FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Robot NAO jako přednášející |
| **Student:** | Bc. Gabriela Turcajová |
| **Vedoucí:** | Ing. Miroslav Skrbek, Ph.D. |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Systémové programování |
| **Katedra:** | Katedra teoretické informatiky |
| **Platnost zadání:** | Do konce zimního semestru 2019/20 |

## Pokyny pro vypracování

Navrhněte a realizujte programové vybavení pro robota NAO, které umožní, aby robot mohl vést přednášku dle předem připravených slidů. Robot pohovoří o každém slidu, bude přepínat na následující slide a také bude umět ukazovat laserovým ukazovátkem na projekční plochu. Při manipulaci s ukazovátkem bude mít robot optickou zpětnou vazbu polohy ukazovátka pro přesnější zaměření objektů na plátně. Nedílnou součástí práce je programové vybavení, které bude integrovat slidy, doprovodnou řeč a informace pro pohyb robota a laserového ukazovátka. Systém by měl být nezávislý na počátečním umístění robota před projekční plochou. Rozsah práce upřesněte po dohodě s vedoucím práce.

## Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. června 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

# Robot NAO jako přednášející

***Bc. Gabriela Turcajová***

Katedra teoretické informatiky
Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

9. ledna 2019

# Poděkování

Ráda bych touto cestou poděkovala vedoucímu práce Ing. Miroslavu Skrbkovi, Ph.D. za pomoc s realizací této práce a za cenné rady.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. ledna 2019 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2019 Gabriela Turcajová. Všechna práva vyhrazena.
*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.*

### Odkaz na tuto práci

Turcajová, Gabriela. *Robot NAO jako přednášející.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

# Abstrakt

Cílem práce je vytvoření programového vybavení pro robota NAO, které umožní robotovi vést přednášku. Obsahem této práce je návrh a realizace aplikace pro zadání jednotlivých textů ke slajdům, které má robot předčítat, a následné spuštění aplikace vedené robotem.

**Klíčová slova** Robot NAO, počítačové vidění, Python, OpenCV, Tkinter.

# Abstract

The aim of the thesis is to create software for the robot NAO, which will allow the robot to lead a lecture. The content of this work is the design and implementation of the application for entering the texts for the slides of presentation which the robot will read, and the subsequent start of the application led by the robot.

**Keywords** Robot NAO, computer vision, Python, OpenCV, Tkinter.

# Obsah

**Úvod** **1**

**1 Cíl práce** **3**

**2 Existující řešení** **5**

**3 Analýza** **7**
3.1 Robot NAO . . . . . 7
3.2 Python knihovny . . . . . 14

**4 Návrh** **21**
4.1 Příprava slajdů pro robota . . . . . 21
4.2 Přednášení robota . . . . . 22
4.3 Návrh třídního rozdělení . . . . . 23

**5 Realizace** **25**
5.1 Grafické uživatelské rozhraní . . . . . 25
5.2 Přednášení robotem . . . . . 31

**6 Testování a diskuze** **49**
6.1 Testování . . . . . 49
6.2 Možné rozšíření aplikace . . . . . 53

**Závěr** **55**

**Literatura** **57**

**A Seznam použitých zkratek** **61**

**B Obsah přiloženého CD** **63**

# Seznam obrázků


3.1 Velikost robota NAO . . . . . 7
3.2 Komponenty robota NAO . . . . . 9
3.3 Klouby hlavy robota NAO . . . . . 9
3.4 Klouby levé ruky robota NAO . . . . . 10
3.5 Klouby levé nohy robota NAO . . . . . 10
3.6 Poloha kamer robota . . . . . 11

4.1 Stavový diagram: NAO přednáší . . . . . 22
4.2 Návrh třídního rozdělení aplikace . . . . . 23

5.1 Obrazovka pro výběr souboru . . . . . 26
5.2 Obrazovka pro editaci prezentace . . . . . 26
5.3 Elipsová oblast pro laserové ukazovátko . . . . . 27
5.4 Posloupnost tlačítek při vytváření oblasti pro laserové ukazovátko 28
5.5 Snímek vyfocený robotem . . . . . 34
5.6 Použití funkce *Canny* pro získání obrysového obrázku . . . . . 34
5.7 Nalezená kontura ohraničující slajd . . . . . 35
5.8 Transformace snímku vyfoceného robotem . . . . . 37
5.9 Porovnání obrázku pomocí ORB deskriptorů . . . . . 38
5.10 Snímek s výchozím nastavením kamery . . . . . 41
5.11 Snímek s nastavenou kamerou pro detekci laserového ukazovátka . 41
5.12 Profil jasu v okolí nejjasnějšího bodu s laserem . . . . . 43
5.13 Profil jasu v okolí nejjasnějšího bodu bez laseru . . . . . 45
5.14 Rozdíl mezi modelem a bodu s jeho okolím bez laseru . . . . . 45
5.15 Zobrazení laserového ukazovátka na původním slajdu . . . . . 46

# Seznam tabulek

3.1 Specifikace kamery . . . . . 11
3.2 Parametry kamery . . . . . 12
3.3 Autonomní schopnosti robota . . . . . 14

# Úvod

Je to už nějakou dobu, co začaly být nepříjemné činnosti vykonávány stroji. Mohou to být činnosti náročné, jako například sestrojování automobilu. Nebo činnosti málo zábavné, což můžeme pozorovat třeba nárůstem samoobslužných pokladen v supermarketech. S příchodem humanoidních robotů můžeme nahrazovat i ty činnosti, u kterých záleží na příjemném vystupování člověka.

Není žádným tajemstvím, že mnoho lidí shledává prezentování před publikem jako značně nepříjemnou záležitost. Také je velmi unavující rok co rok přednášet ty samé informace studentům na školách. To vše by mohlo být vyřešeno aplikací, která jednoho takového humanoidního robota naučí přednášet.

Tato práce se zabývá vytvořením aplikace, díky které bude robot NAO schopný přednášet podle zadaných textů k jednotlivým slajdům prezentace.

V první části této práce je popsána specifikace robota NAO, tedy jeho komponenty a operační systém. Dále se zde seznámíme s některými knihovnami Pythonu, které umožňují tvorbu aplikace. Podrobněji je zde popsána knihovna OpenCV, která je v aplikaci využívána pro zpracování obrázků.

V návrhu řešení jsou vyjmenovány všechny potřebné uživatelské vstupy pro grafické uživatelské rozhraní. Potom se seznámíme se všemi činnostmi, které by měl robot zvládnout, a s pořadím, ve kterém by měly být vykonávány.

V další kapitole je podrobně popsána realizace programu. Zde se podíváme na to, jak vypadá grafické uživatelské rozhraní výsledné aplikace, jeho funkcionality a jaké vstupy musejí být uživatelem zadány. Potom se seznámíme s tím, jakým způsobem robot přednáší. Tedy co přesně musí udělat pro to, aby zkontroloval, zda se dívá na správný slajd, co to obnáší, aby správně namířil laserové ukazovátko na předem dané místo nebo jak vůbec může zjistit, že jím na správné místo ukazuje.

Nakonec se podíváme na testování aplikace a jak lze dílčí části aplikace testovat v případě, že zrovna nemáme robota NAO k dispozici.

Kapitola 1

# Cíl práce

Cílem této práce je programové vybavení pro robota NAO takové, aby robot mohl vést přednášku. To znamená, že bude schopný mluvit o každém slajdu a bude umět slajdy přepínat.

Robot bude také umět ukazovat laserovým ukazovátkem na projekční plochu. Potom si dokáže ověřit, že ukazuje na správné místo.

Veškeré tyto činnosti by měl robot zvládnout, aniž by bylo dopředu známo umístění robota před projekční plochou.

Cílem je také vytvořit jednoduché grafické uživatelské rozhraní, ve kterém uživatel zadá základní nastavení pro přednášku a veškeré texty, které mají být u jednotlivých slajdů řečeny.

Kapitola 2

# Existující řešení

Tato práce navazuje na semestrální práci „NAO učitel“ [1] z předmětu MI-IVS Adama Lehára a Ondřeje Šmída. Výstupem jejich práce je program napsaný v C++, díky kterému robot NAO mluví o každém slajdu, dokáže detekovat pozici promítaného plátna a laserového ukazovátka na obrázku, a pohybovat laserovým ukazovátkem.

Pro přepočet zadané pozice laserového ukazovátka do vyfoceného slajdu používají funkce *getPerspectiveTransform* a *perspectiveTransform* z knihovny OpenCV (o funkcích z knihovny OpenCV budeme mluvit dále v sekci 3.2.6). Promítané plátno hledají pomocí odhadu středu plátna. Laserové ukazovátko detekují pomocí rozpoznávání rozdílu ve dvou obrázcích. Prvním obrázkem je vyfocený slajd bez namířeného laserového ukazovátka, druhý obsahuje bod laserového ukazovátka, který je potřeba detekovat. Pro pohyb laserového ukazovátka na plátně využívají kloubů robota „WristYaw“ a „ShoulderPitch“ (více o kloubech robota v sekci 3.1.2), kterými iterativní metodou postupně pootáčejí, než se laserové ukazovátko dostane do potřebné pozice. Průběh přednášky je programu předáván jako textový soubor a pomocí příkazů je určeno, zda má robot mluvit, posunovat laserové ukazovátko či si má robot říct o přepnutí slajdu.

Tato práce také využívá funkce z knihovny OpenCV pro získání bodu laserového ukazovátka na vyfoceném slajdu. Nebylo ale možné použít metodu odhadu středu promítaného plátna, neboť dopředu nevíme, na jakém místě bude robot stát. Protože semestrální práce měla potíže s detekcí laserového ukazovátka na světlém pozadí, byla pro tento úkol zvolena jiná metoda. Pro pohyb laserového ukazovátka je také používána iterativní metoda, ale aby robot neukazoval celou paží, byly pro pohyb použity klouby „ElbowYaw“ a „WristYaw“. Pro editaci textů ke slajdům a vymezení pozic pro laserové ukazovátko bylo naimplementováno grafické uživatelské razhraní.

Existuje ještě další řešení, které ovšem z robota NAO nedělá přednášejícího v pravém slova smyslu. Tato aplikace totiž učí malé děti a různými hrami se je snaží zaujmout. Tento modul jménem „Ask NAO!“ [2] vydaný přímo od Soft-

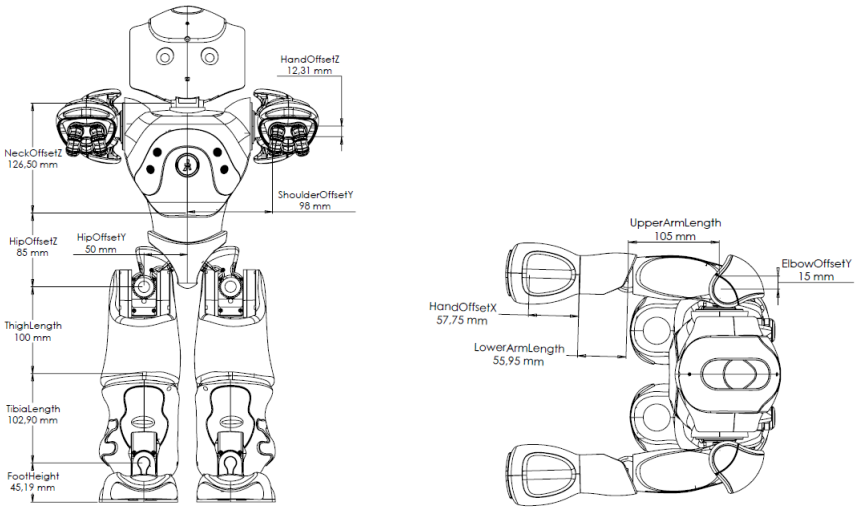
Bank Robotics umožňuje učitelům oživit výuku, naučit děti komunikačním schopnostem a základním znalostem. Robot NAO s tímto modulem pomáhá také dětem s autismem rozvíjet své sociální dovednosti[3].

Kapitola 3

# Analýza

## 3.1 Robot NAO

Robot NAO je prvním humanoidním robotem firmy Aldebaran Robotics. První verze tohoto robota byla vytvořena v roce 2006. Od té doby společnost pracovala také na projektu ROMEO a byla vybrána japonskou společností SoftBank, aby vytvořila prvního emocionálního robota jménem Pepper. V roce 2016 se společnost přejmenovala na SoftBank Robotics a stala se tak součástí SoftBank Group. Aktuálně je robot NAO ve verzi 6, která byla představena v roce 2018. [4]




Obrázek 3.1 – Velikost robota NAO
*SoftBank Robotics: Links*

NAO je schopný rozpoznávat, poslouchat, mluvit a pohybovat se podobně jako člověk. Protože je plně programovatelný, využívá se po celém světě přede-

vším pro vzdělávání a výzkum. Můžeme ho ale potkat i v různých společnostech, kde figuruje především jako zábava pro návštěvníky. [5]

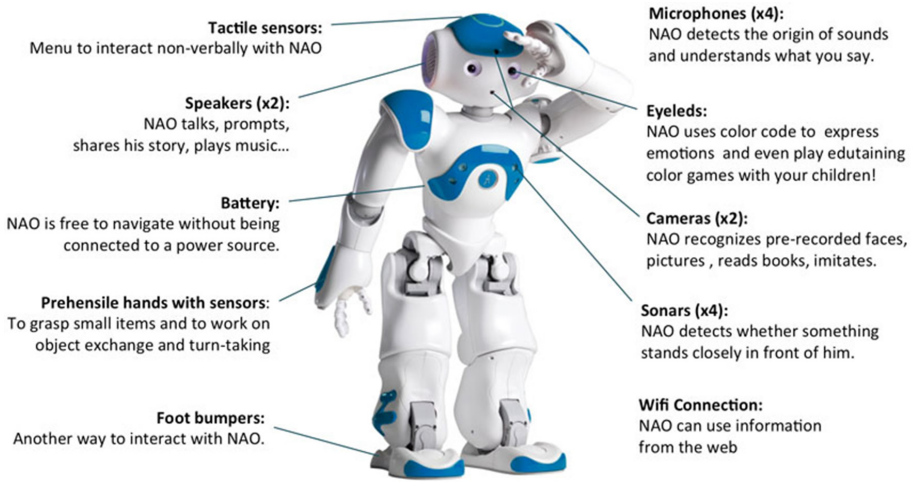
Tato práce byla testována na robotovi NAO typu H25 verze 6. V podsekcích níže se podíváme na technický přehled tohoto robota. Na obrázku 3.1 jsou znázorněny velikosti jednotlivých částí robota. [6]

### 3.1.1 Hlavní komponenty

Aby robot dokázal napodobovat lidské činnosti, potřebuje k tomu určitý hardware. [7] Hlavní komponenty, které Nao využívá, jsou:

- 25 stupňů volnosti, díky kterým je robotovi umožněn volný pohyb,
- celkem 12 dotykových senzorů - tři se nachází shora na hlavě, jeden na hrudníku, tři na každé ruce a jeden na každé noze,
- 4 mikrofony situované na hlavě - dva shora a dva zezadu hlavy,
- 2 reproduktory - každý na jedné straně hlavy,
- 2 kamery - jedna nad a druhá pod očima,
- celkem 50 LED diod - dvanáct shora na hlavě, osm v každém oku, deset okolo každého ucha a po jednom v každém chodidle,
- 2 ultrazvukové snímače, díky kterým je NAO schopný určovat vzdálenost od ostatních objektů,
- baterie, která při aktivitě vydrží až 60 minut a v klidovém režimu až 90 minut.

