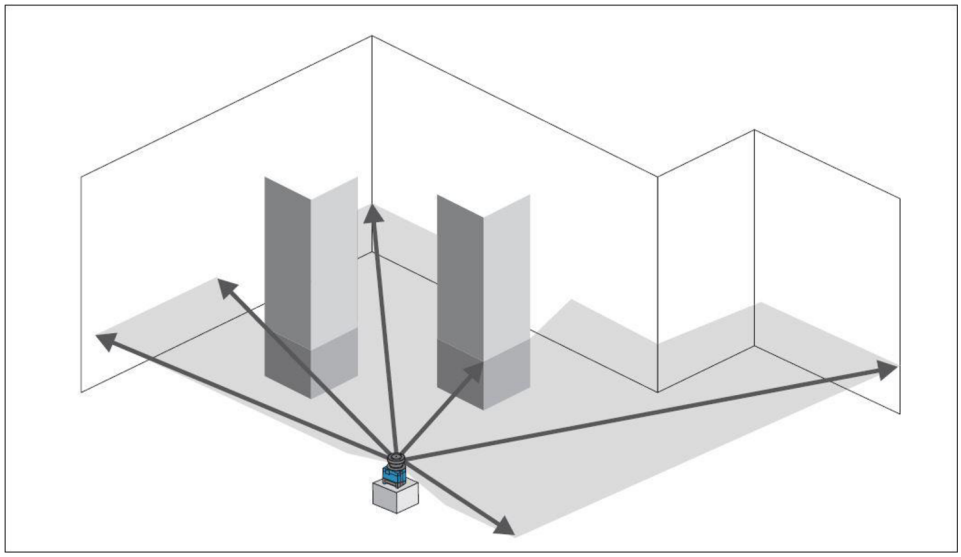
2. TEORETICKÝ ROZBOR

Tato kapitola bude věnována teoretickému rozboru laserového skeneru SICK LMS111 a minipočítači Raspberry Pi 2.

2.1 Laserový skener SICK LMS111

K dispozici jsem měl skener od firmy SICK, konktrétně typ LMS 111-10100. Jedná se o elektro-optický 2D – LiDAR senzor, který skenuje své okolí v rovině s maximálním úhlem 270° až do vzdálenosti přesahující 20 metrů za pomoci laserových paprsků. Výstupem skeneru jsou dvourozměrné polární souřadnice. Souřadnice bodu jsou určeny ze vzdálenosti a směru laserového paprsku dopadajícího na skenovaný objekt. [1]

Obr. 2-1 Skener SICK LMS111 [6]



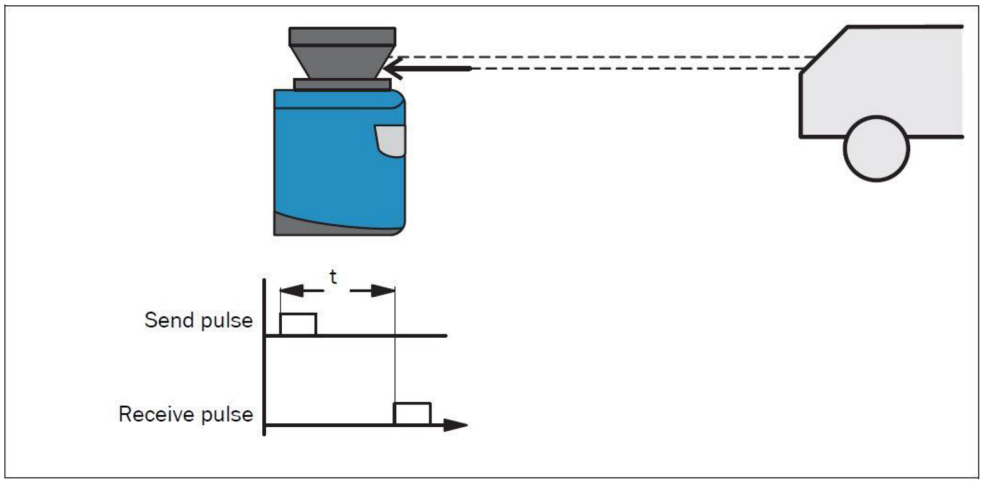
Obr. 2-1 Princip měření LMS111 [1]

2.1.1 Měření vzdálenosti a úhlu

Princip skeneru LMS 111 nespočívá jen v detekci přijetí/nepřijetí odraženého paprsku jako u klasického difuzního optického snímače, ale zde se měří i vzdálenost objektu od čela snímače prostřednictvím měření času doby letu (time-of-flight), podobně jako akustické radary[2].

Snímač v pravidelných intervalech vygeneruje z laserové diody světelný puls, při dopadu paprsku na předmět dojde k odrazu paprsku zpět do skeneru. Tento odražený paprsek je detekován fotodiodou a vzdálenost od předmětu je vypočtena z doby od vyslání po přijetí světelného paprsku.

Paprsek je vychylován pomocí otáčejícího se zrcadla a snímán v pravidelných intervalech za použití úhlového enkodéru. Sekvence měření se postupně provádí v celém skenovaném okolí v úhlu až 270° s frekvenci 25 Hz nebo 50 Hz s krokem 0,25° či 0,5°, což odpovídá 540 až 1080 vyslaných pulsů v čase 20 ms až 40 ms.



Obr. 2-2 Princip měření doby letu světelného pulsu [1].

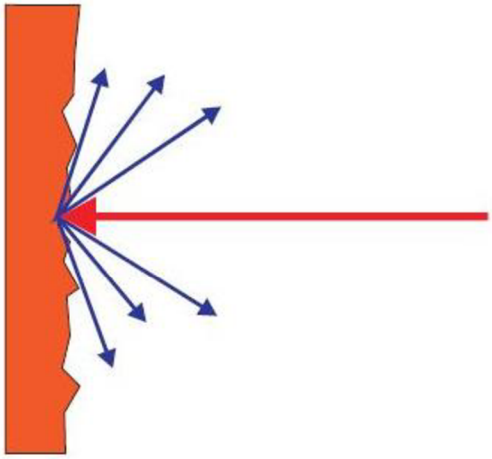
2.1.2 Vliv povrchu objektu na měření

Signál získaný z dokonale difuzního bílého povrchu odpovídá definici 100 % odrazivosti. Z této definice vyplývá, že u povrchů jako zrcadla nebo odrazky bude odrazivost vyšší než 100 % [1].

Intenzita odrazu laserového paprsku je závislá na struktuře a barvě povrchu. Světlé povrchy odrážejí paprsek lépe než tmavé, proto je skener může detekovat na větší vzdálenosti. Jasně bílá omítka odráží přibližně 100 % dopadajícího světla, zatím co černý molitan jen 2,4 %. Na velmi drsných površích se ztrácí část energie z důvodu stínování. V důsledku toho je dosah skeneru nižší. [1]

S barvou povrchu se pojí ještě jeden jev, který má vliv na přesnost měření. Máme-li objekt umístěn v určité vzdálenosti kolmo ke skeneru a jedna jeho část má světlou barvu, zatím co druhá barvu tmavou, bude skener pro tmavou část vyhodnocovat větší vzdálenost. Je to způsobeno skutečností, že odražený paprsek je deformován (prodloužen v čase) a měří se doba kdy je paprsek přijat, skener rozhoduje na základě intenzity signálu o okamžiku přijetí odraženého paprsku. Jelikož odražený paprsek od tmavé barvy má

menší intenzitu, okamžik odečtení bude delší než u světlé, a tudíž i vypočtená vzdálenost bude větší.



Obr. 2-3 Odraz paprsku na povrchu objektu. Většina povrchů odráží paprsek difuzně ve všech směrech [1].

2.1.3 Komunikace skeneru

V této podkapitole se budu věnovat popisu komunikace skeneru s počítačem, respektive minipočítačem Raspberry Pi. Kromě digitálních vstupů a výstupu je skener vybaven čtveřicí datových rozhraní.

Datová rozhraní skeneru:

- Ethernet,
- CAN sběrnice,
- Sériové rozhraní RS-232,
- Pomocné sériové rozhraní RS-232.

V této práci se budu zabývat pouze rozhraním Ethernet, jelikož pro komunikaci s Raspberry Pi budu používat pouze toto rozhraní. Ethernetové rozhraní podporuje přenosovou rychlost dat 10/10MBit. Rozhraní komunikuje po TCP / IP protokolu a podporuje plně duplexní a polo duplexní přenos dat. Přes Ethernet je možné skener konfigurovat a získávat z něj měřené hodnoty. [1]

Komunikace probíhá za pomoci ASCII datových zpráv (tzv. telegramů). Telegram se skládá ze tří částí, první část je tvořena znakem STX (start of text), v prostřední části je zapsán příkaz / odpověď skeneru a poslední řas je tvořena znakem ETX (end of text).