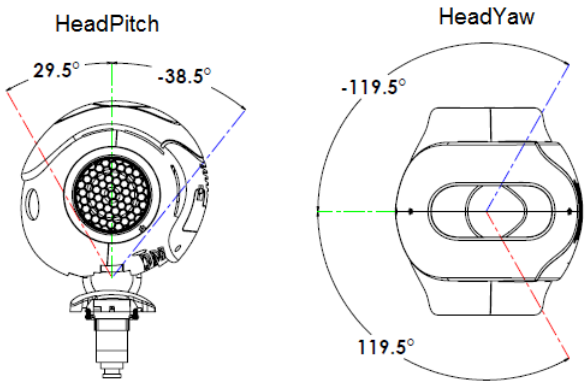
Některé z těchto komponent můžeme vidět na obrázku 3.2 [8].




Obrázek 3.2 – Komponenty robota NAO
*Gigabotics: Features of Nao Robot*

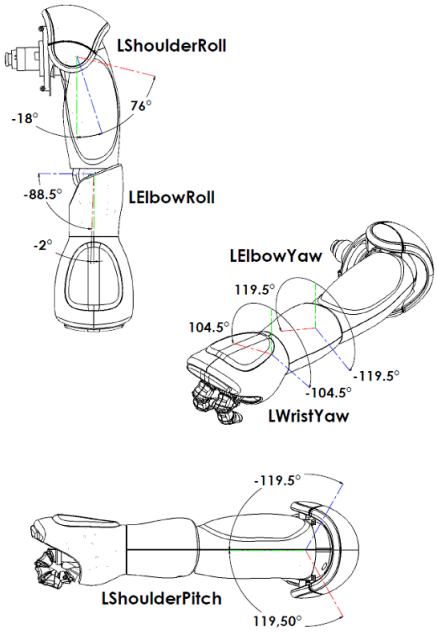
### 3.1.2 Klouby

Klouby [9], neboli spoje mezi určitými částmi robota, umožňují robotovi pohyb. Má jich celkem 25. Dva spojují hlavu s trupem: „HeadPitch“ a „HeadYaw“. Díky kloubu „HeadPitch“ robot může hlavu sklánět a zaklánět. Kloub „HeadYaw“ umožňuje otáčení hlavou doleva a doprava. Pohyblivost těchto kloubů je znázorněno na obrázku 3.3.




Obrázek 3.3 – Klouby hlavy robota NAO
*SoftBank Robotics: Joints*

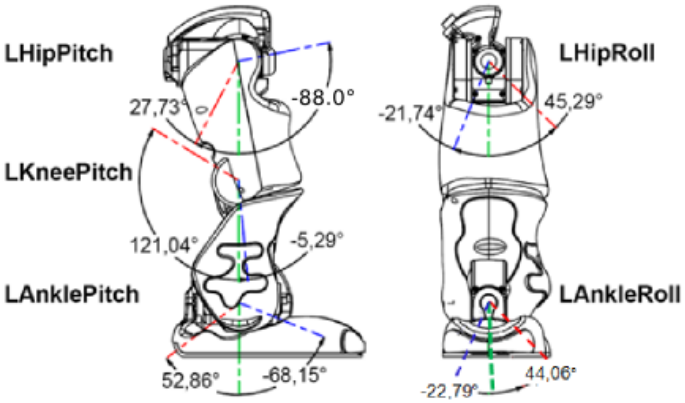
Celkem šest kloubů se nachází na každé paži. Dva klouby umožňují pohybovat celou paží, dva ohýbají paži v lokti, jeden otáčí rukou v zápěstí a jeden svírá a povoluje prsty robota. Směr a velikost pohybu levé ruky můžeme vidět na obrázku 3.4.




Obrázek 3.4 – Klouby levé ruky robota NAO
*SoftBank Robotics: Joints*

Dalšími dvěma klouby jsou pánevní klouby. Fyzicky se ale jedná pouze o jeden motor, takže nemohou být řízeny samostatně. Pokud by mezi těmito klouby nastal konflikt v pohybu, upřednostní se vždy pohyb levého kloubu.

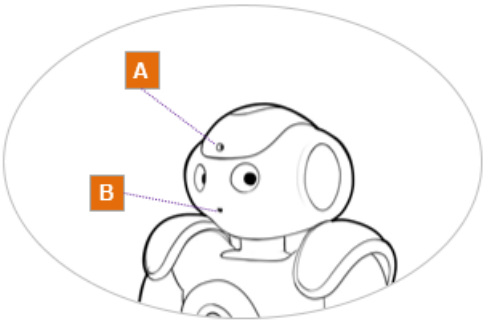
Posledních deset kloubů se nachází v nohách robota. Dva umožňují pohyb v kyčli, jeden v koleni a dva v kotnících. Rozsah a směr pohybu levé nohy robota je vidět na obrázku 3.5.




Obrázek 3.5 – Klouby levé nohy robota NAO
*SoftBank Robotics: Joints*

### 3.1.3 Kamerový systém

Vepředu na hlavě jsou situovány dvě identické kamery OV5640 typu System-on-a-chip (SoC) CMOS image sensor. Kamerový systém [10] s rozlišením až 2560 × 1920 umožňuje robotovi rozpoznávat okolní objekty. Polohu kamer robota můžeme vidět na obrázku 3.6.




Obrázek 3.6 – Poloha kamer robota
*SoftBank Robotics: Video cameras*

V tabulce 3.1 jsou vypsány specifikace kamery OV5640.

| | | |
|---|---|---|
| Kamera | Model | OV5640 |
| | Typ | System-on-a-chip (SoC) CMOS image sensor |
| Zobrazovací pole | Rozlišení | 5 Mp |
| | Velikost snímače | 1/4" |
| | Aktivní pixely | 2592 × 1944 |
| Citlivost | Velikost pixelu | $1.4\mu m \times 1.4\mu m$ |
| | Dynamický rozsah | $68dB$ |
| | Poměr signálu / šumu (max) | maximálně $36dB$ |
| | Citlivost | $550nm$ |
| Výstup | Výstup kamery | 640 × 480 při 30 fps nebo 2560 × 1920 při 1 fps |
| | Formát dat | YUV a RGB |
| | Typ závěrky | Elektronická rolovací závěrka (ERS) |
| Zobrazení | Zorné pole | 67.4°DFOV |
| | Typ zaostření | Automatické zaostření |

Tabulka 3.1 – Specifikace kamery

Protože je robot NAO plně programovatelný, je možné různě nastavovat i

parametry kamery. Mezi nastavitelné hodnoty se řadí například jas, kontrast, saturace nebo expozice. V tabulce 3.2 můžeme vidět všechny nastavitelné parametry s minimálními, maximálními a výchozími hodnotami.

| Parametr | Minimální hodnota | Maximální hodnota | Výchozí hodnota | Poznámka |
|---|---|---|---|---|
| Jas | -255 | 255 | 0 | |
| Kontrast | 0 | 255 | 32 | |
| Saturace | 0 | 255 | 64 | |
| Odstín | -180 | 180 | 0 | |
| Zesílení | 0 | 1023 | 16 | Pouze pro čtení, pokud je zapnutá automatická expozice |
| Horizontální převrácení | 0 | 1 | 0 | |
| Vertikální převrácení | 0 | 1 | 0 | |
| Automatická expozice | 0 | 1 | 1 | 0: vypnutá, 1: zapnutá |
| Automatické vyvážení bílé | 0 | 1 | 1 | 0: vypnutá, 1: zapnutá |
| Expozice | 0 | 65535 | 32 | Pouze pro čtení, pokud je zapnutá automatická expozice |
| Ostrost | 0 | 9 | 4 | |
| Kompenzace protisvětla | 0 | 1 | 0 | 0: vypnutá, 1: zapnutá |
| Průměrný jas | 0 | 255 | Nedostupné | Pouze pro čtení |
| Automatické ostření | 0 | 1 | 0 | 0: vypnutá, 1: zapnutá |
| Manuální ostření | 0 | 250 | 0 | Dostupný, pouze pokud je automatické ostření vypnuté |

Tabulka 3.2 – Parametry kamery

### 3.1.4 Software

Hlavní systém, který v robotovi běží a ovládá ho, se nazývá NAOqi [11]. K tomu, abychom mohli robota naprogramovat, využíváme NAOqi framework. Tento framework umožňuje homogenní komunikaci mezi různými moduly (pohyb, zvuk, video), homogenní programování a homogenní sdílení informací.

NAOqi framework je multiplatformní, umožňuje vyvíjet na operačních systémech Windows, Linux i MacOS. Pro naprogramování NAO robota se vývojář může rozhodovat mezi dvěma programovacími jazyky, a to C++ nebo Python. V obou případech jsou programovací metody úplně stejné. V případě Pythonu je možné jednoduše spustit kód jak v počítači tak přímo na robotovi. Protože je C++ kompilovaný jazyk, před spuštěním v robotovi je potřeba kód nejdříve zkompilovat pro cílový operační systém.

#### 3.1.4.1 NAOqi API

Pro naprogramování robota je ve frameworku NAOqi připraveno mnoho API (modulů), které je možné využít. Tyto moduly umožňují programátorovi jednoduché ovládání robota, jako třeba jeho pohyb, mluvení, nahrávání videa nebo získávání dat ze senzorů. Zároveň je ale možné naprogramovat i modul vlastní.

Hlavní modul pro ovládání pohybu robota je modul „ALMotion“ [12]. Tento modul obsahuje čtyři hlavní skupiny metod, které řídí ztuhlost kloubů, pozici kloubů, chůzi a efektory, neboli předdefinované 3D body v robotovi. Příkladem je metoda „angleInterpolation“, která jeden nebo více zadaných kloubů robota dostane do požadované pozice.

V modulu jsou také naimplementovány některé reflexy robota, které jsou ve výchozím nastavení zapnuté. Díky těmto reflexům například robot pohyb upravuje tak, aby předešel k nechtěnné kolizi buď sám se sebou nebo s nějakým externím objektem, nebo spustí konkrétní úlohu při detekci pádu nebo při detekci kritické chyby na zařízení.

Další modul, který ovládá pohyby robota, je modul „ALTracker“ [13]. Tento modul umožňuje sledovat různé objekty, jako je například červený míček, obličej a podobně. Je možné definovat, jestli robot bude pro sledování daného objektu pohybovat pouze hlavou, nebo bude natáčet celé tělo, či za ním bude chodit.

Dalším modulem pro pohyb může být modul „ALRobotPosture“ [14]. Tento modul umožňuje daným nastavením kloubů dostat robota do předem definované polohy. Polohou může být například sed, leh na břicho nebo na záda, nebo stání.

Moduly řídící kameru robota jsou například „ALVideoDevice“ [15] nebo „ALVideoRecorder“ [16]. První z nich získá obrázek kamerou robota a připraví ho k dalšímu použití. Druhý modul kamerou nahraje video a uloží ho na disk.

Modul „ALTextToSpeech“ [17] umožňuje zadaný text přečíst robotem. Nejprve se zvolí jazyk, ve kterém bude robot mluvit. NAO robot podporuje přes 20 různých jazyků, mezi nimi i češtinu. Potom modul posílá říkazy do text-to-speech stroje a umožňuje také přizpůsobení hlasu. Výsledek je poslán do robotových reproduktorů.

Ve frameworku NAOqi je několik modulů, které umožňují robotovi interagovat s okolím. [18] Veškeré dostupné autonomní schopnosti jsou vypsány v tabulce 3.3. Pokud je některá z těchto schopností zapnutá, robot ji vykonává automaticky po celou dobu vedle aktivit definovaných programátorem. API „ALAutonomousLife“ [19] obsahuje metody pro získávání dat o těchto autonomních schopnostech, tedy například v jakém jsou jednotlivé schopnosti stavu nebo historii schopností, které byly zapnuté v minulosti.

| **Autonomní schopnost** | **Popis** | **API** |
|---|---|---|
| Autonomní mrkání | Umožňuje robotovi mrkat pomocí blikání jeho LED diod v očích. Tato schopnost se spouští, když například zahlédne člověka. | ALAutonomousBlinking |
| Pohyb na pozadí | Určuje, jaké pohyby robot provádí autonomně, když se jeho končetiny nepohybují. | ALBackgroundMovement |
| Základní povědomí | Pomáhá robotovi reagovat na okolí tak, že vytváří a udržuje oční kontakt s lidmi. | ALBasicAwareness |
| Pohyby při poslechu | Robot se drobně pohybuje způsobem, aby ukázal, že poslouchá hovořící osobu. | ALListeningMovement |
| Pohyby při mluvení | Robot se autonomně pohybuje během svého mluveného projevu. | ALSpeakingMovement |

Tabulka 3.3 – Autonomní schopnosti robota

## 3.2 Python knihovny

Protože se práce zabývá především počítačovým viděním, ukázal se být vhodný programovací jazyk Python, neboť umožňuje jednoduché zpracování obrázků. Níže je seznam některých balíčků Pythonu, které byly v implementaci použity.

### 3.2.1 Tkinter

Tkinter [20] je knihovna pro vytváření grafického uživatelského rozhraní v Pythonu. Tato knihovna je dostupná na většině Unixových platformách, na Windows i na Mac OS. Knihovna nabízí různé objekty od textových polí přes tlačítka po rozbalovací seznamy. Tyto objekty lze slučovat a vhodně umisťovat v okně například pomocí objektu „Frame“, který se dá zarovnávat buď vertikálně nebo horizontálně. Dále je možné využít dialogová okna s chybovými hláškami, s upozorněními nebo s informačními zprávami. Tato dialogová okna přebírají vzhled oken daného operačního systému.

### 3.2.2 Pillow

Balíček Pillow [21], neboli Python Imaging Library (PIL), umožňuje zpracovávat obrázky. Tato knihovna poskytuje rozsáhlou podporu formátu souborů, efektivní interní reprezentaci a poměrně výkonné možnosti zpracování obrazu.

### 3.2.3 pdf2image

Pdf2image [22] je malá knihovna, která převádí soubor pdf do seznamu obrázků typu PIL.

### 3.2.4 NumPy

NumPy [23] je základním balíčkem pro vědecké výpočty v Pythonu. Tato knihovna poskytuje multidimenzionální pole, různé odvozené objekty (jako jsou maskovaná pole a matice) a rychlé úpravy polí, jako jsou matematické a logické operace, manipulace s tvarem, řazení, výběr, diskrétní Fourierovy transformace, základní lineární algebra, základní statistické operace, náhodná simulace a mnoho dalšího.

### 3.2.5 PyAutoGUI

PyAutoGUI [24] je knihovna, která poskytuje automatizaci grafického uživatelského rozhraní pro víceúrovňový Python. PyAutoGUI dokáže simulovat pohyb myši, kliknutí myší, přetahování myší, stisknutí kláves, podržení tlačítek a stisknutí klávesových zkratek.

### 3.2.6 OpenCV

OpenCV [25] (Open Source Computer Vision Library) je open-source knihovna s BSD (Berkeley Source Distribution) licencí, která se zaměřuje na software počítačového vidění a strojového učení. Tato knihovna má rozhraní pro C++, Python a Javu a podporuje Windows, Linux, Mac OS, iOS a Android. Knihovna OpenCV byla navržena pro výpočetní efektivitu a se silným zaměřením

na aplikace v reálném čase. Díky tomu, že je napsaná v optimalizovaném C / C++, může být využita při vícejádrovém zpracování.

Knihovna obsahuje více než 2500 optimalizovaných algoritmů a zahrnuje komplexní sadu klasických a nejmodernějších počítačových vizí a algoritmů pro strojové učení. Pomocí této knihovny je například možné detekovat a rozpoznávat lidské obličeje, identifikovat objekty, klasifikovat lidské akce ve videu, sledovat pohybující se objekty ve videu, najít podobné obrázky z databáze obrázků, odstranit červené oči ze snímků pořízených pomocí blesku a mnoho dalšího.

OpenCV má více než 47 tisíc uživatelů a odhadovaný počet stažení přesahuje 14 milionů. Knihovna je široce využívána ve firmách, výzkumných skupinách a vládních orgánech.

Některé z algoritmů, které jsou v knihovně naiplementovány, jsou popsány níže.

#### 3.2.6.1 Geometrické transformace obrazu

Knihovna OpenCV obsahuje několik funkcí pro geometrickou transformaci 2D obrázků [26]. Tyto funkce nemění obsah obrázku, pouze deformují mřížku pixelů a mapují tuto mřížku do výsledného obrázku. Mapování je prováděno v opačném pořadí, tedy z cílového obrázku do zdrojového obrázku, tak, že pro každý pixel $(x, y)$ cílového obrázku funkce vypočítá souřadnice odpovídajícího pixelu zdrojového obrázku a zkopíruje hodnotu pixelu:

$$dst(x, y) = src(f_x(x, y), f_y(x, y)). \tag{3.1}$$

Implementace geometrické transformace obrázku musí řešit dva hlavní problémy, které vznikají z předchozího vzorečku - extrapolace neexistujícího pixelu a interpolace hodnot pixelu.

Extrapolaci neexistujícího pixelu musíme použít, pokud pro nějaké $(x, y)$ spadá jedna z hodnot $f_x(x, y)$ nebo $f_y(x, y)$, nebo obě hodnoty, vně obrázku. Existuje několik extrapolačních metod, například metoda, která pro danou oblast použije vedlejší pixely, které už do obrázku spadají, nebo metoda, která předpokládá, že všechny neexistující pixely jsou nulové.

Interpolaci hodnot pixelu musíme použít proto, že $f_x(x, y)$ a $f_y(x, y)$ jsou většinou čísla s plovoucí desetinnou čárkou. To znamená, že musíme nalézt hodnotu pixelu na neceločíselných souřadnicích. Nejjednodušeji můžeme použít hodnotu pixelu, který se nachází na souřadnicích zaokrouhlených k nejbližšímu celému číslu. Tato interpolace se nazývá „nejbližší soused“. Lepších výsledků ale můžeme dosáhnout pomocí sofistikovanějších interpolačních metod, kde proložíme sousední pixely počítaného pixelu polynomiální funkcí a z této funkce potom získáme hodnotu na souřadnicích $(f_x(x, y), f_y(x, y))$, kterou pak považujeme za interpolovanou hodnotu pixelu. Mezi sofistikovanější interpolační metody patří například bilineární interpolace nebo bikubická interpolace.

Interpolace hodnot je například používaná při změně velikosti obrázku, v OpenCV se tato funkce jmenuje *resize()*. Tato funkce zmenší nebo zvětší zadaný obrázek na požadovanou velikost. Programátor si může vybrat z pěti interpolačních metod, přičemž bilineární interpolace je nastavena jako výchozí.

Další užitečnou funkcí je funkce *getPerspectiveTransform()*, která na základě čtyř bodů ze zdrojového obrázku a čtyř bodů cílového obrázku vytvoří $3 \times 3$ transformační matici takovou, že:

$$\begin{pmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{pmatrix} = mapovaci_matice \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}, \tag{3.2}$$

kde $dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$.

Funkce *warpPerspective()* potom tuto transformační matici aplikuje na zdrojový obrázek. Výstupem je pak transformovaný obrázek, jehož hodnoty se získají pomocí vzorce

$$dst(x, y) = src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right), \tag{3.3}$$

kde $M$ je transformační matice, pokud je příznak *WARP_INVERSE_MAP* nastavený. To znamená, že počítáme inverzní transformaci z cílového obrázku do zdrojového ($dst \rightarrow src$). V opačném případě je nejprve z transformační matice vypočítána matice inverzní, jejíž hodnoty jsou potom aplikovány ve vzorci výše.

#### 3.2.6.2 Další funkce pro zpracování obrazu

Funkce *findContours()* [27] v obrázku vyhledává obrysy. Parametry funkce jsou:

- *zdrojový obrázek* - obrázek musí být 8 bitový a jednokanálový. V průběhu funkce je zdrojový obrázek pozměněn, pokud by se tedy měl později použít, je potřeba do funkce vložit jeho kopii.
- *mód* - mód určuje, které obrysy a v jakém pořadí budou výstupem funkce. Může se jednat pouze o extrémní vnější obrysy nebo všechny obrysy různě seřazené podle topologie obrázku
- *metoda* - uživatel může vybrat aproximační metodu, která určuje, jak budou obrysy zakódovány. Pokud nemá být žádná aproximační metoda použita (*CHAIN_APPROX_NONE*), pro každý obrys jsou výstupem všechny jeho body. Můžeme ale použít metodu s jednoduchou aproximací (*CHAIN_APPROX_SIMPLE*), kde jsou horizontální, vertikální a diagonální segmenty komprimovány. Díky tomu jsou pro každý obrys výstupem pouze jeho koncové body.

Výstupem této funkce jsou pak jednotlivé obrysy v obrázku a hierarchie těchto obrysů, která obsahuje informaci o topologii obrázku.

Knihovna OpenCV obshuje několik funkcí pro filtraci obrazu [28]. Funkce mohou být lineární i nelineární. Pro každý pixel $(x, y)$ je výsledek vypočítán z jeho sousedních pixelů. V případě lineární funkce se jedná o váženou sumu okolních hodnot pixelů, v případě morfologických operací je to například maximum nebo minimum z hodnot. Výsledek je uložen na stejných souřadnicích $(x, y)$ v cílovém obrázku, tedy zdrojový i cílový obrázek mají stejný rozměr.

Protože získáváme hodnoty z okolních pixelů, i zde, jako u geometrické transformace obrazu, musí být použita extrapolace neexistujícího pixelu. Tato situace nastává v případě, že vypočítáváme hodnotu například pro levý horní pixel. Sousední pixely nahoře a vlevo od tohoto pixelu jsou mimo zdrojový obrázek. Díky různým metodám extrapolace neexistujícího pixelu můžeme tento problém vyřešit.

Příkladem funkce pro filtraci obrazu může být funkce *GaussianBlur()*, která na zdrojový obrázek aplikuje rozmazání pomocí Gaussova filtru.

Pro nalezení hran je v knihovně OpenCV implementovaná funkce *Canny()* [29]. Tato funkce využívá algoritmu, který navrhl australan John Canny. Ve svém textu *A Computational Approach to Edge Detection* [30] zhodnotil, že dosavadní metody vyhledávání hran v obrázku se řídí dvěma kritérii, a to:

1. detekce by měla být dobrá; měla by být malá pravděpodobnost neoznačení existující hrany a malá pravděpodobnost označení neexistující hrany,

2. hrany by měly být správně lokalizovány; body, které jsou označeny jako body hrany, by měly být co možná nejblíže ke středu hrany.

John Canny přišel s dalším kritériem, který by měl algoritmus pro vyhledávání hran v obrázku dodržovat, a to takové, že by každá hrana měla být nalezena pouze jednou.

Pro změnu barevného prostoru obrázku může být použita funkce *cvtColor()* [31]. Možné konverze jsou:

- RGB ↔ GRAY
- RGB ↔ CIE XYZ.Rec 709
- RGB ↔ YCrCb
- RGB ↔ HSV
- RGB ↔ HSL
- RGB ↔ CIE L*a*b*
- RGB ↔ CIE L*u*v*

- Bayer → RGB

Při použití funkce je nutné explicitně specifikovat pořadí kanálů, tedy jestli se jedná o RGB nebo BGR. Například při převodu do odstínu šedi se pak jako kód barevného prostoru použije buď *CV_BGR2GRAY* nebo *CV_RGB2GRAY*.

Kapitola 4

# Návrh

Výsledná aplikace by měla uživateli umožnit připravit pro robota slajdy, tedy ke každému slajdu zadat, co má povídat a případně kdy a kam se má ukazovat laserovým ukazovátkem. Další částí aplikace je samotné spuštění robota a jeho přednášení. V této kapitole je popsán návrh řešení výsledné aplikace.

## 4.1 Příprava slajdů pro robota

Aby mohl uživatel zadávat jednotlivé texty ke slajdům, je potřeba vytvořit jednoduché uživatelské rozhraní s touto funkcionalitou. V tomto rozhraní bude možné prohlížet slajdy prezentace, ke každému z nich bude možné napsat text, který robot po spuštění přečte, a zároveň bude možné definovat oblast slajdu, kam má robot v určitou chvíli posvítit laserovým ukazovátkem. Dále bude možné určit, jak dlouho se na dané místo bude ukazovat, případně jestli bude ukazování doprovázeno dalším textem.

Kromě textů ke slajdům je potřeba od uživatele ještě určit:

- ip adresu robota
- číslo portu
- v jaké ruce bude robot držet laserové ukazovátko, neboli na jaké straně od robota budou promítané slajdy
- v jakém jazyce bude přednáška přednášena (čeština nebo angličtina)
- barvu laserového ukazovátka (červená, zelená, fialová)

Aplikace by měla umožnit ukládat rozpracované texty a nastavení. Z toho plyne, že veškeré uživatelské vstupy budou zaznamenávány do souboru, který potom aplikace bude muset umět znovu načíst. Jakmile bude uživatel s přípravou hotov, měl by mít možnost z aplikace spustit prezentování robotem.

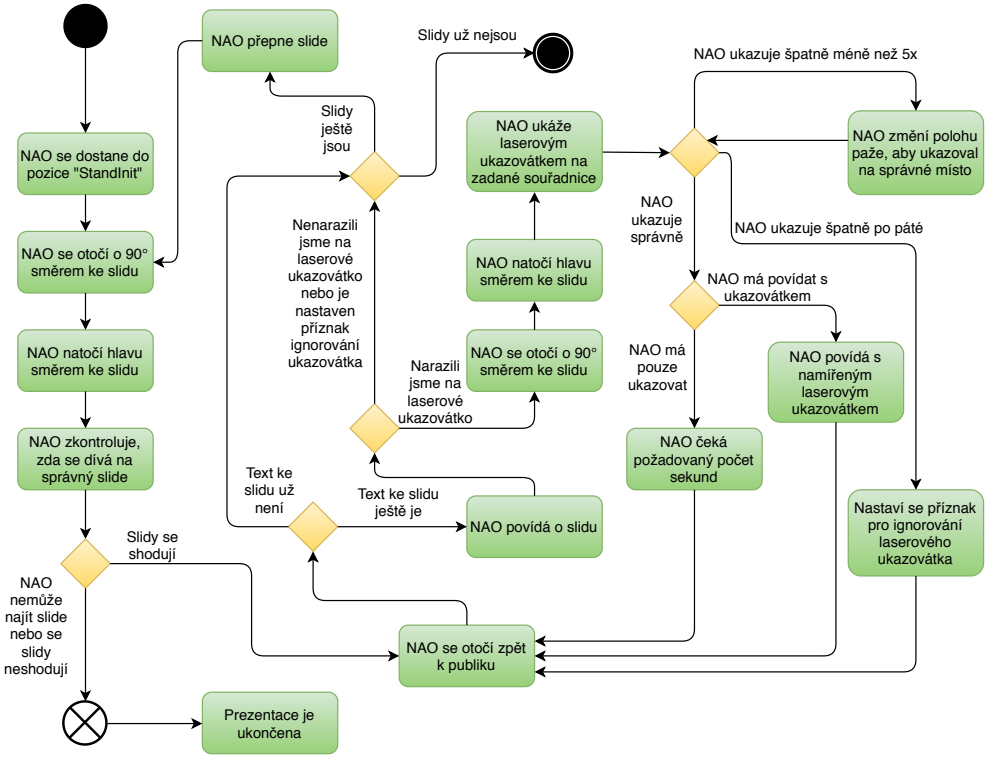
## 4.2 Přednášení robota

Jakmile je přednáška spuštěna, robot by se měl ze své stávající polohy postavit a začít povídat o jednotlivých slajdech.

Nejprve musí zkontrolovat, zda se dívá na správný slajd. Pokud zjistí, že se slajd neshoduje s tím, o kterém by měl povídat, prezentace se rovnou ukončí. Může nastat i situace, že NAO nebude schopný prezentovaný slajd najít, například kvůli špatným světelným podmínkám. I v tomto případě bude prezentace ukončena.

Když se NAO ujistí, že vidí správný slajd, začne o něm povídat. Pokud narazí na příznak, že by měl ukázat na slajd laserovým ukazovátkem, pokusí se zamířit do předem dané oblasti. Když se mu to hned nepodaří, vypočítá se vzdálenost od oblasti a robot posune paži s laserovým ukazovátkem na nové místo. NAO má pět pokusů na to, aby se mu to podařilo. Pokud to do té doby nezvládne, bude v prezentaci pokračovat dál, ale přestane ukazovat laserovým ukazovátkem.

Stejným způsobem pokračuje pro všechny slajdy. Jakmile dopovídá o posledním, prezentace se vypne. Jednotlivé kroky jsou vyobrazeny na stavovém diagramu 4.1.