Tabulka 2.1 Příklad ASCII telegramu [3]

ASCII	<STX>	sMN{SPC}SetAccessMode{SPC}03{SPC}F4724744	<ETX>
Hex	02	73 4D 4E 20 53 65 74 41 63 63 65 73 73 4D 6F 64 65 20 30 33 20 46 34 37 32 34 37 34 34	03
	Start	Data	Stop

Tento skener umí komunikovat binárním způsobem kódování, výhodou tohoto způsobu kódování je, že délka datové zprávy je vždy stejná. U ASCII kódování dojde

k odstranění nul na začátku každého hodnoty, a tudíž se délka řetězce mění. Skener automaticky rozpozná kódování příkazu a stejným kódováním odesílá odpověď [3]. Veškeré příkazy se kterými skener dokáže pracovat jsou uvedeny v dokumentaci [4], je dostupná i nová verze seznamu telegramů [3]. Jelikož mám ale skener staršího data výroby se starším firmwarem, musím používat i starší dokumentaci telegramů. V nové dokumentaci jsou uvedeny funkce, které starší typ skeneru neumí. Skener je možné plně konfigurovat a ovládat z počítače softwarem SOPAS Engineering Tool od firmy SICK. Tento software je k dispozici na webových stránkách výrobce a je určený pro celou škálu snímačů.

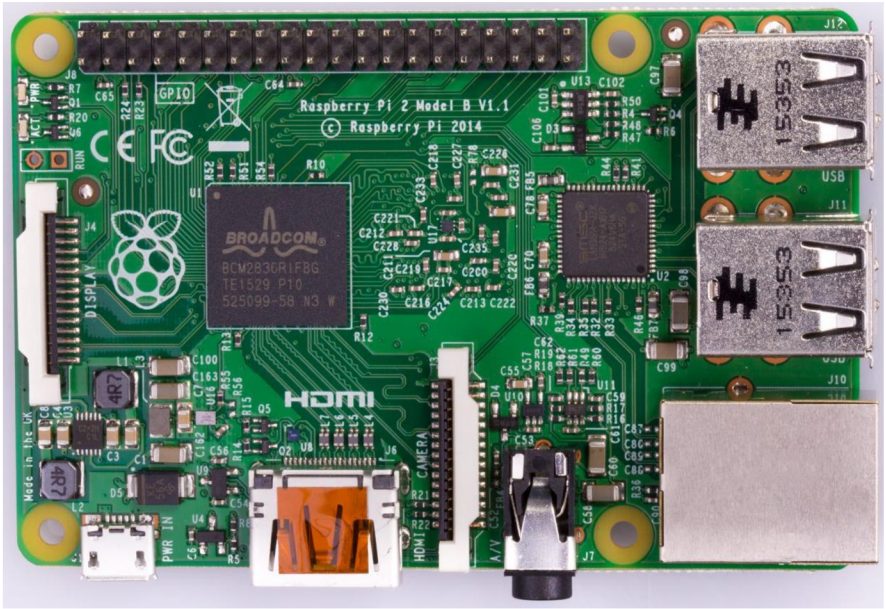
Tabulka 2.2 Příklad binárního telegramu [3]

02 02 02 02	00 00 00 17	73 4D 4E 20 53 65 74 41 63 63 65 73 73 4D 6F 64 65 20 03 F4 72 47 44	B3
Header	Length	Data	CS

2.2 Raspberry Pi 2

Raspberry Pi je jednoduchý počítač o velikosti kreditní karty. Byl navržen britskou nadací Raspberry Pi Foundation jako prostředek pro výuku programování na školách. Raspberry Pi je velice oblíbené pro svoji všestrannost, nízkou spotřebu, malé rozměry, a především příznivou cenu. K dostání je i spousta obdobných zařízení, z nichž asi nejznámější je Arduino, Banana Pi nebo Intel Galileo.

Mnou použité Raspberry Pi 2 B je v pořadí pátým modelem tohoto minipočítače. Do prodeje byl uveden v únoru 2015. Poprvé byl použit 1 GB operační paměti RAM sdílený s grafickým čipem Broadcom. Deska je osazena čtyřjádrovým procesorem Broadcom BCM2836 z rodiny ARM Cortex-A7 taktovaným na 900 MHz, dále čtveřicí USB portů, Ethernetovým konektorem RJ45, HDMI konektorem, napájecím micro USB konektorem, 3,5 mm jack konektorem pro analogový výstup zvuku a videa, slotem na MicroSD kartu, konektorem pro display a kameru a jako poslední čtyřiceti pinový GPIO konektor [5].



Obr. 2-4 Raspberry Pi 2 [7]

2.3 Operační systém Raspberry Pi

Operační systém na Raspberry Pi je uložen a spouštěn z paměťové karty Micro SD. Mnou zvolený operační systém pro Raspberry Pi je Raspbian. Tento výrobcem oficiálně podporovaný operační systém je založen na Linuxovém jádru. Jedná se o optimalizovanou distribuci pro Raspberry PI založenou na Debianu, která je volně ke stažení na stránkách výrobce.

V této práci je použitá verze Raspbian Stretch s datem sestavení 18. 4. 2018. Po připojení monitoru, klávesnice a myši se zařízení chová podobně jako stolní počítač, je možné psát dokumenty, prohlížet Internet, přehrávat video, pomocí předinstalovaných nástrojů programovat aplikace, a dokonce hrát i jednoduché hry. Raspbian je od svého uvedení aktivně vyvíjen a obsahuje přes 35 000 balíčků předkompilovaného software k okamžité instalaci. Cílem týmu, který jej vyvíjí, je stabilita a výkon co nejvyššího počtu balíčků. Jelikož je Raspberry Pi velice oblíbené, a tudíž je kolem něj rozsáhla komunita vývojářů, tak jsou dostupné i jiné operační systémy. Většina jich je založena na různých Linuxových distribucích upravených pro různé účely např. multimediální zařízení k televizoru, domácí NAS server nebo Web server.

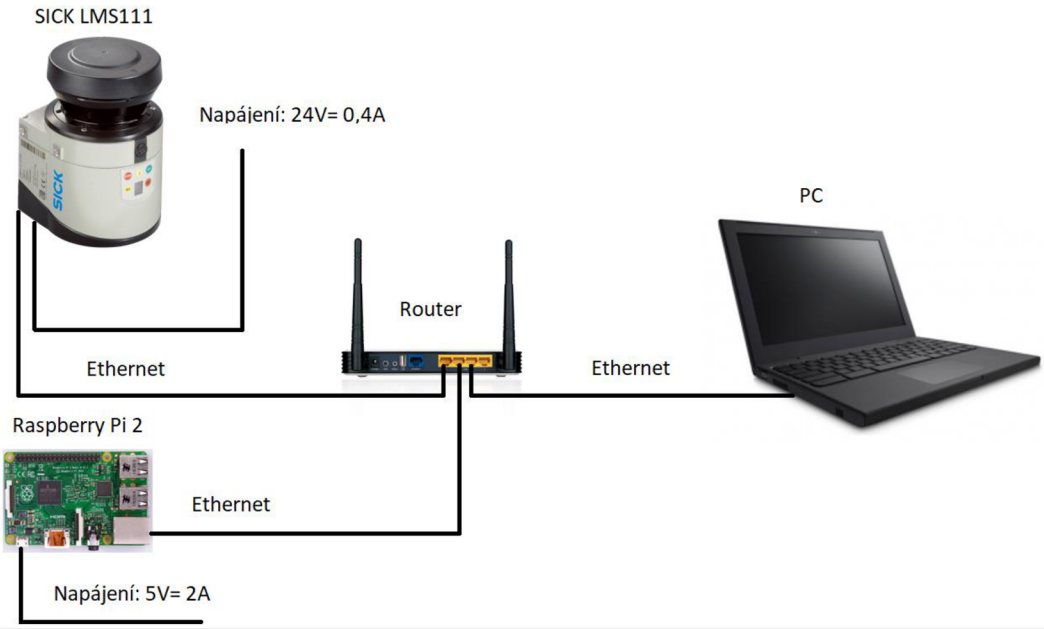
Jedním z operačních systémů, který není založen na Linuxu je Windows 10 IoT od Microsoftu. Vzhledem k tomu, že Windows 10 je poměrně nový systém, nenabízí tolik možností jako Raspbian. Windows 10 IoT jsou oproti desktopovým Windows 10 dosti zeštíhlené, aby mohly fungovat na Raspberry Pi. Označení IoT česky překládané jako Internet věcí je označení pro síť fyzických objektů opatřených senzory a softwarem schopným komunikovat přes Internet. Tyto zjednodušené Windows pro Raspberry Pi nabízí pouze základní přehled a nastavení, tento systém slouží k ovládání Raspberry Pi na dálku. Na běžném počítači s Windows 10 si v Microsoft Visual Studiu naprogramujete aplikaci, kterou nahrajete do Raspberry Pi a spustíte [5].

3. NÁVRH ŘEŠENÍ

3.1 Zapojení

Skener je s Raspberry Pi a počítačem spojen ethernetovým kabelem přes router případně switch. Router využívám, abych byl schopný komunikovat přes počítač s Raspberry Pi a zároveň i se skenerem. Při vlastním požití v určené aplikaci nebude již router nutný, k Raspberry Pi se může připojit například USB Wi-fi adaptér a skener se bude ovládat pomocí mého ovladače prostřednictvím Raspberry Pi.

Skener je napájen ze zdroje v rozsahu 11 V ss až 30 V ss (typicky 24 V ss při 0,4 A). Na originálním přívodním kabelu je kladný pól na hnědém vodiči a záporný pól na modrém vodiči. Raspberry Pi je napájeno přes microUSB konektor zdrojem 5 V ss a alespoň 2 A. Maximální potřebný proud se odvíjí od připojených periferií.