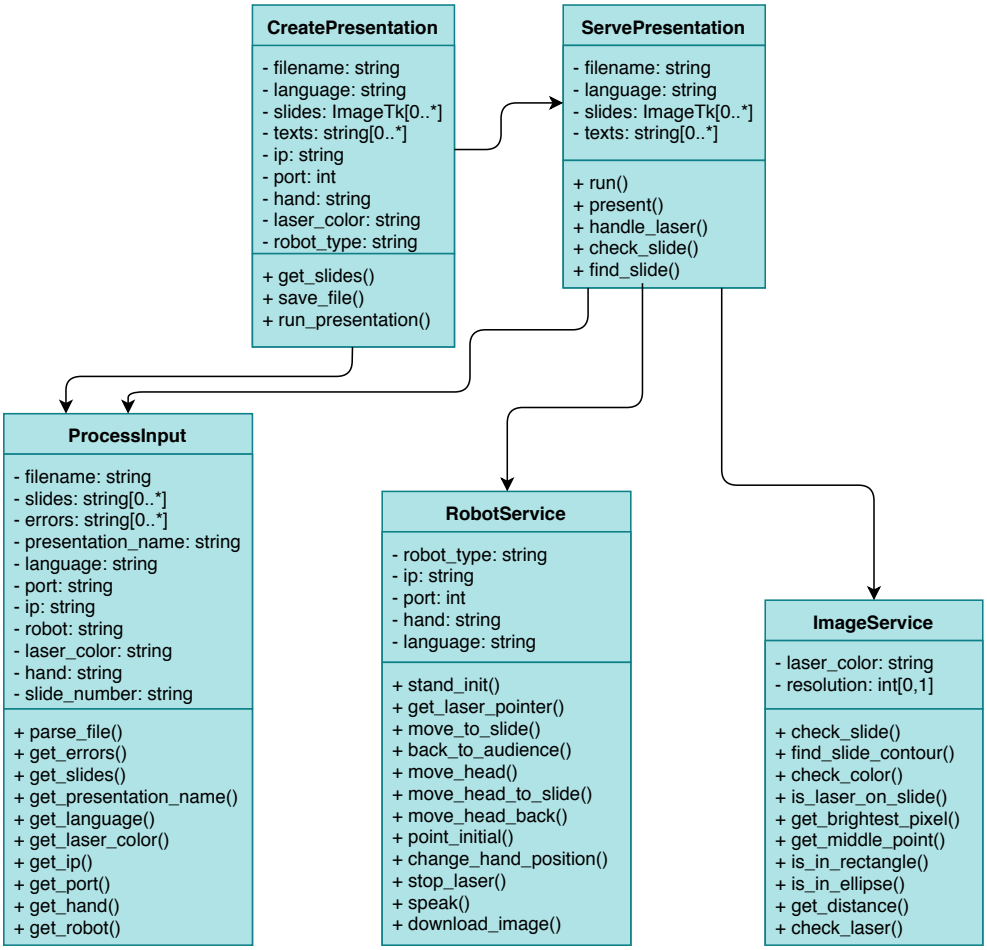

Obrázek 4.1 – Stavový diagram: NAO přednáší

## 4.3 Návrh třídního rozdělení

Aby výsledná aplikace pokryla veškeré požadavky, bude rozdělena do několika tříd.

Třída *CreatePresentation* představuje především grafické uživatelské rozhraní, stará se o ukládání vstupů zadaných uživatelem a následné spuštění prezentace, tak aby byla dostupná všechna potřebná nastavení.

Třída *ServePresentation* spravuje průběh prezentování robotem. Přepíná a kontroluje slajdy, určuje robotovi, jaké úkony by měl právě dělat a rozhoduje, jaká činnost bude následovat. Tato třída má v podstatě kontrolu nad průběhem celé přednášky přesně tak, jak je znázorněno na stavovém diagramu 4.1.




Obrázek 4.2 – Návrh třídního rozdělení aplikace

Třída *ProcessInput* získává data z načteného souboru a kontroluje jejich správnost. Tato funkcionalita se využívá v případě, že byl aplikací načten

rozpracovaný soubor pro prezentaci. Další využití tato třída má při samotné prezentaci robotem. V případě, že má robot ukázat laserovým ukazovátkem, zjišťuje, do jaké oblasti na slajdu se má trefit.

Třída *RobotService* obstarává veškeré ovládání robota. Mezi to patří například vyfocení obrázku kamerou robota, pohybování, otáčení hlavou nebo pohyb ruky s laserovým ukazovátkem.

Třída *ImageService* obsahuje veškeré operace s obrázky. Zde se tedy kontroluje, jestli vyfocený slajd odpovídá slajdu v aplikaci nebo zda je laserové ukazovátko správně nasměrováno a pokud ne, kam je potřeba ho nasměrovat.

Návrh třídního rozdělení aplikace se seznamem proměnných a metod je znázorněn na obrázku 4.2.

Kapitola 5

# Realizace

V této kapitole je podrobný popis jednotlivých částí aplikace. Nejprve se podíváme na grafické uživatelské rozhraní, tedy na vizualizaci aplikace, její ovládání, jakým způsobem se ukládá rozpracovaná prezentace a jak je možné připravenou prezentaci spustit. Potom se zaměříme na samotné prezentování robotem, co robot během přednášení dělá, jak rozpoznává jednotlivé slajdy a jakým způsobem ukazuje laserovým ukazovátkem do vymezených oblastí.

## 5.1 Grafické uživatelské rozhraní

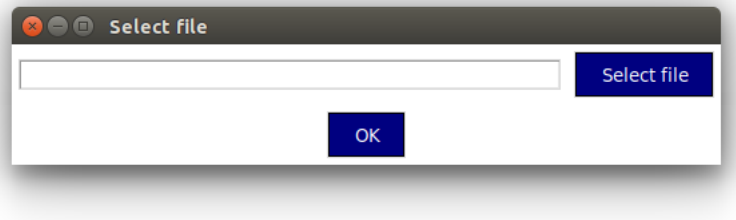
Grafické uživatelské rozhraní sestává ze dvou hlavních obrazovek. První z nich je jednoduché okno s výběrem souboru. Druhé z nich je okno, které umožňuje editovat texty ke slajdům a měnit další nastavení.

### 5.1.1 Okno s výběrem souboru

Po spuštění aplikace se zobrazí jednoduché okno pro výběr souboru. Zde uživatel vybírá .pdf soubor s prezentací, kterou bude robot později přednášet. Druhým podporovaným typem souboru je soubor s příponou .nao, který obsahuje rozpracovaný text ke slajdům, a byl dříve vytvořen touto aplikací. Struktura tohoto souboru je popsána v sekci 5.1.3.

Toto okno sestává z textového pole (typ *Entry*) a dvou tlačítek (typ *Button*). Jeho grafické znázornění můžeme vidět na obrázku 5.1. Po stisknutí tlačítka *Select file* se objeví dialog s možností listování adresáře v počítači. Tento dialog je tvořen pomocí funkce *askopenfilename()* z modulu *tkFileDialog* knihovny *Tkinter*.

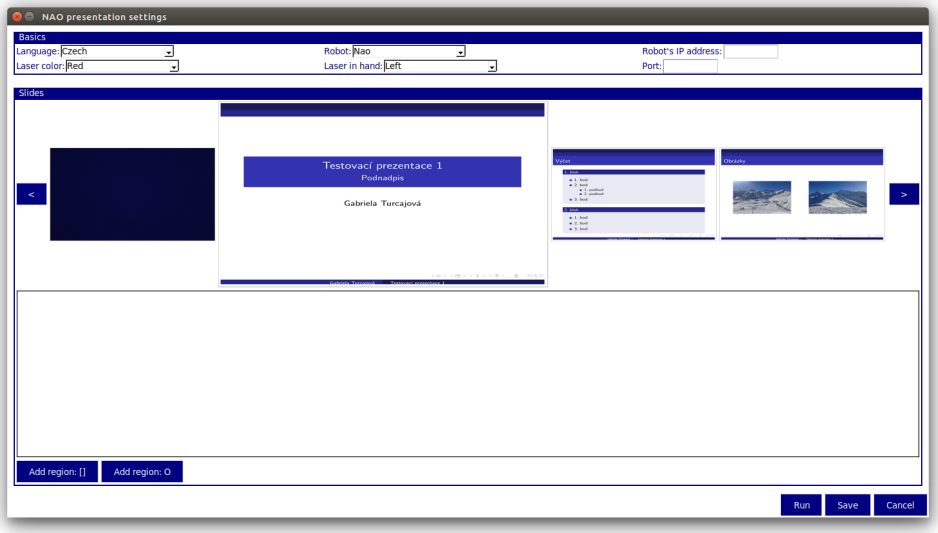
Pokud žádný soubor nebyl vybrán nebo je nepodporovaný a uživatel stiskne tlačítko *OK*, objeví se chybová hláška: „Jsou podporovány pouze .nao a .pdf soubory“. Pokud je vybrán správný soubor a je stisknuto tlačítko *OK*, objeví se nové dialogové okno s možností editace jednotlivých slajdů.




Obrázek 5.1 – Obrazovka pro výběr souboru

### 5.1.2 Editační okno

Editační okno sestává ze tří hlavních částí - základní nastavení, editace textů pro jednotlivé slajdy a tlačítka pro ukončení aplikace, uložení do souboru a spuštění prezentace. Náhled tohoto okna můžeme vidět na obrázku 5.2.




Obrázek 5.2 – Obrazovka pro editaci prezentace

V základním nastavení uživatel definuje důležité vlastnosti pro spuštění prezentace:

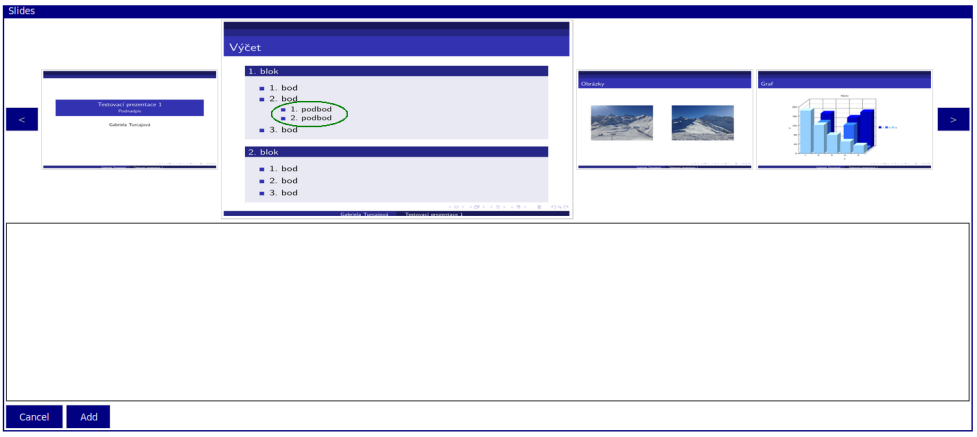
- jazyk - je možné zvolit mezi češtinou a angličtinou
- barva laserového ukazovátka - na výběr je červené, zelené nebo fialové
- robot - zda bude prezentovat robot NAO nebo robot Pepper
- v jaké ruce bude robot držet laserové ukazovátko

- IP adresa robota
- port připojení

IP adresa robota a port jsou povinné hodnoty a pokud nejsou políčka vyplněná, prezentace se nespustí. Nicméně je možné rozpracovanou editaci uložit do souboru bez těchto dvou hodnot, kontrola probíhá až před samotným spuštěním prezentace.

V druhé části editačního okna je možné vyplnit text ke každému slajdu prezentace. Textové pole se vždy vztahuje k tomu slajdu, který je větší než ostatní. Slajdy lze procházet pomocí šipek, které se nachází nalevo a napravo od slajdů.

Kromě textů lze určit i oblast, kam má robot ukázat laserovým ukazovátkem. Pomocí tlačítka *Add region: []* lze ohraničit oblast obdélníkem, pomocí tlačítka *Add region: ()* bude oblast ohraničená elipsou. Po stisknutí jednoho z těchto dvou tlačítek může uživatel na plochu zvětšeného slajdu nakreslit předem zvolený tvar. Naopak již není možné procházet slajdy, dokud není editace oblasti pro laserové ukazovátko dokončeno nebo dokud není editace zrušena. Pokud není uživatel spokojený s vybranou oblastí, může začít kreslit novou, a původní oblast se odstraní. Příklad vykreslené oblasti pomocí elipsy můžeme vidět na obrázku 5.3.



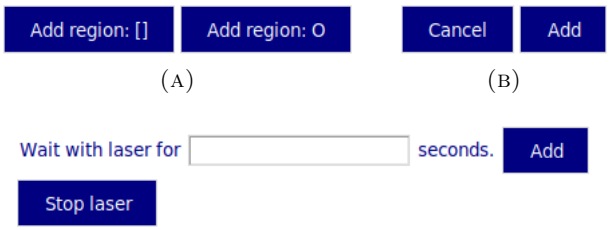

Obrázek 5.3 – Elipsová oblast pro laserové ukazovátko

Zrušit celou editaci pro laserové ukazovátko lze pomocí tlačítka *Cancel*. Pokud je uživatel s vybranou oblastí spokojený, pomocí tlačítka *Add* vloží definici této oblasti do textového pole. Pokud je tvarem obdélník, vloží se tag *StartRect*, v případě elipsy se jedná o tag *StartCircle*. Ve složených závorkách u těchto tagů jsou pak souřadnice levého horního rohu a pravého dolního rohu obdélníku, resp. obdélníku, který opisuje elipsu.

Jakmile je oblast vložena, je potřeba ještě zadefinovat, kdy má robot s laserovým ukazovátkem přestat svítit. To je možné udělat dvěma způsoby. Buď

je do textového pole pod tag *StartRect*, resp. *StartCircle*, vložen text, který má robot říkat v průběhu svícení laserovým ukazovátkem a následně stisknout tlačítko *Stop laser*. V tomto případě robot skončí ukazovat laserovým ukazovátkem hned po tom, co dopovídá. Pokud robot během svícení mluvit nemá, je možné nastavit počet vteřin, během kterých bude na slajd ukazovat. Při vepsání hodnoty a stisknutí tlačítka *Add* se do textu vloží tag *StopLaser* s danou hodnotou ve složených závorkách.

Během editace oblasti pro laserové ukazovátko se mění sestavení tlačítek pod textovým polem. Tato posloupnost je vidět na obrázku 5.4. Nejprve se vybírá tvar oblasti (obrázek (a)), poté je možné vložit oblast do textu (obrázek (b)) a nakonec jsou vidět tlačítka pro ukončení laserového ukazovátka (obrázek (c)).




Obrázek 5.4 – Posloupnost tlačítek při vytváření oblasti pro laserové ukazovátko

Tyto tagy se vkládají do textového pole na místo, kde je aktuálně kurzor. Tedy robot bude říkat text, který je nad tímto tagem, potom se otočí ke slajdu a ukáže laserovým ukazovátkem na určené místo, povídá nebo čeká předepsanou dobu, a nakonec přestane ukazovat, otočí se zpět k publiku a pokračuje v prezentaci.

Poslední část grafického uživatelského rozhraní obsahuje tři tlačítka. Tlačítko *Cancel* zavře editační okno a provedené změny neuloží. Po stisknutí tlačítka *Save* se změny uloží do souboru a otevře se dialogové okno s informací, že změny byly uloženy. Tlačítko *Run* nejprve uloží změny do souboru, potom zkontroluje, že soubor obsahuje všechny potřebné náležitosti, a nakonec spustí prezentaci robotem.

### 5.1.3 Struktura souboru .nao

Soubor .nao slouží k tomu, aby bylo možné rozpracovanou editaci slajdů uložit a později se k ní vrátit. V tomto souboru jsou uložené veškeré hodnoty, které byly zadané v uživatelském rozhraní.

Soubor je ukládán na stejné místo, kde je uložen soubor .pdf s prezentací. Název je taktéž shodný s tímto souborem.

Soubor se skládá z tagů a z hodnot těchto tagů. Aby bylo možné odlišit text prezentace od tagu, musí tagu předcházet zpětné lomítko, a tato dvojice musí začínat na novém řádku. Za tagem se v některých případech nachází hodnota ohraničená ve složených závorkách, některé tagy hodnoty neobsahují. Jakýkoliv text, který se nachází na stejné řádce za tagem, je ignorován.

#### 5.1.3.1 Seznam tagů

V souboru se mohou nacházet tři typy tagů - tagy s hodnotou, tagy bez hodnoty a speciální tag pro slajd, který obsahuje hodnotu (číslo slajdu) a v následujících řádcích po tagu přednášený text ke slajdu. Níže je uveden seznam tagů a k čemu slouží.

- *presentation* - jeho hodnotou je absolutní cesta k .pdf souboru, pro který je prezentace vytvářena
- *language* - zde je uložen jazyk, v jakém bude robot přednášet
- *robot* - tag udává, který robot bude přednášet
- *ip* - zde se ukládá ip adresa robota
- *port* - tag obsahuje port připojení k robotovi
- *hand* - zde je definována ruka, ve které robot drží ukazovátko
- *laserColor* - tag obsahuje barvu laserového ukazovátka
- *startRect* - tag určuje, že má robot začít ukazovat laserovým ukazovátkem, jeho hodnotou jsou souřadnice levého horního a pravého dolního rohu vymezené obdélníkové oblasti
- *startCircle* - tag určuje, že má robot začít ukazovat laserovým ukazovátkem, jeho hodnotou jsou souřadnice levého horního a pravého dolního rohu obdélníku, který opisuje vymezenou oblast ve tvaru elipsy
- *waitStop* - hodnota x v tomto tagu říká, že robot má přestat ukazovat laserovým ukazovátkem po x vteřinách
- *stopLaser* - tag určuje, že má robot skončit ukazovat laserovým ukazovátkem, tento tag neobsahuje žádnou hodnotu
- *slide* - tento tag určuje, že text na dalších řádcích patří ke slajdu definovaném číslem jako jeho hodnota. Text vždy patří ke slajdu, jehož tag je definován nejblíže nad ním

Zde je krátký příklad souboru .nao, přičemž prezentace má dva slajdy a v jednom z nich je použito laserové ukazovátko ve tvaru elipsy:

```
\presentation{/home/turcagab/prezentace.pdf}
\language{Czech}
\robot{Nao}
\ip{127.0.0.1}
\port{9559}
\hand{Right}
\laserColor{Green}

\slide{1}
Dobrý den,
jmenuji se Ervin a
vítám Vás u první prezentace přednášené robotem NAO.

\slide{2}
Rád bych Vám pověděl něco o sobě.
\startCircle{154:88,251:192}
Na tomto obrázku můžete vidět všechny moje klouby a jejich umístění.
\stopLaser
```

#### 5.1.3.2 Kontrola souboru .nao

Před spuštěním prezentace probíhá kontrola uloženého souboru, neboť bez některých hodnot by nemohla prezentace fungovat správně. Tato kontrola se spouští i v případě otevření souboru v aplikaci, ale zde je její funkce čistě informativní. Má uživatele upozornit, co je potřeba doladit, než bude možné prezentaci spustit.

V souboru .nao jsou kontrolovány tyto vlastnosti:

1. nic jiného, kromě definovaných tagů, nesmí začínat zpětným lomítkem na nové řádce,
2. pokud má mít tag hodnotu, je ohraničená složenými závorkami,
3. tagy *presentation*, *language*, *robot*, *ip*, *port*, *hand* a *laserColor* musí být v souboru definovány právě jednou a musejí obsahovat neprázdné hodnoty,
4. hodnota tagu *presentation* je absolutní cesta k souboru, který lze otevřít,
5. hondota tagu *language* je buď „Czech“ nebo „English“,
6. hodnota tagu *robot* je buď „Nao“ nebo „Pepper“,
7. hodnota tagu *ip* musí být platná IPv4 adresa,
8. hodnota tagu *port* musí být celé číslo,
9. hodnota tagu *hand* je buď „Left“ nebo „Right“,
10. hodnota tagu *laserColor* je „Red“, „Green“ nebo „Purple“,

11. hodnota tagů *startRect* a *startCircle* jsou souřadnice ve tvaru „x1:y1, x2:y2“, přičemž bod x1:y1 představuje levý horní roh obdélníku a bod x2:y2 představuje pravý horní roh obdélníku, tedy $x1 < x2$ a $y1 < y2$ (v OpenCV je bod 0:0 vlevo nahoře),

12. hodnota tagu *waitStop* je číslo,

13. hodnoty tagů *slide* musejí být čísla, která jdou vzestupně od 1,

14. po každém tagu *startRect* a *startCircle* musí být další následující tag buď *waitStop* nebo *stopLaser*

15. každému tagu *waitStop* a *stopLaser* musí předcházet buď *startRect* nebo *startCircle*

Pokud soubor splňuje všechny tyto vlastnosti, je možné prezentaci spustit.

## 5.2 Přednášení robotem

Jakmile se v aplikaci stiskne tlačítko *Run*, robot se z jakékoliv pozice, ve které se zrovna nachází, postaví. Potom poprosí uživatele, aby mu dal rozsvícené laserové ukazovátko do té ruky, kterou předtím zvolil v aplikaci. Na laserové ukazovátko robot čeká 10 vteřin, pak sevře prsty a začne s přednáškou.

U každého slajdu nejprve zkontroluje, zda se promítá správný slajd. Pokud je vše v pořádku, začne říkat zadaný text z aplikace. V opačném případě je prezentace ukončena.

Pokud má během textu ukazovat laserovým ukazovátkem, otočí se ke slajdu, nasměruje ukazovátko na správné místo, a pak se otočí zase zpátky k posluchačům a pokračuje v prezentaci. V textu níže jsou podrobně popsány všechny tyto úkony.

### 5.2.1 Kontrola přednášeného slajdu

Před tím, než začne robot o slajdu mluvit, musí zjistit, zda je promítaný správný slajd. To znamená, že se musí otočit ke slajdu, vyfotit promítanou plochu a porovnat promítaný slajd s očekávaným slajdem.

Pokud se robotovi nepodaří slajd najít nebo pokud jsou slajdy rozdílné, prezentace se ukončí.

První možnost může nastat v případě, že jsou buď špatné světelné podmínky a slajd příliš splývá se zdí, nebo že slajd vůbec promítaný není. V takovém případě by vůbec nemělo smysl v prezentaci pokračovat, neboť by robot mohl povídat bez zapnutého projektoru.

Protože se .pdf soubor, podle kterého má robot přednášet, předává přímo z aplikace, nemělo by se stát, že by se promítala jiná prezentace. Pokud by ale

uživatel měl aktivovanou rozšířenou obrazovku s projektorem a spustil by prezentaci na svém monitoru místo na projektové ploše, slajdy by se posluchačům nezobrazily, a proto by neměl robot s přednáškou začít.

Pokud se slajd podaří porovnat, robot může začít povídat o daném slajdu.

#### 5.2.1.1 Nasměrování robota ke slajdu

Předpokládá se, že slajdy jsou promítané po té ruce robota, ve které drží laserové ukazovátko. Robot se tedy bude otáčet po směru hodinových ručiček, pokud drží laserové ukazovátko v pravé ruce, pokud ho drží v ruce levé, bude se otáčet opačným směrem.

Není žádoucí, aby se přednášející otáčel zády k posluchačům, proto je robot otočen pouze o 100 stupňů a pro nalezení slajdu potom otáčí už jenom hlavou.

K tomu, aby se robot otočil okolo své osy, slouží metoda *moveTo* z modulu „ALMotion“, se kterým jsme se seznámili v sekci 3.1.4.1. Tato metoda přijímá 3 parametry: o kolik se má robot posunout dopředu/dozadu v metrech, o kolik se má posunout doleva/doprava v metrech a o kolik stupňů v radiánech se má otočit okolo své osy. V tomto případě se první dva parametry nastaví na nulu a poslední na 100 stupňů převedených na radiány.

Jakmile se robot dostane do této pozice, začne otáčet hlavou. Protože dopředu nevíme, kde bude robot stát, není jisté, o kolik by měl hlavu natočit. Robot ale není moc vysoký a slajdy bývají promítané aspoň jeden metr nad zemí. Proto je nastaven maximální možný záklon hlavy robota, aby se díval směrem nahoru. Otáčení hlavy do strany je pak nastaveno od 15 do 75 stupňů po 10 stupních. Robot hlavou přestane otáčet v okamžiku, kdy se mu podaří slajd nalézt. Pokud ani v 75 stupních slajd nenajde, je prezentace ukončena.

Pro otáčení hlavy je použita metoda *angleInterpolation* z modulu „ALMotion“. Ta přijímá 4 parametry: názvy kloubů, kterým se bude nastavovat pozice, nastavovaná pozice v radiánech, rychlost, v jaké má být změna provedena a zda jde o pozici absolutní či je závislá na aktuální poloze.

V tomto případě jsou klouby, které nás zajímají, „HeadPitch“ a „HeadYaw“. „HeadPitch“ je kloub, kterým je možné natáčet robotovu hlavu do strany. Zde je tedy hodnota tohoto kloubu nastavena postupně na 15 až 75 stupňů, pokud se otáčí směrem doleva, resp. na -15 až -75 stupňů, pokud se otáčí směrem doprava. „HeadYaw“ je vždy nastavena na -38 stupňů, abychom dosáhli největšího možného záklonu. Ne vždy je ale robot schopen dodržet zaklonění hlavy v takovém úhlu, například mu v poloze může bránit vlastní rameno. V takovém případě se automaticky nastaví největší možný úhel záklonu. Čas změny nastavení kloubů je nastaven na 1,5 vteřiny, aby robot hlavou trhaně nekmital, a zároveň, aby hledání slajdu netrvalo příliš dlouho.

Jak je možné v Pythonu naprogramovat natočení robota ke slajdu, pokud má slajdy promítané po jeho levé ruce, je znázorněno ve zdrojovém kódu 5.1.

Pokud robot zhodnotí, že je slajd v pořádku, otočí se zpátky k posluchačům stejným způsobem, jako se otočil na slajd. Ke srovnání robota si

pomůžeme metodou *goToPosture* z modulu „ALRobotPosture“, díky které se robot srovná - tedy natočí hlavu zpět dopředu. Potom se otočí o 100 stupňů zpátky v opačném směru.

Mohlo by se zdát, že je hledání slajdu pokaždé zbytečné. Nicméně otáčení robota kolem své osy není úplně přesné. Může se stát, že se podruhé natočí ke slajdu trošku jinak než poprvé. Kvůli tomu není možné u dalších slajdů používat pouze nastavení kloubů hlavy, při kterém robot poprvé slajd našel. Pro ušetření času je ale toto nastavení vždy vyzkoušeno jako první, pro případ, že by se robot natočil stejně.

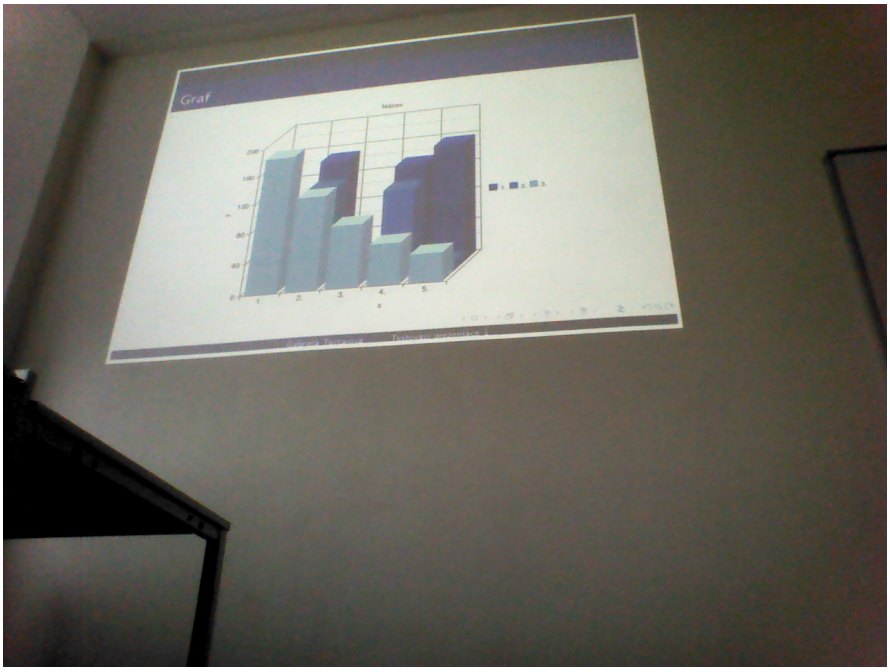
```
1  from naoqi import ALProxy
2
3  def move_to_slide(ip, port):
4    motion_proxy = ALProxy("ALMotion", ip, port) # pripojeni k modulu
5    theta = degree_to_radian(100) # funkce pro prepocet stupnu na radiany
6    motion_proxy.moveTo(0.0, 0.0, theta) # otoceni robota kolem sve osy
7    effectors = ["HeadYaw", "HeadPitch"] # klouby hlavy, se kterymi se bude hybat
8    shifts = ((15, 38), (25, -38), (35, -38), (45, -38), (55, -38), (65, -38), (75,
         -38)) # stupne posunuti hlavy doleva
9    for shift in shifts:
10     yaw = self.degree_to_rad(shift[0])
11     pitch = self.degree_to_rad(shift[1])
12     motion_proxy.angleInterpolation(effectors, [yaw, pitch], 1.5, True) #
           posouvani hlavy podle zadanych stupnu
13     if slide_found() == True: # funkce pro nalezeni slajdu z fotky
14       return True
15       break
16   return False
```

Zdrojový kód 5.1 – Nasměrování robota ke slajdu

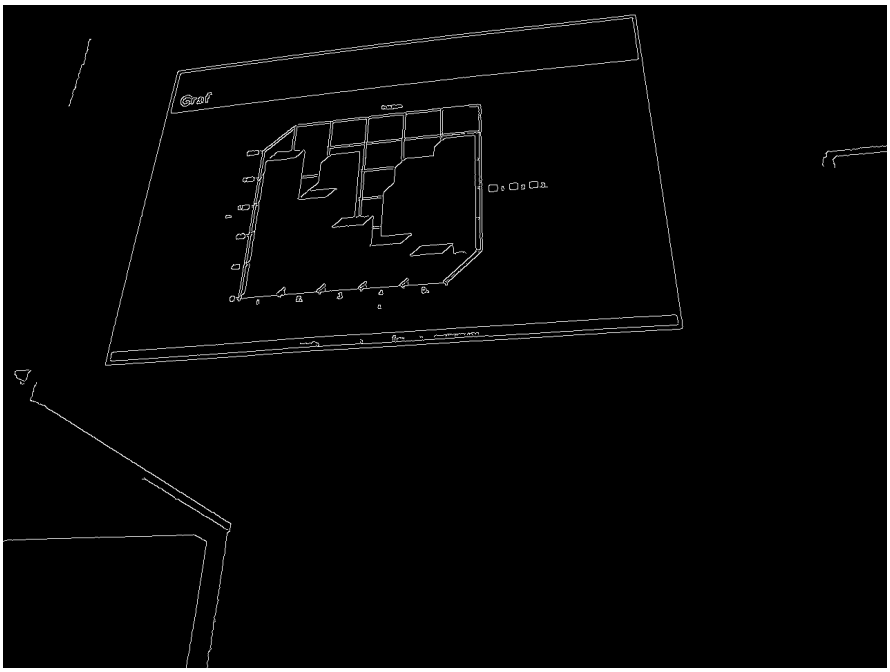
#### 5.2.1.2 Hledání slajdu na vyfoceném obrázku

Aby robot poznal, že se dívá na slajd, musí při každém natočení hlavy vyfotit snímek, který je potřeba dál zpracovat. K tomu nám pomůžou funkce z knihovny OpenCV, která byla popsána v sekci 3.2.6.