Obr. 3-1 Zapojení Skeneru [6], Raspberry Pi [7], routeru[8] a PC[9]

3.2 Programovací jazyk a vývojové prostředí

Pro naprogramování ovladače jsem si již v Semestrální práci vybral programovací jazyk C++, protože jsem se jej učil již na střední škole a dále jsem si jeho znalosti prohloubil v předmětu Praktické programování v C++. Dále proto, abych byl schopný rozhraní zpracovat tak, aby bylo použitelné i pro jiné laserové skenery. Cílem bylo aby nadřazená aplikace přistupovala k různým skenerům totožně. Toho dosáhnu využitím dědičnosti a polymorfismu a vytvořením tzv. abstraktní třídy Skener.

Klasické C++ jsem rozšířil použitím Qt knihovny a jejich funkcí. Qt je multiplatformní knihovna pro tvorbu grafického uživatelského rozhraní (GUI), která navíc obsahuje i řadu dalších nástrojů: podpora zpracování XML, práce s databází a multimédii, síťové programování, vícevláknové programování. Tato univerzálnost a velice podrobně zpracovaná dokumentace, plná příkladů, jsou největší přednosti Qt knihovny [10]. Mezi podporované platformy patří: Linux, OS X, Windows, Android, iOS a další. Díky požití této knihovny je možné mnou vytvořený ovladač používat jak pod Windows, tak pod Linuxem v podstatě bez úprav. Dále je možné ovladač použít ve vícevláknových aplikacích pro větší efektivitu u vícejadrových procesorů. Qt má vlastní vývojové prostředí, které se nazývá Qt Creator. Toto vývojové prostředí běží pod Windows, Linuxem a OS X a nabízí inteligentní doplňování kódu, zvýraznění syntaxí, integrovanou nápovědu, příklady a další užitečné možnosti. Pro programování v Qt je též možné použít doplněk do Microsoft Visual Studia [11] anebo programovat ve vývojovém prostředí NetBeans IDE. NetBeans IDE je volně dostupné vývojové prostředí, které podporuje řadu programovacích jazyků (PHP, JavaFX, C/C++, Javascript a další). NetBeans je vyvíjí forma Orcale Corporation [12].

Pro vývoj ovladače jsem požíval Microsoft Visual Studio s doplňkem Qt Visual Studio Add-in protože jsem zvyklý na jeho prostředí. Výhodou je i rychlejší kompilace a spouštění než u Raspberry Pi. Pro otestování funkčnosti knihovny ovladače jsem použil NetBeans IDE, které se propojí s Raspberry Pi a následně je možné kompilovat a spouštět aplikace přímo v Raspberry Pi.

3.3 Vstupy a výstupy ovladače

Vstupními daty do ovladače jsou datové telegramy odesílané skenerem, které jsou dále zpracovány. Výstupem ovladače jsou naměřená data v polárních souřadnicích. Pro výstup z ovladače slouží dvojice funkcí.

3.3.1 Funkce MeasData

Právní z nich *MeasData* slouží pro načtení dat ze skeneru a uložení jednotlivých naměřených hodnot v [mm] do proměnné *MData* typu *QVector<double>*. Třída QVector je třída šablony, která poskytuje dynamické pole. QVector <T> je jedna z generických kontejnerových tříd Qt. Uloží své položky do sousedních paměťových míst a poskytuje k nim rychlý přístup založený na indexech [13].

```
int LMS111::MeasData(QVector<double>& MData, uint& TimeSinceStart,
                     long int& StartAngle, int& Steps) override;
```

Zdrojový kód 3.1: Hlavička funkce MeasData

Dále funkce *MeasData* uloží čas od spuštění skeneru v [µs] do proměnné *TimeSinceStart*, do proměnné *StartAngle* uloží počáteční úhel měření v [1/10000°] a do proměnné *Steps* uloží úhlové rozlišení, což znamená úhel mezi jednotlivými měřícími body v [1/10000°]. Počet naměřených bodů zjistíme příkazem `int PocHodnot = MData.size()`.

Na základě těchto dat jsme schopni dopočítat konečný úhel měření podle rovnice (3.1).

$$StopAngle = (PocHodnot - 1) \cdot Steps + StartAngle \qquad \textbf{(3.1)}$$

Implementaci tohoto výpočtu ponechám na uživateli, protože předpokládám, že si uživatel skener nastaví dle svých potřeb a tuto rovnici může použít jen jako kontrolu správného nastavení skeneru.

Funkce *MeasData* rozdělí příchozí telegram na jednotlivé části, které jsou oddělené mezerou a uloží je do pole Q. Zde jsou nyní uložená jednotlivá slova telegramu jejichž detailní popis je dostupný v seznamu telegramů [4] na straně 25. Nyní můžu z pole Q vyčíst potřebné hodnoty pro proměnné *MData*, *TimeSinceStart, StartAngle, Steps,* tyto hodnoty je ovšem potřeba převést z hexadecimální soustavy do soustavy decimální.

Pro výpočet měřených bodů je nutné načíst z pole Q a převést do decimální soustavy hodnoty Scalefactor a Scalefactor offset. Poté vypočteme vzdálenost měřeného bodu podle rovnice (3.2).

$$Vzdalenost = ScaleFactorOffset + ScaleFactor \cdot MerenyBod\ [mm] \qquad \textbf{(3.2)}$$

3.3.2 Funkce SaveData

Druhá funkce pro výstup dat je funkce s názvem *SaveData*. Tato funkce uloží získaná data z funkce *MeasData* do souboru MerData.csv, a to tím způsobem, že připadá jedno měření na jeden řádek a sloupce jsou poté jednotlivé naměřené body. Cesta k tomuto souboru je: \Bakalarka\Ovladac\MerData.csv.

Při novém spuštění programu je souboru vytvořen, nebo pokud existuje, jsou hodnoty do souboru ukládány pod hodnoty stávající. S takto vytvořeným a naplněným souborem můžeme dále libovolně pracovat.

```
int LMS111::SaveData(QVector<double>& MData)
{
	QFile file("MerData.csv");

	if (!file.open(QIODevice::WriteOnly | QFile::Append))
	{
		std::cerr << "Cannot open file for writing: "
			<< qPrintable(file.errorString()) << std::endl;
		return -1;
	}

	QTextStream out(&file);
	int Delka = file.size();

	for (int i = 0; i < MData.size(); i++)
	{
		out << MData[i] << ";";
	}

	out << "\n";
	file.close();
	return 0;
}
```

Zdrojový kód 3.2: Funkce SaveData

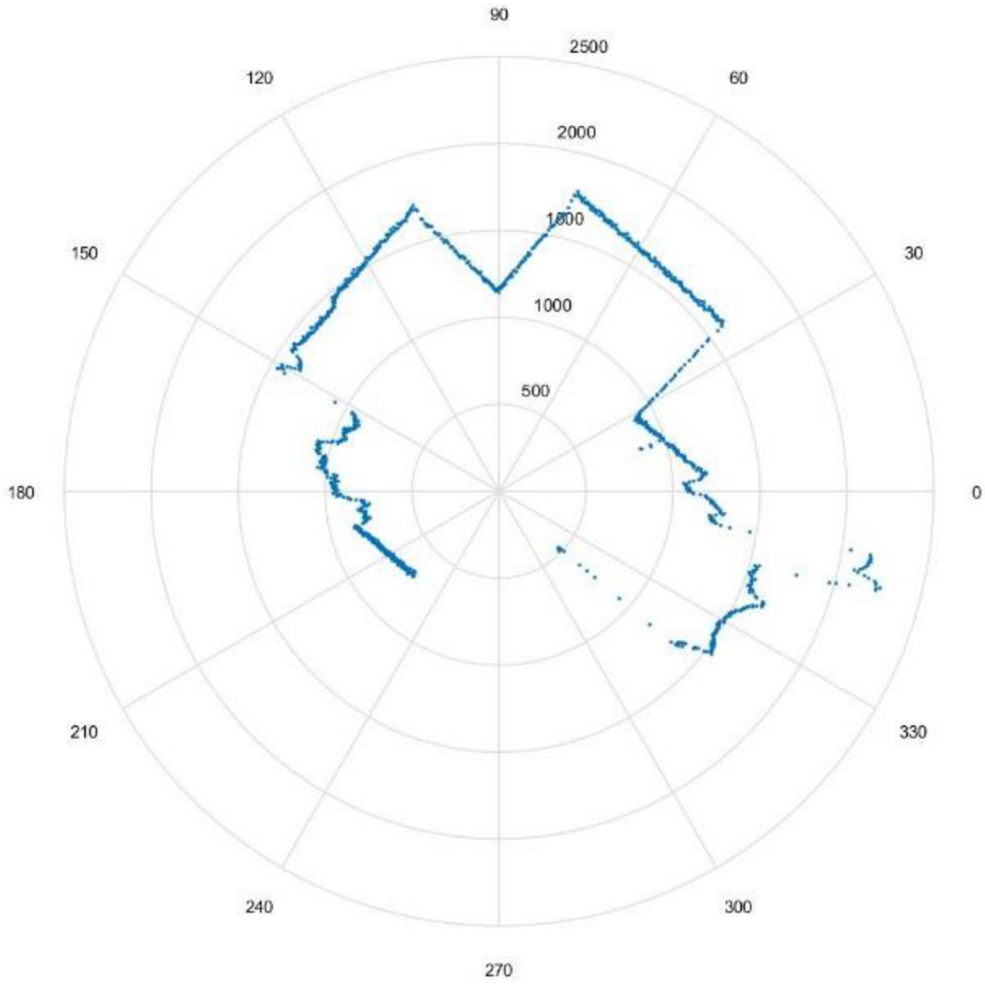
3.4 Vizualizace naměřených hodnot

I přesto, že vizualizace dat není součástí zadání, jsem se rozhodl ji udělat. Domnívám se, že je to výborná kontrola správné funkce ovladače. Pro vizualizaci jsem zvolil program Matlab.