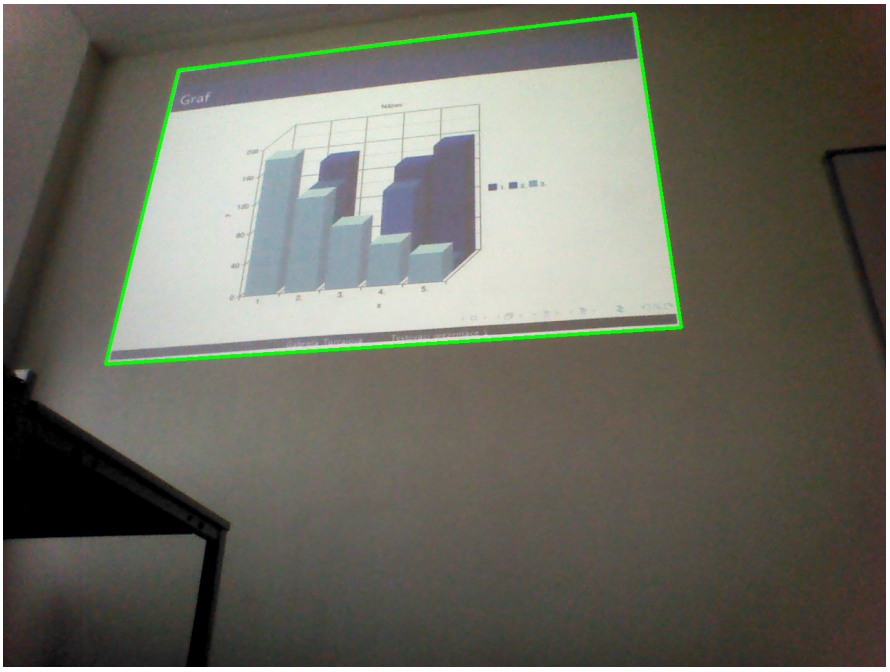
Nejprve převedeme obrázek do odstínů šedi pomocí funkce *cvtColor*. Potom pomocí Gaussovského filtru odstraníme případný šum. Funkce *Canny* z knihovny OpenCV obrázek převede na obrysový. Tyto funkce nám obrázek předzpracují a můžeme začít hledat slajd.




Obrázek 5.5 – Snímek vyfocený robotem




Obrázek 5.6 – Použití funkce *Canny* pro získání obrysového obrázku




Obrázek 5.7 – Nalezená kontura ohraničující slajd

Funkce *findContours* opět z knihovny OpenCV vyhledá všechny kontury, které se na obrázku nachází. Z nich je potřeba vybrat tu, která odpovídá hledanému slajdu. Ze všech kontur, které funkce *findContours* nalezla, nás zajímají pouze ty, které mají 4 souřadnice, tedy se jedná o čtyřúhelník. Z těchto pak vybíráme tu, která ohraničuje co největší plochu, neboť slajd by měl zabírat velkou část obrázku. I tak může být nalezena kontura i na snímku, kde není slajd nebo jeho část vidět. Aby mohla být kontura použita, musí zabírat alespoň 10 % snímku.

Jednotlivé úpravy snímku můžeme vidět na obrázcích 5.5 a 5.6 a 5.7. Na obrázku 5.5 je původní snímek vyfocený robotem. Na obrázku 5.6 je ten samý snímek, který byl převeden do odstínů šedi, potom na něj byl aplikován Gaussovský filtr a nakonec z něj byl vytvořen obrysový obrázek. Na obrázku 5.7 je pak zvýrazněna kontura nalezená funkcí *findContours*.

Problém nastává, když je během pořizování snímku moc světla. Hrany slajdu pak začínají splývat se zdí a pomocí výše zmíněných funkcí není možné hrany slajdu nalézt. Aby byla šance pro nalezení zvětšena, je lepší mít slajd ohraničený bílým rámečkem. Mezi slajdem a zdí pak vzniká větší přechod a funkce *Canny* pak pravděpodobněji vyhodnotí ohraničení slajdu jako hranu.

#### 5.2.1.3 Vyříznutí slajdu ze snímku a jeho transformace

Abychom mohli slajd porovnat, musíme ho nejprve vyříznout z robotem vyfoceného snímku. Slajd může být vyfocen z různých úhlů, protože nemáme

předem definováno, na jakém místě bude robot stát. Proto může být slajd různě zakřivený.

Pozici slajdu už ale máme zjištěnou pomocí nalezené kontury. Potom už nám stačí použít pouze dvojici funkcí *getPerspectiveTransform* a *warpPerspective* z knihovny OpenCV, které pomocí souřadnic této kontury vytvoří nový obrázek obsahující pouze slajd.

```
import cv2
import numpy as np

def transforme_image(contour, photo):
  (tl, tr, br, bl) = self.get_coordinates(contour) # ziskani rohu slajdu ve
      spravnem poradi z kontury

  # vzdalenost bodu pro ziskani sirky
  width_bot = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
  width_top = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))

  # vzdalenost bodu pro ziskani vysky
  height_right = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
  height_left = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))

  #vysledna velikost transformovaneho obrazku
  width = max(int(width_bot), int(width_top))
  height = max(int(height_right), int(height_left))

  dst = np.array([[0, 0], [width, 0], [width, height], [0, height]],
       dtype="float32")
  src = np.array([tl, tr, br, bl], dtype="float32")

  mapping = cv2.getPerspectiveTransform(src, dst) # ziskani mapovaci matice
  warp = cv2.warpPerspective(photo, mapping, (width, height)) # ziskani vysledneho
      obrazku
  return warp
```

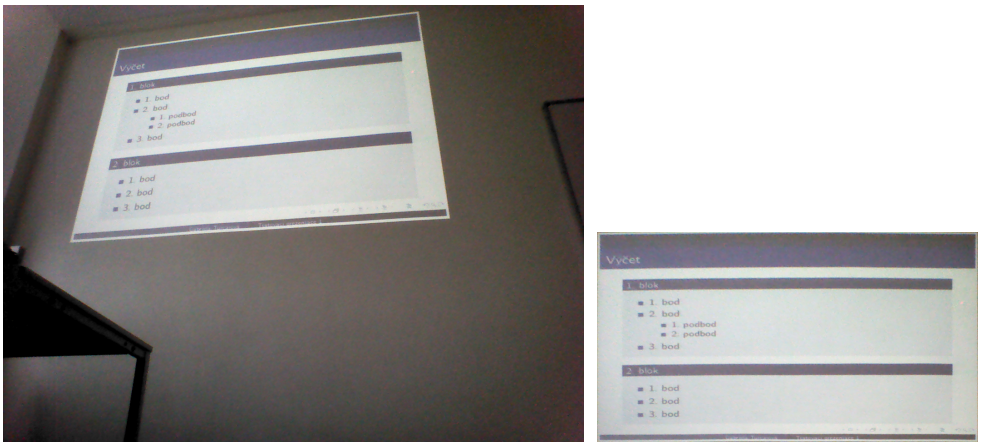
Zdrojový kód 5.2 – Nasměrování robota ke slajdu

Velikost výsledného obrázku je dána vzdáleností rohů slajdu. Jsou tedy nalezeny vzdálenosti mezi levým horním a pravým horním rohem a stejně tak i mezi levým dolním a pravým dolním rohem, a maximum z těchto dvou hodnot bude šířkou výsledného obrázku. Stejným způsobem pak získáme výšku výsledného obrázku.

Jakmile známe konturu slajdu a výslednou velikost, můžeme tyto hodnoty včetně vyfoceného snímku předat funkci *getPerspectiveTransform*. Ta nám vrátí mapovací matici, pomocí které, společně s funkcí *warpPerspective*,

získáme vyříznutý slajd.

Vyříznutí a trasformace obrázku v programovacím jazyce Python můžeme vidět ve zdrojovém kódu 5.2. Výsledek tohoto procesu je zobrazen na obrázku 5.8.




Obrázek 5.8 – Transformace snímku vyfoceného robotem

#### 5.2.1.4 Porovnání transformovaného obrázku se slajdem

Když máme obrázek transformovaný, můžeme provést samotné porovnání. První metodou, která se nabízela k vyzkoušení, bylo porovnání obrázků pomocí ORB deskriptorů [32], které vytvoří deskriptory pro oba obrázky pomocí funkce *BFMatcher* z knihovny OpenCV a potom každý z těchto deskriptorů jednoho obrázku porovnává se všemi deskriptory obrázku druhého a vypočítává pro ně Hammingovu vzdálenost.

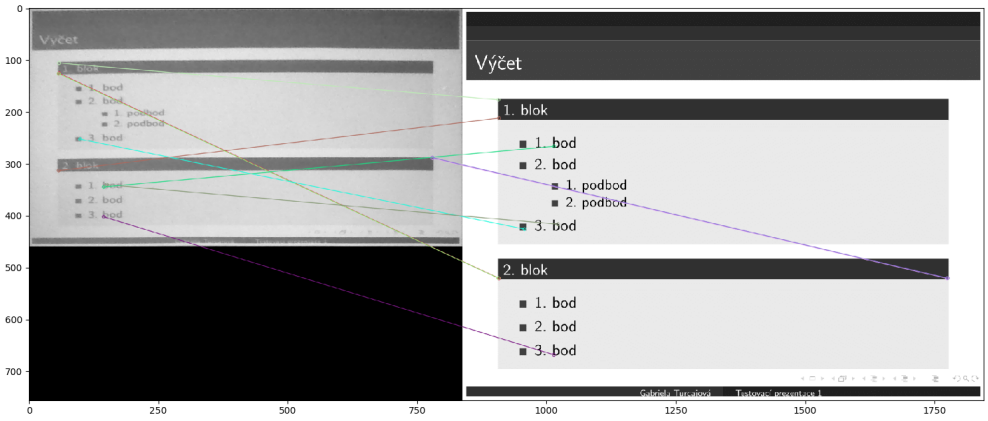
Tato metoda se ale nakonec ukázala jako nepraktická, neboť slajdy mají mnoho podobných prvků, takže jako body shody jsou označené i ty body, které se sice shodují, ale na špatném místě. Na obrázku 5.9 vidíme porovnání dvou stejných slajdů a 10 jejich nejlepších bodů shody, tedy body s nejmenší Hammingovou vzdáleností.

Některé shody jsou nalezeny dobře, např. třetí bod v druhém bloku, ale některé jsou nalezeny na špatném místě, např. hlavička druhého bloku má shodu s hlavičkou prvního bloku. Vzhledem k tomu, že slajdy mají většinou velmi podobné rozložení, tato metoda v tomto případě nemůže fungovat.

Další metodou bylo porovnání obrázků pomocí výpočtu střední kvadratické chyby. Střední kvadratická chyba [33] je definována jako:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2, \tag{5.1}$$

kde $m$ je výška obrázku, $n$ je šířka obrázku, $I$ první a $K$ druhý obrázek, přičemž $I(i,j)$ je pixel na $i$-tém řádku a v $j$-tém sloupci prvního obrázku.




Obrázek 5.9 – Porovnání obrázku pomocí ORB deskriptorů

Pro tuto metodu je potřebné, aby oba obrázky měly stejnou velikost. Proto je původní slajd pomocí funkce *resize* z knihovny OpenCV zmenšen tak, aby odpovídal vyfocenému slajdu. Vyfocený slajd má totiž menší velikost a jeho zvětšením bychom kvalitu obrázku zhoršili. Dále je porovnání prováděno na obrázcích v odstínech šedi, protože původní slajd má mnohem ostřejší barvy než ten vyfocený a vypočítaná chyba by se tím o dost zvýšila.

Když se obrázky shodují, chyba se pohybuje pod 7000. Pokud jsou obrázky rozdílné, chyba je vypočítána na 8000 a více. Práh pro chybu byl proto nastaven na 7500. Pokud je chyba menší než tento práh, je snímek označen za správný, v opačném případě se jedná o jiný slajd.

### 5.2.2 Ukázání laserovým ukazovátkem

Pokud program detekuje v textu přednášeného slajdu tag *StartRect* nebo *StartCircle*, předá robotovi, že má laserovým ukazovátkem ukázat na patřičné místo. To znamená, že se robot musí opět natočit směrem ke slajdu a z vyfoceného obrázku přepočítat, kam by měl laserové ukazovátko namířit.

Potom je potřeba, aby si ověřil, že namířil na správné místo. Pokud se bod laserového ukazovátka nenachází ve vymezené oblasti, musí polohu ukazovátka změnit. Pokud je ukazovátko namířeno správně, chová se dál podle toho, jak bylo zadáno v aplikaci, tedy buď pouze svítí zadanou dobu na slajd, nebo během svícení říká zadaný text.

Pak se otočí zpět k posluchačům a pokračuje v prezentaci. V dalších podsekcích je podrobněji popsáno, co všechno musí robot udělat, aby správně zamířil laserovým ukazovátkem do vymezené oblasti na slajdu.

#### 5.2.2.1 Namíření laserového ukazovátka na slajd

Aby robot mohl namířit laserové ukazovátko na zadané místo, musí se nejprve otočit ke slajdu. To udělá stejným způsobem jako při kontrole slajdu,

tedy tak, jak bylo popsáno v sekci 5.2.1.1. Potom už může mířit s laserovým ukazovátkem.

K míření laserovým ukazovátkem robot využívá dvou kloubů na paži, a to loketní kloub „ElbowYaw“ a kloub v zápěstí „WristYaw“. Pohybováním kloubu „ElbowYaw“ docílíme toho, že se laserové ukazovátko pohybuje doleva a doprava. Otáčením kloubu „WristYaw“ pohybujeme laserovým ukazovátkem nahoru a dolů.

Robot nejprve namíří laserovým ukazovátkem na počáteční pozici. Po vyzkoušení bylo zvoleno takové nastavení, že se kloub „ElbowYaw“ otočí na 45 stupňů a kloub „WristYaw“ na 70 stupňů. Zároveň je také nastaven kloub „ShoulderRoll“ na 15 stupňů, což způsobí, že robot ukazuje laserovým ukazovátkem směrem k projekční ploše. Všechny zmíněné hodnoty platí pro pravou paži robota, pro levou paži musí být tyto hodnoty záporné. Tímto nastavením bychom měli dosáhnout toho, že se bod laserového ukazovátka nachází v zorném poli robota.

Pohyb laserového ukazovátka na projekční ploše ovšem není úplně přesný. I přestože se otočením kloubu „WristYaw“ ukazovátko pohybuje nahoru nebo dolů, pohybuje se zároveň i trochu do strany. Aby byl pohyb co možná nejpřesnější, je při počátečním nastavení paže robota nastaven i druhý loketní kloub „ElbowRoll“ na 80 stupňů, resp. na -80 stupňů. Paže je díky tomu v lokti ohnutá.

Pro změny pozice kloubů je opět použita funkce *angleInterpolation* z modulu „ALMotion“. Ve chvíli, kdy má robot nasměrovat laserové ukazovátko na slajd, se použije počáteční nastavení laserového ukazovátka a zjistí se, jestli laserové ukazovátko spadá do vymezené oblasti. Pokud se bod laserového ukazovátka v této oblasti nenachází, vypočítá se, jak moc je bod vzdálený od středu požadované oblasti.

Abychom zjistili, že se bod laserového ukazovátka nachází ve vymezené oblasti, musíme opět použít funkce *getPerspectiveTransform* a *warpPerspective* z knihovny OpenCV. Stejným způsobem, jako v sekci 5.2.1.3, získáme transformovaný slajd ze snímku. Na tomto slajdu nalezneme laserové ukazovátko. Způsob, jakým detekujeme laserové ukazovátko na slajdu, je popsán v sekci 5.2.2.2.

Jakmile máme transformovaný slajd a bod na tomto slajdu, kam svítí laserové ukazovátko, můžeme zjistit, jestli se tento bod nachází ve vymezené oblasti.

Pokud se jedná o obdélníkovou oblast, je výpočet poměrně jednoduchý, neboť víme, že obdélník není nijak pootočený. Stačí nám poue spočítat, že se bod nachází mezi rohy obdélníku, tedy: $lhx < x$, $x < pdx$, $lhy < y$ a $y < pdy$, kde $lhx$ je $x$-ová složka a $lhy$ je $y$-ová složka levého horního bodu, $pdx$ je $x$-ová a $pdy$ $y$-ová složka pravého dolního rohu a $(x, y)$ je bod laserového ukazovátka.

V případě elipsové oblasti musíme znát střed $(h, k)$ a hlavní $a$ a vedlejší $b$ poloosu této elipsy. Všechny tyto hodnoty můžeme vypočítat pomocí levého horního a pravého dolního rohu obdélníku opisujícího tuto elipsu, neboť:

$$
\begin{aligned}
a &= (pdx - lhx)/2 \\
b &= (pdy - lhy)/2 \\
h &= a + lhx \\
k &= b + lhy
\end{aligned} \tag{5.2}
$$

Pokud je následující nerovnice splněna, bod se nachází uvnitř elipsy:

$$
\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} <= 1 \tag{5.3}
$$

Když zjistíme, že se laserové ukazovátko nenachází ve vymezené oblasti, musíme vypočítat vzdálenost bodu od středu této oblasti. K tomu opět potřebujeme transformaci obrazu, ale tentokrát opačnou - potřebujeme zjistit, kde se střed oblasti, který je definovaný pro slajd, nachází na snímku.

Znovu využijeme funkce *getPerspectiveTransform* z knihovny OpenCV, díky které získáme mapovací matici, která představuje transformaci slajdu do bodů kontury na snímku. Díky této matici pak můžeme převést kterýkoliv bod ze slajdu do slajdu na snímku. Stačí, když vynásobíme mapovací matici maticí $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$. Výsledkem budou tři hodnoty $\begin{pmatrix} a & b & c \end{pmatrix}$. Souřadnice na snímku pak jsou:

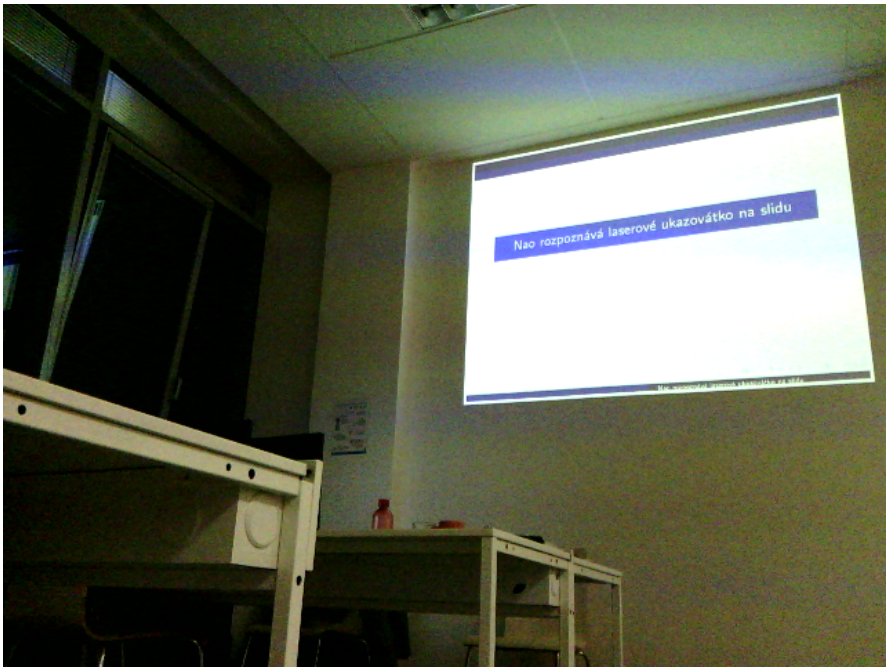
$$
\begin{aligned}
x' &= a/c, \\
y' &= b/c.
\end{aligned} \tag{5.4}
$$

Po tom, co získáme bod $(x', y')$, nám chybí už jen dopočítat vzdálenost laserového ukazovátka od středu vymezené oblasti. Protože nás zajímá, o kolik pixelů se musíme posunout po $x$-ové a $y$-ové ose, stačí pouze od sebe tyto dva body po složkách odečíst. Podle toho, jestli hodnoty vyšly kladně nebo záporně, otáčíme oba klouby o 2, resp. o -2, stupně.

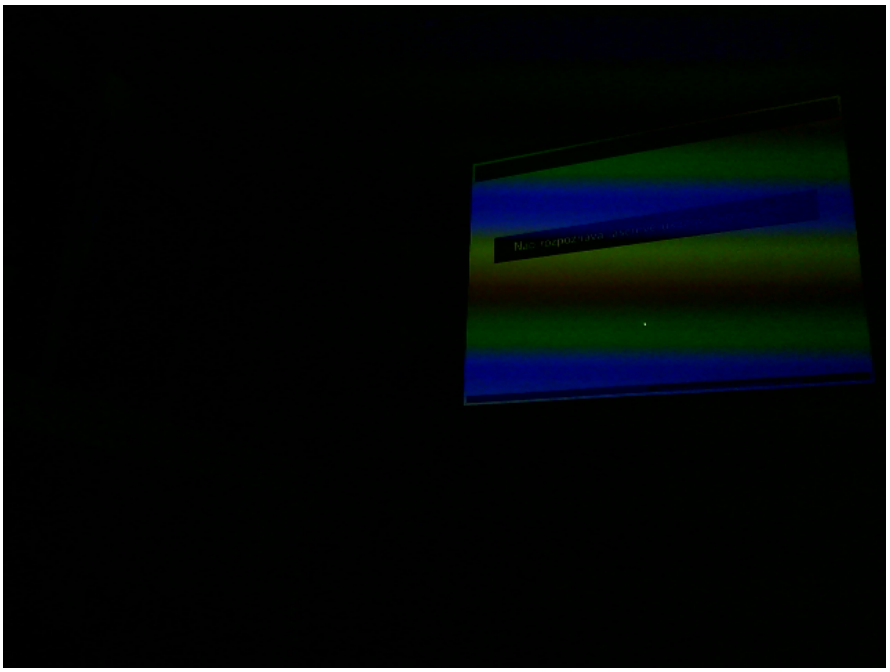
Tento proces opakujeme, dokud se laserové ukazovátko nedostane do vymezené oblasti, nejvýše ale desetkrát. Pokud se laserové ukazovátko nepodaří na správné místo dostat, nebo pokud ho není možné na snímku detekovat (například, že přestane svítit), ukončí se funkcionalita laserového ukazovátka. To znamená, že prezentace bude pokračovat dál, ale robot se již nebude pokoušet ukazovat laserovým ukazovátkem.

#### 5.2.2.2 Detekce laserového ukazovátka na slajdu

Můžeme předpokládat, že laserové ukazovátko promítaný slajd vždy přesvítí. Díky tomu můžeme jednoduše nalézt nejjasnější pixel obrázku pomocí funkce *minMaxLoc* z knihovny OpenCV. Protože tato funkce vrací pouze jeden nejjasnější pixel z obrázku, je lepší ještě předtím využít Gaussovský filtr pro odstranění šumu.




Obrázek 5.10 – Snímek s výchozím nastavením kamery



Obrázek 5.11 – Snímek s nastavenou kamerou pro detekci laserového ukazovátka

Problém může nastat ve chvíli, kdy laserové ukazovátko svítí na tmavou plochu slajdu. Tato metoda pak nedetekuje jako nejjasnější bod laserové ukazovátko, ale nějaký bod na světlejší ploše. Kvůli tomu bylo používáno laserové ukazovátko s větším průměrem světelného svazku.

Aby byla pravděpodobnost detekování laserového ukazovátka zvýšena, v době pořizování snímku bylo upraveno nastavení kamery. Doba expozice byla nastavena na 80 ms, oproti výchozímu nastavení bylo mírně zvýšeno zesílení obrazu (z 32 na 40), kontrast byl snížen z 32 na 30, a ostrost obrazu nastaven na maximální hodnotu 7.

```
proxy.setParam(vs.kCameraAutoWhiteBalanceID, 0)
proxy.setParam(vs.kCameraAutoGainID, 0)
proxy.setParam(vs.kCameraAutoExpositionID, 0)
proxy.setParam(vs.kCameraGainID, 40)
proxy.setParam(vs.kCameraExposureID, 80)
proxy.setParam(vs.kCameraContrastID, 30)
proxy.setParam(vs.kCameraSharpnessID, 7)
```

Na obrázcích 5.10 a 5.11 můžeme vidět, jak je díky tomuto nastavení kamery laserové ukazovátko viditelnější. Oba snímky byly pořízeny v podobnou dobu, na obrázku 5.10 není laserové ukazovátko vůbec vidět, na druhém obrázku 5.11 na něm díky nastavení kamery vynikne.

Obrázek má samozřejmě nejjasnější bod, i v případě, že na něj laserové ukazovátko nesvítí. Je tedy potřeba rozlišit, kdy na slajdu laser je a kdy není.

K tomu, abychom zjistili, jestli je nejjasnější bod laserovým ukazovátkem, můžeme využít kontrolu barvy bodu, která by se měla shodovat s barvou laserového ukazovátka, a srovnání jasového profilu s modelem. Obě tyto metody jsou provedeny pouze na okolí nejjasnějšího pixelu na obrázku. Z obrázku je tedy vyříznutá malá část o velikosti $11 \times 11$ pixelů, přičemž přesně uprostřed se nachází nejjasnější bod slajdu.

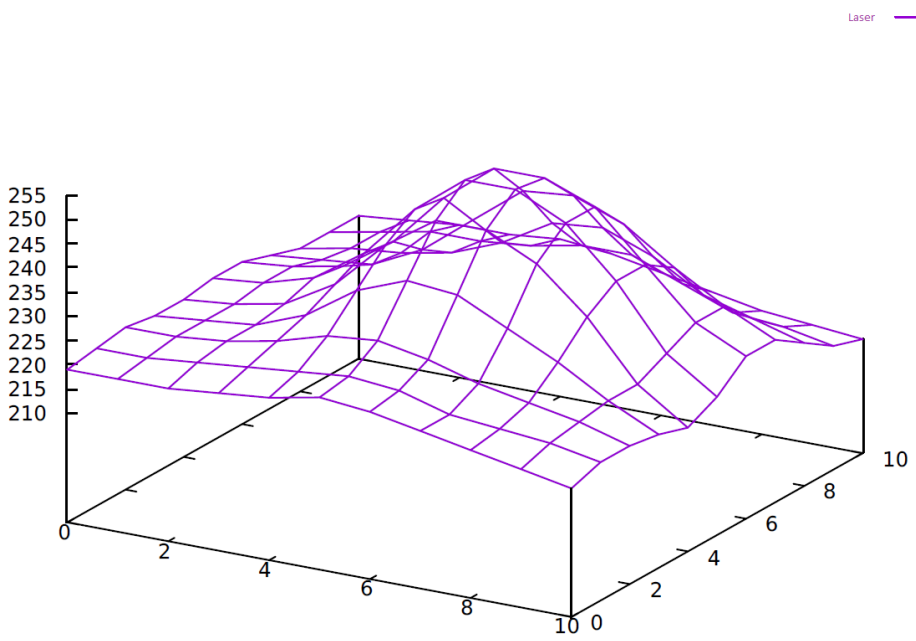
Pro kontrolu barvy převedeme vyříznutý obrázek do barevného modelu HSV, který nám umožní jednoduše určit barvu jednotlivých pixelů. Rozpoznáváme tři různé barvy: zelenou, červenou a fialovou. Pro rozpoznání dané barvy můžeme použít funkci *inRange* z knihovny OpenCV, která přijíma tři parametry - obrázek v barevném modelu HSV a spodní a horní hranici barvy v HSV.

Pro naše tři barvy byly nastaveny hranice následovně:

```
GREEN_BOT = (60-SENSITIVITY, 100, 50)
GREEN_TOP = (60+SENSITIVITY, 255, 255)
PURPLE_BOT = (140-SENSITIVITY, 70, 50)
PURPLE_TOP = (140+SENSITIVITY, 255, 255)
RED_BOT1 = (0, 100, 50)
RED_TOP1 = (0+SENSITIVITY, 255, 255)
RED_BOT2 = (180-SENSITIVITY, 100, 50)
RED_TOP2 = (180, 100, 50)
```

Protože se červená barva nachází na začátku i na konci modelu HSV, je nutné kontrolovat barvu pro obě tyto části.

Funkce *inRange* pak vrací pole, které pro jednotlivé pixely zaznamenává, zda se nachází v daném rozsahu. Pro pixel, který se v tomto rozsahu nenachází, nastavuje 0, v opačném případě nastavuje 255.




Obrázek 5.12 – Profil jasu v okolí nejjasnějšího bodu s laserem

Po provedení funkce *inRange* vypočítáme střední hodnotu výsledného pole. Pokud je výsledná hodnota větší než 20, říkáme, že daný nejjasnější bod je laserovým ukazovátkem. Pokud se výsledná hodnota rovná 0, je jisté, že se nejedná o laserové ukazovátko. Pokud není splněna ani jedna z těchto podmínek, přejdeme na kontrolu pomocí srovnání jasového profilu s modelem.

Tato situace nastává, pokud se laserové ukazovátko nachází na světlé ploše. Daný bod je totiž potom tak přesvícený, že barva laserového ukazovátka se nachází pouze po okraji světělného svazku. Na tmavých plochách naopak barva laserového ukazovátka vyniká, jeho jas je ale mírně pohlcen.

Pro porovnání modelů můžeme využít lineární regresi. Potom je potřeba data vyříznutého obrázku znormalizovat. Nejprve byla použita MinMax normalizace. Zde ale nastaly hned dva problémy.

Protože se hodnoty rovnoměrně rozprostírají na intervalu $\langle 0; 1 \rangle$, minimální hodnota v datech musí odpovídat 0. Z toho ale vzniká nechtěné dělení nulou v další části programu.

Dalším problémem je to, že křivka z těchto hodnot může změnit svůj tvar. Ta část křivky odpovídající maximálním hodnotám se roztáhne nahoru i v případě, že původní křivka je docela plochá.

První problém by se dal vyřešit posunutím křivky o maličko výš, druhý problém nám ale dost ztěžuje rozpoznání, zda se jedná o laserové ukazovátko nebo ne. Proto je jako normalizace od hodnot odečteno minimum z těchto

hodnot a přičtena jednička. Tím je vyřešen problém s nulou a zachováme tak i tvar křivky.