Vytvořil jsem soubor polar_graf.m, který pomocí funkce *importfile* načte hodnoty ze souboru MerData.csv a uloží je do matice MerData. Dále je nutné vytvořit vektor úhlů podle nastavení skeneru. Například pokud skenuji v celém rozsahu skeneru s úhlovým rozlišením 0,25°, nadefinujeme vektor od -45° do 225° s krokem 0,25° příkazem st = (-45:0.25:225). Následně je nutné tento vektor ve stupních převést na radiány. Poté můžeme vykreslit polární graf funkcí *polar*. Do závorky vložíme jako první parametr vektor úhlu v radiánech a jako druhý parametr řádek z matice který chceme vykreslit. Ukázkový polární graf naměřených hodnot viz. Obr. 3-2.

```
close all
clear all
clc

st=(-45:0.25:225);
st_radians = deg2rad(st);
%% import dat
MerData = importfile('MerData.csv');
%% vykresleni grafu
figure(1)
polar(st_radians,MerData(1,:),'.');
```

Zdrojový kód 3.3: Vykreslení dat v Matlabu



Obr. 3-2 Ukázka grafu z Matlabu

3.5 Realizace knihovny ovladače

Mnou realizovaná knihovna ovladače je tvořena dvěmi hlavičkovými soubory a jedním zdrojovým souborem. Celá knihovna je zdokumentovaná pomocí programu Doxygen. Podrobná dokumentace všech metod je ve formátu PDF obsažená v přiloženém DVD.

3.5.1 Abstraktní třída Skener

První soubor *Skener.h* obsahuje abstraktní třídu *Skener*. Abstraktní třída poskytuje jednotné rozhraní pro přístup ke svým potomkům a obsahuje čistě virtuální metody, které slouží pouze jako předpis a nelze v této třídě vytvořit jejich implementaci. Implementace metod ovšem musí být vytvořena v potomcích dané abstraktní třídy.

V mém případě třída *Skener* slouží k tomu, aby z ní mohli dědit různé typy skenerů, a přitom každý bude mít svoji vlastní implementaci čistě virtuální metody. I přes rozdílné implementace čistě virtuálních metod s nimi může aplikace zacházet stejně jako kdyby to byly dva totožné skenery.

V této třídě jsou ještě nadefinované dvě struktury, které jsou předávány parametrem z metody *AskOutputStat*. První struktura *Output* slouží pro zapouzdření stavu a čítače jednoho výstupu. Tato struktura je vnořena do druhé struktury *OutputState*, která zapouzdřuje informace o stavech a čítačích třech interních a osmi externích výstupů. Navíc zapouzdřuje informaci o stavovém kódu.

```
class Skener
{
public:
	struct OutputState
	{
		struct Output
		{
			int State = 0;
			int Count = 0;
		};

		int StatusCode[2] = {0, 0};
		Output Interni[3];
		Output Externi[8];
	};

	virtual int Connect(QString IP, int Port) = 0;
	virtual int Disconnect() = 0;
	virtual int DeviceState() = 0;
	virtual int GetDataTime(int& Status, bool& TempOutRange, QDate& Date,
			QTime& Time, bool& LED1, bool& LED2, bool& LED3) = 0;
	virtual int GetOneScanData(QByteArray& Telegram) = 0;
	virtual int GetScanDataPermanent(bool Stav) =0;
	virtual int AskOutputState(OutputState& OutStat) = 0;
	virtual int SetOutputState(int OutputNumber, bool State) = 0;
	virtual int GetStatusDevice() = 0;
	virtual int DeviceIdent(QByteArray& DeviceIdent) = 0;
	virtual int DeviceName(QByteArray& DeviceName) = 0;
	virtual int GetContaminationValues(int& Strategy, int& ResponseTime,
				int& ThresholdWarning, int& ThresholdError) = 0;
	virtual void SopasErrorCodes(QByteArray Telegram) = 0;
```

```
    virtual int MeasData(QVector<double>& MData, uint& TimeSinceStart,
                         long int& StartAngle, int& Steps) = 0;
    virtual int GetScancfg(int& ScanFreq, int& AngResol, int& StartAngle,
                         int& StopAngle) = 0;
    virtual int ReadOutputRange(int& AngResol, int& StartAngle, int&
                                StopAngle) = 0;
    virtual int SaveData(QVector<double>& MData) = 0;

    virtual ~Skener() = default; //d'tor
};
```

Zdrojový kód 3.4 Abstraktní třída Skener

3.5.2 Třída LMS111

Další hlavičkový soubor LMS1xx.h obsahuje definici třídy *LMS111* a třídy *Token*. Tyto dvě třídy slouží pro plnohodnotné ovládání skeneru SICK LMS111. Implementace všech metod jsou umístěny ve zdrojovém souboru LMS1xx.cpp

Třída *LMS111* je potomkem třídy *Skener,* a tudíž musí obsahovat implementaci všech metod, které jsou nadefinované v abstraktní třídě *Skener*. Slouží k připojení a odpojení skeneru, dále k vyčítání uloženého nastavení ve skeneru, a především ke čtení a zpracování naměřených dat. Při vytvoření instance této třídy je v konstruktoru vytvořen ukazatel na třídu *QTcpSocket* s názvem *mSocket* zajišťující komunikaci se skenerem.

V následujících podkapitolách se budu věnovat těm nejdůležitějším metodám této třídy. Popis všech metod je možné nalézt v přiložené dokumentaci.

3.5.2.1 Metoda Connect

```
int Connect(QString IP = "192.168.0.1", int Port = 2111) override;
```

Zdrojový kód 3.5: Hlavička metody Connect

Pravděpodobně dvě nejvýznamnější metody, slouží pro připojení (*Connect*) a odpojení (*Disconnect*). Metoda *Connect* slouží pro připojení ke skeneru, na zadané IP adrese a TCP portu. Defaultní hodnota IP adresy je 192.168.0.1, jelikož je to i výchozí hodnota nastavená na skeneru od výrobce. Defaultní hodnota TCP portu je nastavena na 2111, skener komunikuje i na portu 2012. Po kontrole správnosti zadaných parametru je proveden pokus o připojení ke skeneru. Pokud se do nastaveného času připojení nepodaří, na konzoli je vypsána informace "Connect: Nepripojeno" a metoda vrací číslo -3. Proběhne-li přihlášení ke skeneru úspěšně, na konzoli je vypsána informace "Connect: Pripojeno!" a metoda vrací hodnotu 0.

3.5.2.2 Metoda Disconnect

V metodě Disconnect je nejdříve smazán ukazatel na třídu Token, pokud je nadefinovaný. Následně je otestován stav připojení, jestliže skener není připojen, uživatel je informován výpisem na konzoli a metoda vrací hodnotu -1. Pokud je skener připojen, je odeslán příkaz k ohlášení, jestliže se odhlášení nezdaří metoda vrací hodnotu -2. Po úspěšném odpojení od skeneru je zavřen mSocket a uživatel je informován o uspěšném odpojení, metoda vrací hodnotu 0.

3.5.2.3 Metoda WriteTelegram

```
int WriteTelegram(const QByteArray& WriteText, QByteArray& Answer);
```

Zdrojový kód 3.6: Hlavička metody WriteTelegram

Další velice důležité metody jsou *WriteTelegram* a *SocketRead* zajištující odesílání a přijímání telegramů. První jmenovaná metoda slouží k odeslání telegramu, který je uložený v parametru *WriteText*, do skeneru. Následně se z této metody zavolá metoda *SocketRead* fungující jako buffer přijatých telegramů.

Vrátí-li se z metody *SocketRead* platný a úplný telegram, je otestován nejedná-li se o chybu metodou *SopasErrorCodes*, funkce této metody je vysvětlena níže. Jestliže se o chybu nejedná je telegram uložen do parametru *Answer*. Předávání dat pomocí parametru ven z funkce je provedeno přes reference, poněvadž je tento způsob rychlý a šetrný k paměti.

3.5.2.4 Metoda SocketRead

```
QByteArray SocketRead(int& Chyba);
```

Zdrojový kód 3.7: Hlavička metody SocketRead

V této metodě se ukládají přijaté telegramy do privátní proměnné *mDataReceived*. Pokud je v této proměnné již celý telegram, začínající \x02 a končící \x03, je tento telegram poslán přes návratovou hodnotu zpět do metody *WriteTelegram* a do proměnné *Chyba je* uloženo číslo 0.

Pokud je přijat neúplný telegram a za ním následně přijde telegram kompletní, neúplný telegram je vymazán. Jestliže do 7 vteřin nepřijde žádný telegram na konzole se vypíše chybová hláška *"SocketRead: TimeOut"*, do proměnné *Chyba* je uložena hodnota -2 a je vrácena prázdna proměnná typu *QByteArray*.

Vzhledem k tomu, že ovladač vytvářím, aby jej bylo možné použít ve více vláknech, je nutné v této metodě implementovat funkce knihovny *QMutex*. Ve více vláknových aplikacích je nutné zajistit, aby k proměnné přistupovalo v daný okamžik pouze jedno vlákno, toho lze dosáhnout použitím tzv. mutexů. Vytvořením instance *Lock* třídy *QMutexLocker* s parametrem reference na privátní proměnnou *mMutex* typu *QMutex* dosáhneme, že od vytvoření instance *Lock* do skončení bloku, ve kterém byla instance vytvořena, budou ostatní vlákna pozastavena. Tím zajistíme, že nám k proměnným v jeden okamžik nebude přistupovat více vláken.

3.5.2.5 Třída ErrorCode a metoda SopasErrorCodes

Součástí této třídy je i třída *ErrorCode* dědící z třídy *std::exception*. Tato třída byla vytvořena za účelem vyvolání vlastní výjimky nesoucí informaci o čísle chyby skeneru. Každá odpověď, která přijde ze skeneru je poslána do metody *SopasErrorCodes*, kde je otestována, zda se nejedná o telegram informující o chybě skeneru. Tato chyba má tvar *"\x02sFA x\x03"*, kde x představuje číslo chyby. Seznam a popis všech chyb je možné nalézt seznamu telegramů [4] na straně 75.

```
class LMS111 : public Skener
{
	friend class Token;
	int mTimeOut = 3000;
	QMutex mMutex;
	Token* mToken = nullptr;
	QTcpSocket* mSocket = nullptr;

protected:

	QByteArray mDataReceived;

	int WriteTelegram(const QByteArray& WriteText, QByteArray& Answer);

public:

class ErrorCode : public std::exception
	{
		friend class Threads;
		friend class LMS111;
		int Error = 0;

	public:

	ErrorCode (int Chyba)
	{
		Error=Chyba;
	}
		virtual const char* what();
		int GetError();
	};

	LMS111(void);
	QByteArray SocketRead(int& Chyba);
	int Connect(QString IP = "192.168.0.1", int Port = 2111) override;
	int Disconnect() override;
	int DeviceState() override;
	int GetDataTime(int& Status, bool& TempOutRange, QDate& Date,
			QTime& Time, bool& LED1, bool& LED2, bool& LED3) override;
	int GetOneScanData(QByteArray& Telegram) override;
	int GetScanDataPermanent(bool Stav) override;
	int AskOutputState(OutputState& OutStat) override;
	int SetOutputState(int OutputNumber, bool State) override;
	int GetStatusDevice() override;
	int DeviceIdent(QByteArray& DeviceIdent) override;
	int DeviceName(QByteArray& DeviceName) override;
	int GetContaminationValues(int& Strategy, int& ResponseTime,
				int& ThresholdWarning, int& ThresholdError) override;
	void SopasErrorCodes(QByteArray Telegram) override;
	int MeasData(QVector<double>& MData, uint& TimeSinceStart,
			long int& StartAngle, int& Steps) override;
```

```
        int GetScancfg(int& ScanFreq, int& AngResol, int& StartAngle,
                       int& StopAngle)override;
        int ReadOutputRange(int& AngResol, int& StartAngle, int& StopAngle)
                            override;
        int SaveData(QVector<double>& MData) override;

        virtual ~LMS111();
};
```

Zdrojový kód 3.8 Třída LMS111

3.5.3 Třída Token

Třída Token slouží k přihlášení ke skeneru. Pro odesílání konfiguračních příkazů je nutné být ke skeneru přihlášen. Jsou definovány tři úrovně přihlášení, které se zadávají jako číselný parametr *UserLevel* do konstruktoru. Úrovně přihlášení a jejich číselné reprezentace jsou:

- Maintenance neboli údržba, *UserLevel* = 2
- Authorized client neboli oprávněný klient, *UserLevel* = 3
- Service neboli servis, *UserLevel* = 4

Čili při vytvoření instance třídy *Token* doje k přihlášení do skeneru a tím se aktivuje konfigurační režim skeneru. V tomto okamžiku je možné volat konfigurační metody z třídy Token a jelikož je v této třídě i ukazatel na instanci třídy LMS111 je možné volat i metody z této třídy. Konfigurační režim je aktivní po dobu existence instance třídy Token, jelikož v destruktoru této třídy dochází k odhlášení od skeneru. Navíc se zde vyčká, dokud není skener připraven.

Výhodou tohoto řešení je, že k odhlášení od skeneru dojde automaticky, a to i v případě, že nastane chyba v průběhu konfigurace. Ukončení konfiguračního režimu je možné zajistit umístěním celé konfigurační sekvence do bloku ohraničeného složenými závorkami. Jak je možné vidět na Zdrojový kód 3.9. Jakmile překladač dojde na uzavřenou složenou závorku zavolá destruktor a dojde k odhlášení od skeneru.

```
LMS111 Lms111;

if (Lms111.Connect("10.0.0.74", 2111) != 0)
        return;

{
        Token Tok(&Lms111, 3); // login do konfiguracniho rezimu
        Tok.SetScancfg(2500, 2500);
        Tok.OutputRange(-450000, 2250000);
        Tok.SaveParamPermanent();
} //logout z konfiguracniho rezimu
```

Zdrojový kód 3.9: Příklad použití třídy Token

Důležitou metodou této třídy je *SaveParamPermanent*. Tato metoda slouží trvalému uložení nastavení do paměti skeneru. Proto je vhodné před ukončením konfiguračního režimu tuto metodu zavolat. Ovšem zápis do paměti nějakou dobu trvá proto je třeba počítat s tím, že odhlášení od skeneru bude trvat delší dobu.

Dále stojí za zmínku metoda pro nastavení časového razítka *SetDataTime*, tuto metodu je možné přetížit a nastavení parametrů lze realizovat jednak využitím tříd QDate a QTime nebo zadáním jednotlivých číslic času a datumu. Pokud není některý z parametrů uveden, nastaví se podle systémového času počítače.

Ještě je třeba podotknout, že po zavolání metody RebootDevice dojde k restartování skeneru. Tudíž k odhlášení i odpojení skeneru od počítače. Restartování trvá přibližně 65 vteřin, během kterých je vlákno uspáno a vyčkává až bude skener připraven na opětovné připojení.

```
class Token
{
	LMS111* A;
	bool Restart = false;

public:

	Token(LMS111* A, int UserLevel);
	~Token();

	int SetDataTime(QDate Date, QTime Time);
	int SetDataTime(int Year = -1, int Month = -1, int Day = -1,
                        int Hour = -1, int Minute = -1, int Second = -1,
                        int MicroSecond = -1);
	int SaveParamPermanent();
	int SetScancfg(int ScanFreq, int AngRes);
	int OutputRange(int StartAngle, int StopAngle);
	int ScanDatacfg(int DataChannel, bool Remission, bool Resolution,
                       int Encoder, bool Position, bool DeviceName,
                       bool Comment, bool Time, int OutputRate);
	int RstOutputCounter();
	int RebootDevice();
	int SetContaminationValues(int Strategy, int ResponseTime,
                                   int ThresholdWarning, int ThresholdError);
	int StartMeas();
	int StopMeas();
};
```

Zdrojový kód 3.10: Třída Token

3.6 Příklad použití ovladače

V této kapitole představím dva příklady využiti ovladače laserového skeneru. První trošku složitější příklad využívá vícevláknových možností QT knihovny. Toto řešení výhodnější z výpočetního i praktického hlediska. Pokud by k Raspberry Pi byly připojeny dva skenery, může každý běžet ve vlastním vlákně a tím šetřit výpočetní výkon. Druhý příklad je pouze jednovláknový a tudíž jednodušší.

Oba ukázkové programy mám otestované jak na Windows, tak na Raspberry Pi s Raspbianem a dokonce i na desktopovém Ubuntu. Na všech zařízeních aplikace pracovaly stejně bez úprav a chyb. Mezi oběma ukázkami je možné přepínat v souboru *main.cpp*, záměnou čísla 1 za číslo 0 respektive čísla 0 za číslo 1 na 8. a 64. řádku.

3.6.1 Vícevláknový příklad použití ovladače

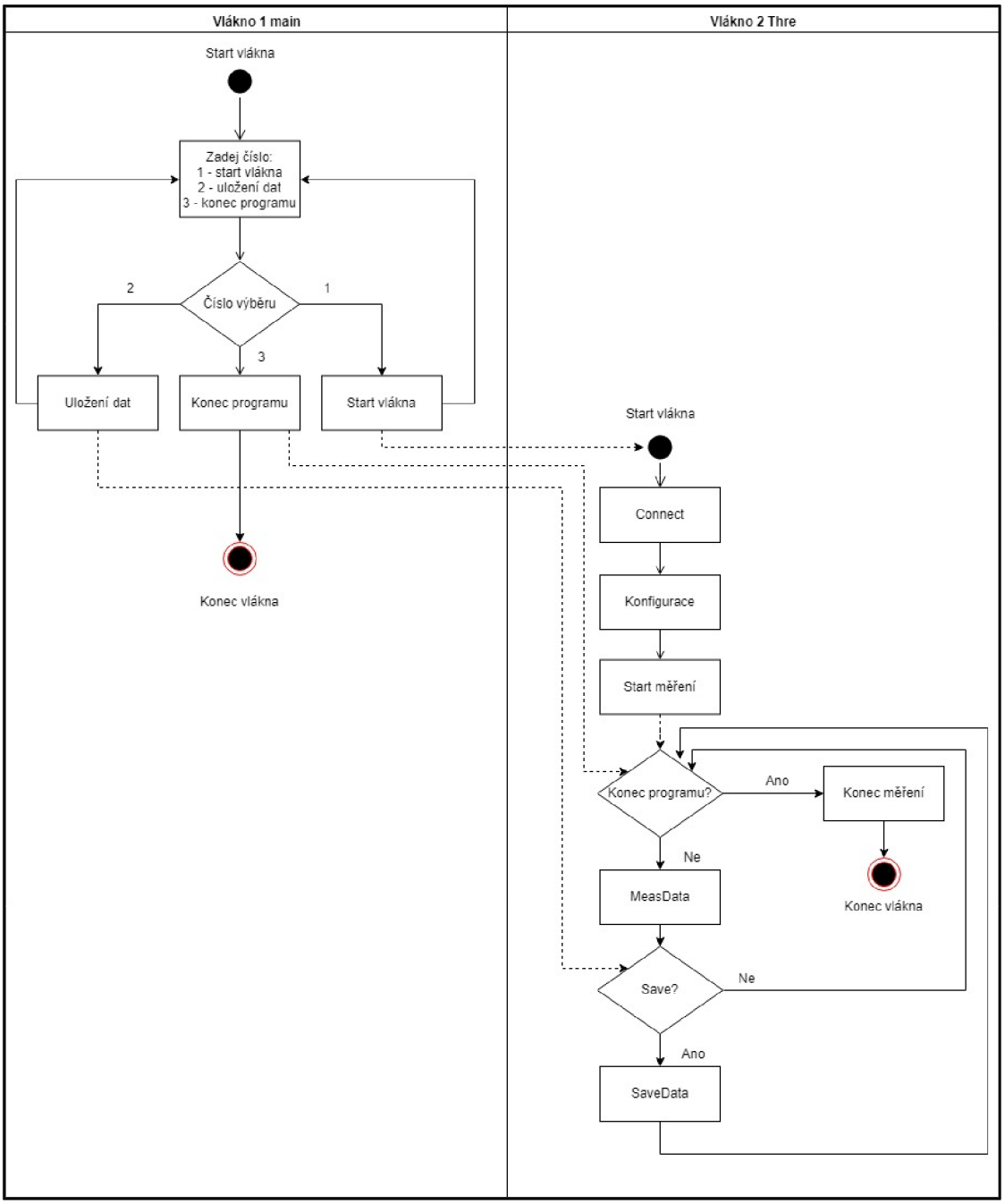
V mém příkladu v hlavním vlákně běží uživatelské rozhraní, v něm je možné spustit druhé vlákno, uložit naměřená data do souboru a ukončit celý program. Výhodou tohoto řešení je, že uživatelské rozhraní je aktivní i když probíhá například zpracování dat v druhém vlákně. Funkce programu je graficky znázorněna na vývojovém diagramu viz. Obr. 3-3.

3.6.1.1 Třída Threads

Třída *Threads* je potomek třídy QThread a LMS111. Její nejdůležitější metoda se jmenuje *run*. V této metodě je napsaný kód, který se po spuštění vlákna začne vykonávat. Dále obsahuje metodu *SaveChange* sloužící k bezpečnému přepínání proměnné *Save*. Jelikož se do této proměnné zapisuje z více vláken je třeba užít mutexů. U proměnné Stop není ošetření potřeba, protože se do ní zapisuje pouze ve funkci main

```
class Threads : public QThread, protected LMS111
{
Q_OBJECT

public:
      bool Stop = false, Save = false;
      QMutex mMutex;

      void SaveChange(bool Stav)
      {
            QMutexLocker Lock(&mMutex);
            Save = Stav;
      }

      explicit Threads(QObject* parent = 0);

      virtual ~Threads();

      void run() override;
   };
```

Zdrojový kód 3.11: Třída Threads



Obr. 3-3 Vývojový diagram vícevláknového příkladu

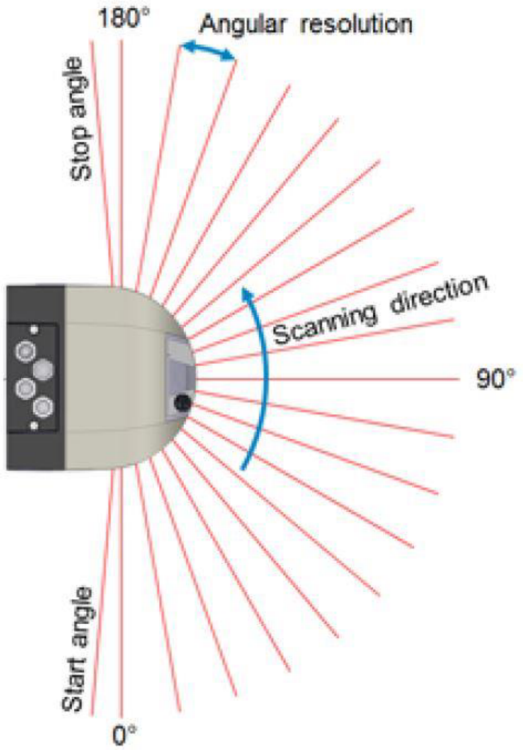
Pro vytvoření vlákna je nutné vytvořit instanci třídy *Threads*. Následně se provede jeho spuštění zavoláním metody *start()*. Tímto příkazem se spustí vlákno a jsou vykonávány instrukce v metodě *run*. V případě tohoto příkladu se nejdříve vytvoří instance třídy *LMS111*, následně se provede pokus o připojení skeneru na IP adresu 10.0.0.74 a TCP port 2111.

Po úspěšném připojení je vytvořena instance třídy *Token* s parametry: reference na již vytvořenou instanci třídy LMS111 a číslem 3, což odpovídá oprávnění Autorizovaný klient. Tím je aktivován konfigurační režim. Jako první konfigurační instrukce je `Tok.SetScancfg(2500, 2500)`. Tato metoda slouží k nastavení skenovací frekvence a

úhlového rozlišení. První parametr nastavuje skenovací frekvenci v [1/100 Hz] a druhý parametr nastavuje úhlové rozlišení v [1/10000 °].

Druhá instrukce je `Tok.OutputRange (-450000, 2250000)`, ta slouží k nastavení počátečního a konečného úhlu skenování, tyto hodnotu jsou zadány v [1/10000 °]. Takže skener nyní bude měřit v plném rozsahu 270 ° s nejmenší frekvencí 25 Hz a nejnižším úhlovým rozlišením 0,25 °. Postup měření a význam jednotlivých veličin je znázorněn na Obr. 3-4. Takto nastavený skener naměří 1081 bodů.

Poslední konfigurační instrukce má za úkol uložit toto nastavení do paměti skeneru. Po úspěšném uložení je zavolán destruktor třídy *Threads*, čímž je ukončen konfigurační režim.



Obr. 3-4 Znázornění měření skeneru LMS111 [3]

Následuje metoda `Lms111.GetScanDataPermanent(true)`, která má za úkol spustit permanentní odesílání naměřených dat. Jako parametr má proměnnou typu bool, pokud je hodnota True, odesílání hodnot bude spuštěné, pokud má hodnotu False, odesílání hodnot bude zastavené.

Po úspěšném spuštění odesílání naměřených dat je v cyklu while volána metoda `Lms111.MeasData(Pole, Time, StAng, Step)`. Výstupem této metody je proměnná Pole ve které jsou uložené naměřené body. Pokud je proměnná *Save* nastavena na True, je volána metoda `Lms111.SaveData(Pole)`. Zavoláním této metody jsou naměřená data uložena do souboru MerData.csv. Uživatel je o úspěšném uložení informován zprávou na konzoli. Jakmile je proměnná Stop nastavena na True, je cyklus while ukončen, odesílání naměřených dat zastaveno a toto vlákno ukončeno.

Nastane-li v průběhu metody *run* chyba skeneru, která vyvolá výjimku, tato výjimka je odchycena a vlákno ukončeno. Uživatel je o tomto stavu informován výpisem čísla chyby na konzoli.

```
void Threads::run()
{
	LMS111 Lms111;
	try
	{
		if (Lms111.Connect("10.0.0.74", 2111) != 0)
			return;

		{
			Token Tok(&Lms111, 3); // login do konfiguracniho rezimu
			Tok.SetScancfg(2500, 2500);
			Tok.OutputRange(-450000, 2250000);
			Tok.SaveParamPermanent();
		} //logout z konfiguracniho rezimu

		Lms111.GetScanDataPermanent(true);

		while (!Stop)
		{
			QVector<double> Pole;
			uint Time;
			int Step;
			long int StAng;
			Lms111.MeasData(Pole, Time, StAng, Step);

			if (Save)
			{
				int Chyba = Lms111.SaveData(Pole);
				if (Chyba == 0)
					qDebug() << "Namerena data byla uspesne
                                                 ulozena do souboru";
				SaveChange(false);
			}
		}

		if (Stop)
		{
			Lms111.GetScanDataPermanent(false);
			quit();
		}
	}
	catch (ErrorCode& Er)
	{
		qDebug() << Er.what();
		Er.GetError();
		quit();
	}
}
```

Zdrojový kód 3.12: Metoda Threads::run()

3.6.1.2 Funkce main

Tato funkce slouží jako jednoduché uživatelské prostředí pro ovládání aplikace. Nejdříve je vytvořena instance třídy *Threads*. Následuje while cyklus, ve kterém se na konzoli vypíše nabídka možností. Následuje příkaz switch, který podle vybrané možnosti z nabídky spustí příslušnou část kódu.

První z možností slouží ke startu druhého vlákna, zavoláním metody `Thre.start()`. Druhá z možností slouží pro uložení naměřených dat do souboru. Poslední z možností nastaví druhému vláknu nejvyšší prioritu a nastaví jeho proměnou *Stop* na True, čím dá druhému vláknu povel k ukončení. Následně je první vlákno uspáno, dokud druhé není ukončené. Poté je celá aplikace ukončena.

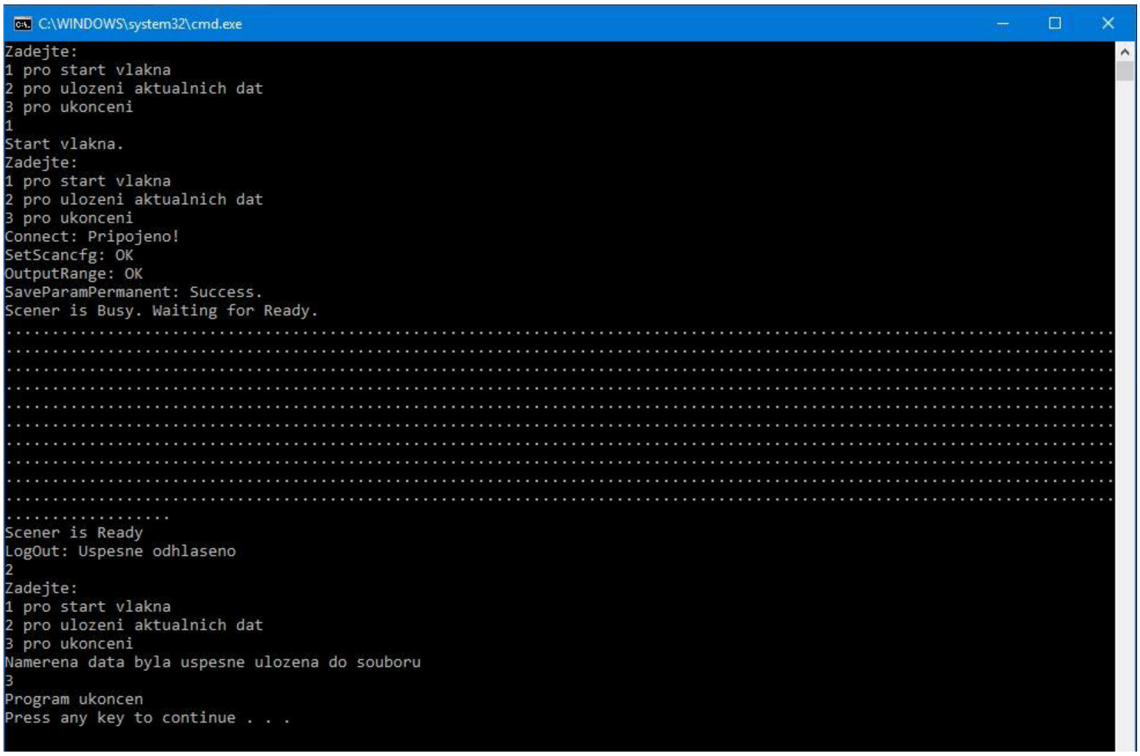
```
int main()
{
	bool Stop = false;
	Threads Thre;

	while (Stop == 0)
	{
		int Vyber = 0;
		qDebug() << "Zadejte: \n1 pro start vlakna \n2 pro ulozeni
			 aktualnich dat \n3 pro ukonceni ";
		std::cin >> Vyber;
		switch (Vyber)
		{
			case 1:
				qDebug() << "Start vlakna.";
				Thre.start();
				break;

			case 2:
				Thre.SaveChange(true);
				break;

			case 3:
				Stop = true;
				if (Thre.isRunning())
				{
					Thre.setPriority(QThread::HighPriority);
					Thre.Stop = true;

					while (!Thre.isFinished())
					{
						QThread::msleep(1);
					}
				}
				qDebug() << "Program ukoncen";
				break;

			default:
				qDebug() << "Spatne zadani!!";
				break;
		}
	}
```

Zdrojový kód 3.13: Funkce main

3.6.1.3 Ovládání aplikace

Před samotným spuštění aplikace je nutné připojit skener k počítač. Buď přímo kabelem do počítače, v tom případě je nutné na PC nastavit ručně IP adresu ve stejném rozsahu jako má skener. Pří mé konfiguraci skeneru například 10.0.0.70. Nebo přes router, ale i zde je nutné mít nastavenou IP adresu ve stejném rozsahu jako má skener. IP adresu skeneru je možné nastavit přes program SOPAS Engineering Tool.

Po spuštění aplikace je uživatel vyzván k zadání čísla z nabídky. Zvolí-li uživatel možnost číslo 1, je spuštěno vlákno, proběhne připojení ke skeneru a konfigurace. Následně je nutné vyčkat na uložení nastavení. Během čekání jsou na konzoli tištěny tečky, aby bylo patrné, že aplikace běží. Jakmile se zobrazí „Scaner id Ready“ a „LogOut: Uspesne odhlaseno“ je možné zvolit možnost 2. Tím dojde k uložení aktuálních naměřených dat. Uložení dat je možné provádět opakovaně. Při zvolení možnosti 3 dojde k ukončení aplikace. Příklad, jak může vypadat výpis na konzoli po jednom úspěšném měření můžeme vidět na Obr. 3-5.



Obr. 3-5 Ukázka prostředí aplikace po uložení jednoho měření

3.6.2 Jednovláknový příklad použití ovladače

Tento jednovláknový příklad je obdobou vícevláknové aplikace popsané výše. Rozdíl je pouze v tom, že po spuštění se program ihned připojí ke skeneru a provede jeho konfiguraci. Poté už jen stačí zmáčknout enter pro uložení naměřených hodnot. Po pěti uložených záznamech je program ukončen.

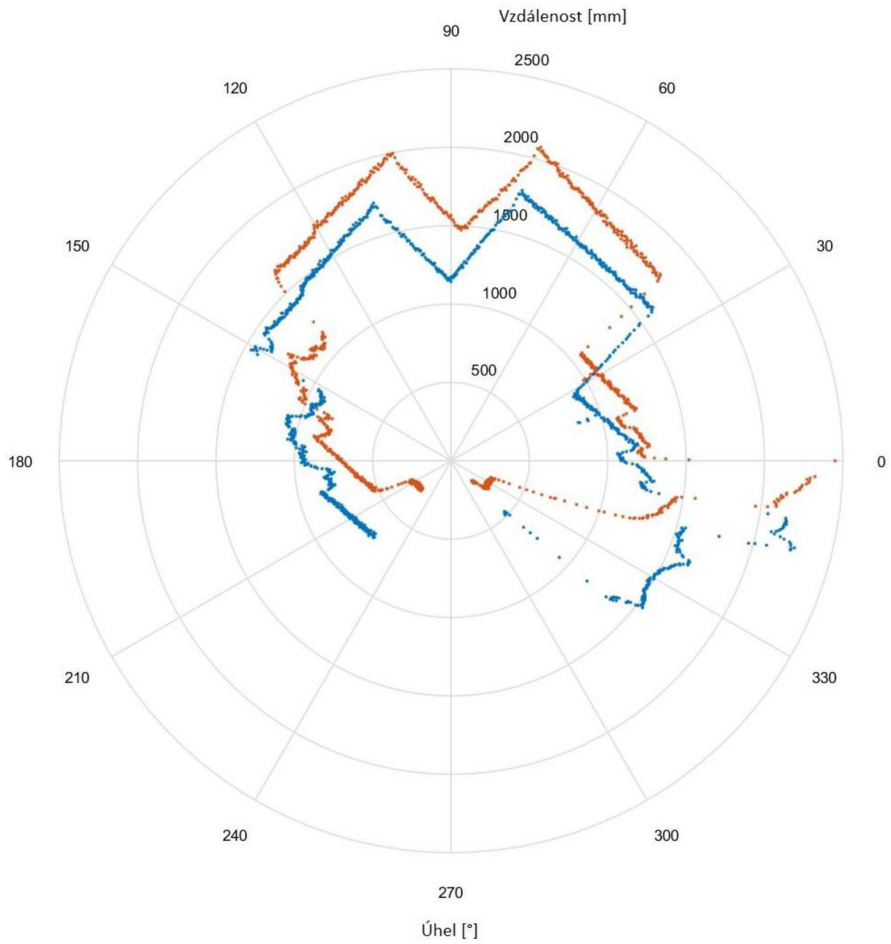
```
int main()
{
	LMS111 Lms111;

	try
	{
		if (Lms111.Connect("10.0.0.74", 2111) != 0)
		{
			getchar();
			return -1;
		}

		{
			Token Tok(&Lms111, 3);
			Tok.SetScancfg(2500, 2500);
			Tok.OutputRange(-450000, 2250000);
			Tok.SaveParamPermanent();
		}

		Lms111.GetScanDataPermanent(true);
		int Poc = 0;

		while (Poc < 5)
		{
			qDebug() << "Stiskni klavesu enter pro zaznam hodnot.";
			getchar();
			QVector<double> pole;
			uint Time;
			int Step;
			long int StAng;
			Lms111.MeasData(pole, Time, StAng, Step);
			int Chyba = Lms111.SaveData(pole);
			if (Chyba == 0)
				qDebug() << "Namerena data byla uspesne ulozena do
						souboru";
			Poc++;
		}

		Lms111.GetScanDataPermanent(false);
	}
	catch (LMS111::ErrorCode& Er)
	{
		qDebug() << Er.what();
		Er.GetError();
		return -2;
	}

	return 0;
}
```

Zdrojový kód 3.14: Jednovláknový příklad

3.6.3 Prezentace reálných naměřených dat

Pomocí ukázkové aplikace jsem naskenoval roh místnosti se zásuvkami od stolu. Skenovaný prostor je vidět na Obr. 3-6. Provedl jsem dvojici měření, přičemž mezi měřeními byla změněna poloha skeneru. Výsledný graf je na Obr. 3-7. V případě modrých bodů je roh zásuvek vzdálen přibližně 1143 mm, v druhém případě 1472 mm. Jedna tečka na grafu představuje jeden naměřený bod.

Obr. 3-6 Fotografie skenovaného prostoru



Obr. 3-7 Graf naměřených hodnot

4. ZÁVĚR

Cílem této práce byla analýza funkcí laserového skeneru SICK LMS111-10100, realizace třídy pro plnohodnotné ovládání skeneru z minipočítače Raspberry Pi 2 a prezentace správné funkčnosti ovladače pomocí jednoduché demo aplikace. Dále vypracování podrobné dokumentace pro snadné použití ovladače.

Úvodní část práce je věnována teoretickému rozboru. Teoretický rozbor obsahuje základní informace o laserovém skeneru SICK LMS111. Dále obsahuje popis principu měření, popis vlivu povrchu měřeného objektu na měření a informaci o komunikačních možnostech skeneru. V teoretickém rozboru jsou též uvedeny základní parametry minipočítače Raspberry Pi 2 spolu s rozborem operačního systému pro Raspberry Pi 2.

Prvním úkolem bylo vybrat programovací jazyk a vývojové prostředí. Standartní programovací jazyk C++ jsem rozšířil použitím Qt knihovny. Výhodou Qt knihovny je mimo jiné i možnost programovat multiplatformní a vícevláknocé aplikace. Jako hlavní vývojové prostředí pro psaní knihovny jsem požíval Microsoft Visual Studio, pro testování knihovny na Raspberry Pi 2 jsem používal NetBeans IDE.

Stěžejní částí této práce byl návrh a realizace třídy pro ovládání laserového skeneru. Vytvořil jsem nadřazenou abstraktní třídu Skener, která slouží jako předloha pro třídu LMS111 ovládající skener. Výhoda abstraktní třídy je, že pokud je použita jako předloha pro třídu jiného skeneru, můžu k oběma skenerům přistupovat stejně i když fungují odlišně.

Úspěšně se mi podařilo realizovat multiplatformní knihovnu pro ovládání laserového skeneru SICK LMS111. Knihovna je zdokumentována pomocí programu Doxygen a kompletní dokumentace je uložena na přiloženém DVD. Při realizaci knihovny bylo využito prostředků Git / GitLab, repozitář je možné nalézt na tomto odkaze: https://student.robotika.ceitec.vutbr.cz/DPBP/DriverLMS111. Kompletní repozpitář je uložen i na přiloženém DVD.

Funkčnost knihovny mám otestovanou na operačních systémech Windows, Ubuntu a Raspbian běžícím na Raspberry Pi 2. Funkčnost jsem ověřil pomocí dvou demo aplikací. První demo aplikace pracuje se dvěma vlákny, zatímco druhá pouze s jedním. Na jednovláknové aplikaci je demonstrováno, že použití knihovny je jednoduché. Funkčnost knihovny jsem ověřil i vykreslením polárního grafu z naměřených dat.

V budoucnu plánuji práci rozšířit o tabulku časů jednotlivých funkcí a na základě této tabulky ještě více optimalizovat kód. Dále bych rád tuto knihovnu využil pro implementaci algoritmů na ovládání mobilního robota v prostoru.

Literatura

[1] Operating instructions: LMS1xx: LASER MEASUREMENT SENSORS. Germany, 2017 [cit. 2018-05-07]. Dostupné také z: https://www.sick.com/media/docs/1/31/331/Operating_instructions_LMS1xx_Laser_Measurement_Sensors_en_IM0031331.PDF

[2] VOJÁČEK, Antonín. Sick LMS100 - laserové snímače prostoru. Automatizace.hw.cz [online]. Praha 4: HW server, 2011 [cit. 2018-05-07]. Dostupné z: https://automatizace.hw.cz/sick-lms100-laserove-snimace-prostoru

[3] Technical information: Telegram Listing:Ranging sensors LMS1xx, LMS5xx, TiM5xx, MRS1000, MRS6000, NAV310, LD-OEM15xx, LD-LRS36xx. Germany, 2018 [cit. 2018-05-07]. Dostupné také z: https://www.sick.com/media/docs/7/27/927/Technical_information_Telegram_Listing_Ranging_sensors_LMS1xx_LMS5xx_TiM5xx_MRS1000_MRS6000_NAV310_LD_OEM15xx_LD_LRS36xx_en_IM0045927.PDF

[4] Telegram listing: Telegrams for Configuring and Operating the LMS1xx, LMS5xx, TiM3xx, JEF300, JEF500. Germany, 2012 [cit. 2018-05-07]. Dostupné také z: http://183.91.33.52/gnat-tang-archive.qiniudn.com/lms.pdf

[5] LAHODA, V. Ovladač laserového skeneru pro Raspberry Pi 2. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 23 s. Vedoucí semestrální práce Ing. Aleš Jelínek.

[6] SICK Sensor Intelligence. 2D laserové skenery LMS1xx / LMS11x / Outdoor [online]. 2015 [cit. 2015-12-28]. Dostupné z: https://www.sick.com/cz/cs/reseni-mericich-a-detekcnich-aplikaci/2d-laserove-skenery/lms1xx/lms111-10100/p/p109842

[7] Raspberrypi.org. RASPBERRY PI 2 MODEL B [online]. UK: RASPBERRY PI FOUNDATION [cit. 2018-05-08]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[8] Router Wireless N 300Mbps Gigabit. In: *MONDO PLAST* [online]. Rumunsko: mondo plast srl, c2004-2018 [cit. 2018-05-09]. Dostupné z: http://www.mondoplast.ro/Router-Wireless-N-300Mbps-Gigabit-TP-Link-TL-WR1042ND-pg_ft-4985

[9] KRČMÁŘ, Petr. Počítač jako pouhý terminál k internetu. ROOT.CZ [online]. ČR: Internet Info, 2012 [cit. 2018-05-09]. Dostupné z: https://blog.root.cz/petrkrcmar/pocitac-jako-pouhy-terminal-k-internetu/

[10] Moderní programování objektových aplikací v C++ (ESF). *Mendelova univerzita v Brně* [online]. Brno: Mendelova univerzita v Brně, 2011 [cit. 2018-05-10]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/index.pl?opora=3048

[11] About Qt. *Qt wiki* [online]. MediaWiki.org, 2018 [cit. 2018-05-10]. Dostupné z: http://wiki.qt.io/About_Qt

[12] Welcome to the NetBeans Community. *NetBeans*[online]. Oracle Corporation, c2018 [cit. 2018-05-10]. Dostupné z: https://netbeans.org/about/index.html

[13] QVector Class. *Qt Documentation* [online]. The Qt Company, c2018 [cit. 2018-05-14]. Dostupné z: http://doc.qt.io/qt-5/qvector.html

[14] Rozumíme C++. Praha: Computer Press, 2003. Programování. ISBN 80-722-6656-X.

[15] PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Přeložil Boris SOKOL. Brno: Computer Press, 2013.Bestseller (Computer Press). ISBN 9788025138281.

Seznam příloh

Přílohy uložené na DVD

Příloha 1 - Gitový repozitář se zdrojovými kódy ovladače

Příloha 2 - Doxygenová dokumentace knihovny v pdf

Příloha 3 - Návod k obsluze pro laserový skener SICK LSM111-10100

Příloha 4 - Seznam telegramů pro konfiguraci a provoz laserového skeneru SICK LMS111-10100