Na obrázku 5.12 můžeme vidět profil jasu v okolí nejjasnějšího bodu s laserem, která je velmi podobná Gaussovské křivce [34]. Předpis pro získaná data tedy je:

$$z = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5.5}$$

Pomocí jednoduchých matematických operací získáváme:

$$\begin{aligned} \ln z &= -\frac{x^2+y^2}{2\sigma^2} \\ \ln z - (-\frac{1}{2\sigma^2}(x^2+y^2)) &= 0 \end{aligned} \tag{5.6}$$

kde $z$ odpovídá získaným hodnotám, $x$ a $y$ zase pozici daného pixelu (tedy od -5 do 6). Lineární regrese [35] má předpis:

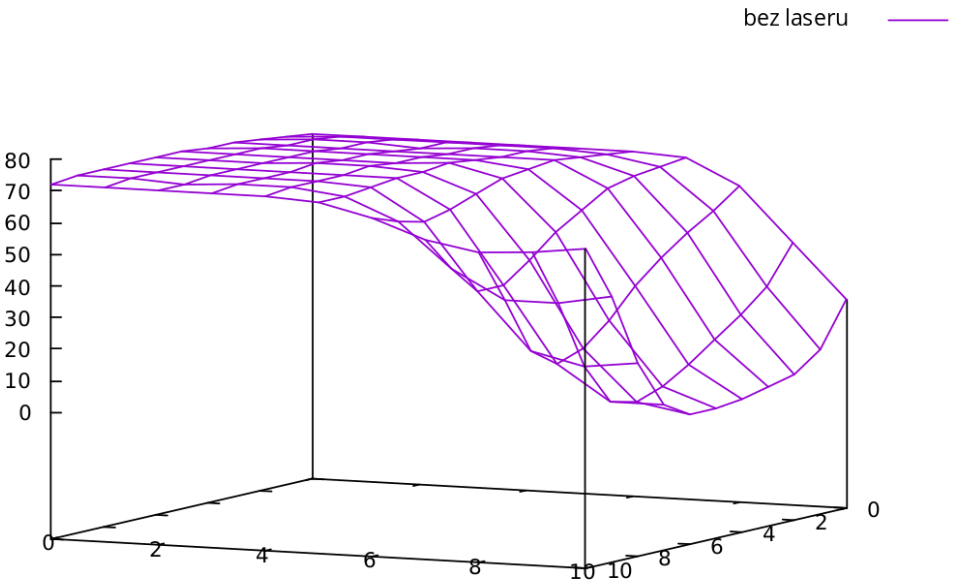
$$\begin{aligned} q_1\varphi + q_0 &= \psi \\ \psi - q_1\varphi - q_0 &= 0 \end{aligned} \tag{5.7}$$

Když porovnáme rovnici lineární regrese s rovnicí získaných dat, můžeme říct, že $\psi = \ln z$, $\varphi = (x^2 + y^2)$ a $q_1 = -\frac{1}{2\sigma^2}$. Z toho můžeme vypočítat koeficient $q_1$ pomocí vzorečku:

$$q_1 = \frac{mean(xy)}{mean(x^2)}, \tag{5.8}$$

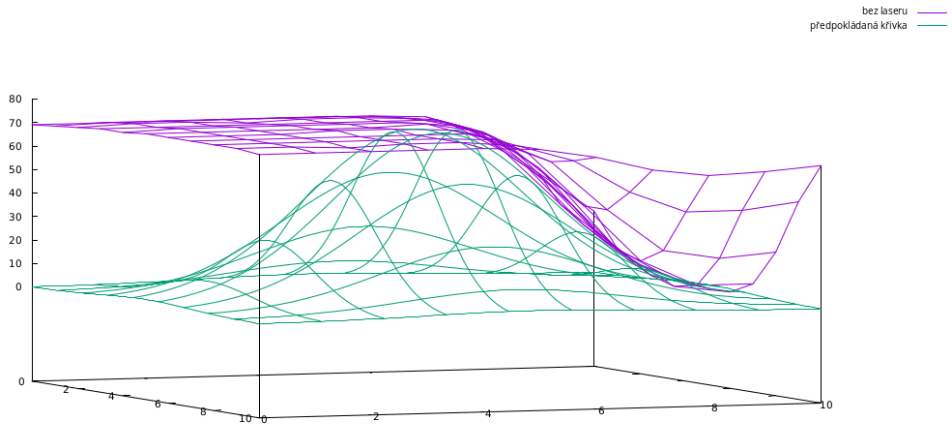
přičemž zanedbáváme koeficient $q_0$. Když máme hodnotu $q_1$, vypočítáme si, jak by s takovýmto koeficientem měla vypadat přesná Gaussovská křivka pomocí:

$$model = normData.max() * e^{(q_1*(-(x^2+y^2))}. \tag{5.9}$$




Obrázek 5.13 – Profil jasu v okolí nejjasnějšího bodu bez laseru

Když odečteme od normalizovaných dat hodnoty tohoto modelu, získáme chybu mezi těmito dvěma křivkami. Pomocí 2. momentu můžeme vypočítat rozptyl těchto chyb. Čím větší tento rozptyl je, tím rozdílnější křivky jsou. Pro rozhodnutí, zda se jedná o laserové ukazovátko, byl zvolen práh 50. Tedy pokud je rozptyl menší než 50, bereme tento bod jako laserové ukazovátko, pokud je vyšší, říkáme, že daný bod laserovým ukazovátkem není.




Obrázek 5.14 – Rozdíl mezi modelem a bodu s jeho okolím bez laseru

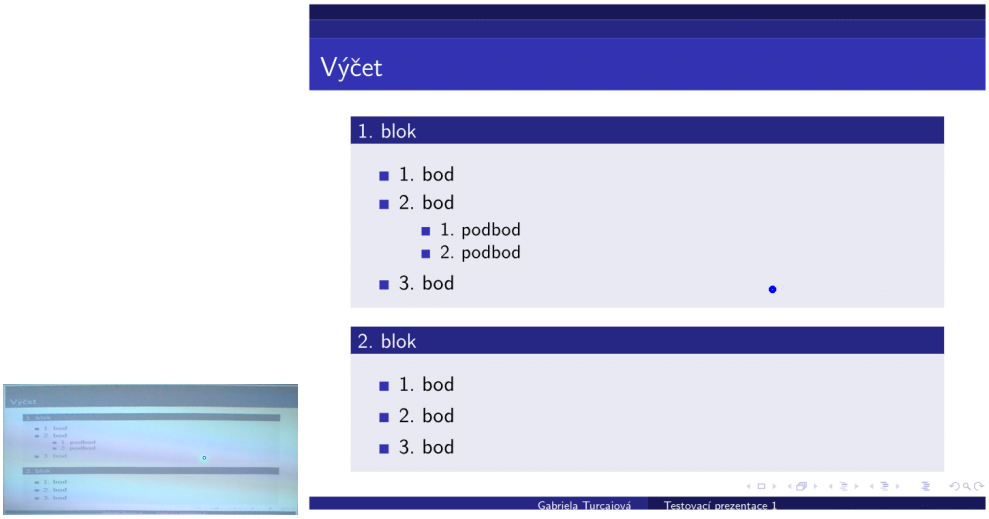
Na obrázku 5.13 můžeme vidět v okolí nejjasnějšího bodu bez laseru. Tento profil se skoro vůbec nepodobá Gaussovské křivce. Na obrázku 5.14 vidíme ze-

lenou barvou vykreslený model pro data tohoto bodu, aby se jednalo o laserové ukazovátko. Fialovou barvou je pak vykreslený profil nejjasnějšího bodu a jeho okolí. Fialová křivka se jen velmi málo podobá té zelené. V tomto případě bude mít rozptyl chyb vysokou hodnotu.

Po tom, co získáme souřadnice laserového ukazovátka na vyfoceném slajdu, je potřebujeme převést tak, aby odpovídaly bodu na původním slajdu. K tomu můžeme využít vzorečky:

$$x_{slajd} = \frac{x_{foto}}{delka_{foto} * delka_{slajd}}$$
$$y_{slajd} = \frac{y_{foto}}{vyska_{foto} * vyska_{slajd}} \tag{5.10}$$

Jakmile máme vypočtené $x_{slajd}$ a $y_{slajd}$, mužeme jednoduše překreslit laserové ukazovátko do slajdu, jak můžeme vidět na obrázku 5.15.




Obrázek 5.15 – Zobrazení laserového ukazovátka na původním slajdu

### 5.2.3 Přepínání slajdů

Pro přepínání slajdů byla použita knihovna „PyAutoGUI“. Tato knihovna umožňuje ovládat klávesy a myš místo uživatele. Na začátku je pomocí modulu *os* spuštěna prezentace. Hned potom je stisknuté tlačítko *f5* pomocí knihovny „PyAutoGUI“. Vždy, když robot dopovídá o daném slajdu, stiskne se šipka doprava. Po skončení prezentace se stiskne tlačítko *escape*.

### 5.2.4 Konec prezentace

Po tom, co robot dohovoří o posledním slajdu, se pomocí modulu „ALRobotPosture“ dostane do pozice „posed“. Aplikace běží dál, dokud se nezavře

editační okno prezentace.

Kapitola 6

# Testování a diskuze

V této kapitole se podíváme, jak byla aplikace testována a jakým způsobem může být spuštěna. Potom si ukážeme, jak můžou být otestovány jednotlivé části programu, pokud nemáme k dispozici robota NAO.

V druhé části této kapitoly se seznámíme s existujícími řešeními a možným rozšířením této práce.

## 6.1 Testování

Výsledná aplikace byla testována dvěma způsoby - program jako celek, kdy je k dispozici robot NAO, a dílčí části aplikace, u kterých robot NAO nemusí být přítomný.

### 6.1.1 Testování celého programu

Grafické uživatelské rozhraní bylo vyvíjeno na operačním systému Ubuntu 18.04 v programovacím jazyku Python verze 2.7. Testováno bylo i na operačním systému openSUSE. Protože knihovna TKinter přebírá styl od operačního systému, v obou systémech se uživatelské rozhraní zobrazuje trochu jinak. Na funkcionalitu to ale nemá žádný vliv.

Celá aplikace byla testována na robotovi NAO verze 6. Program byl spouštěn na operačním systému openSUSE běžícím ve virtuálním stroji. Tento virtuální stroj byl vytvořen pomocí virtualizačního nástroje Oracle VM VirtualBox.

Aplikaci je možné spustit pomocí následujícího příkazu v příkazové řádce na linuxovém operačním systému s nainstalovaným Pythonem verze 2.7:

```
python NAOPresenting.py
```

Pro spuštění této aplikace je nutné mít nainstalované následující balíčky pro Python ve verzi 2.7:

- NAOqi

- Tkinter
- Pillow
- pdf2image
- NumPy
- SciPy
- PyAutoGUI

### 6.1.2 Další možnosti testování

V průběhu vývoje byly vytvořeny dílčí části aplikace, které nejsou závislé na knihovně NAOqi, tedy není potřeba k jejich spuštění mít robota NAO. Všechny tyto programy se nachází na přiloženém CD ve složce „testing“. Ve stejné složce se nachází v podsložce „data“ i testovací data - fotky vyfocené robotem, testovací prezentace a její slajdy rozdělené do obrázků.

#### 6.1.2.1 Grafické uživatelské rozhraní

Pro testování grafického uživatelského rozhraní byla vytvořena samostatná aplikace, která je identická s uživatelským rozhraním výsledné aplikace, pouze při stisknutí tlačítka *Run* se otevře .pdf soubor a v intervalu dvou vteřin se přepínají slajdy.

Pro spuštění této aplikace je nutné mít nainstalované následující balíčky pro Python:

- Tkinter
- Pillow
- pdf2image
- NumPy
- PyAutoGUI

Soubory této aplikace se nachází na přiloženém CD ve složce „testing/gui“. Aplikaci je možné spustit pomocí následujícího příkazu v příkazové řádce na jakémkoliv linuxovém operačním systému s nainstalovaným Pythonem verze 2.7:

```
python SelectFile.py
```

Po spuštění se objeví okno pro vybrání souboru s prezentací, která se bude upravovat. Po vybrání souboru se zobrazí editační okno, kde lze zadat základní informace o robotovi a jednotlivé texty ke slajdům. Je možné rozpracovanou prezentaci uložit do souboru s příponou .nao. Při stisknutí tlačítka *Run* se zkontroluje aktuální stav souboru a spustí se prezentace s přepínáním slajdů. Podrobné informace o tom, co vše grafické uživatelské rozhraní umí, jsou popsány v sekci 5.1.

#### 6.1.2.2 Nalezení slajdu na snímku

Pro otestování nalezení slajdu na snímku byl vytvořen program s názvem „FindSlide.py“, který je uložen ve složce „testing“ na přiloženém CD.

Výstupem tohoto krátkého programu je vizualizace nalezení slajdu na snímku vyfoceným robotem. Program požaduje v parametru předat cestu k fotografii, na které se má nalézt slajd.

Na konci programu, pokud je slajd nalezen, se otevře okno se vstupní fotkou se zvýrazněným okrajem slajdu. Program následně čeká na stisknutí klávesy, a potom se ukončí. Pokud slajd nalezen není, vypíše se o tom zpráva do příkazové řádky.

Pro otestování programu lze použít obrázky uložené ve složce „testing/data/photo“ na přiloženém CD.

Pro spuštění je potřeba mít nainstalované balíčky „NumPy“ a „OpenCV“. Program lze spustit na linuxovém operačním systému s nainstalovaným Pythonem ve verzi 2.7.

Použití tohoto programu je následující:

```
usage: FindSlide.py [-h] photo

positional arguments:
  photo              path to input photo

optional arguments:
  -h, --help         show this help message and exit
```

#### 6.1.2.3 Porovnání slajdu se slajdem na snímku

K otestování porovnání slajdů slouží program „CheckSlides.py“, který je uložen ve složce „testing“.

Program na vstupu očekává cesty ke dvěma souborům - snímek vyfotografovaný robotem a slajd, o kterém by měl robot právě hovořit.

Výstupem tohoto programu je zpráva obsahující informaci, zda se slajdy shodují nebo ne. Tato zpráva je vypsána do příkazové řádky.

Pro otestování je možné použít snímky vyfotografované robotem, které jsou uložené ve složce „testing/data/photo“, odpovídající slajdy prezentace jsou uloženy ve složce „testing/data/slides“.

Pro spuštění je potřeba mít nainstalované balíčky „NumPy“ a „OpenCV“. Program lze spustit na linuxovém operačním systému s nainstalovaným Pythonem ve verzi 2.7.

Použití tohoto programu je následující:

```
usage: CheckSlides.py [-h] photo slide

positional arguments:
  photo               path to input photo
  slide               path to input slide

optional arguments:
  -h, --help          show this help message and exit
```

#### 6.1.2.4 Detekce laserového ukazovátka

Pro zjištění, zda a kde se na vstupním snímku nachází laserové ukazovátko, slouží program „DetectLaser.py“, který je uložen ve složce „testing“ na přiloženém CD.

Program na vstupu očekává cesty ke dvěma souborům - snímek vyfotografovaný robotem a slajd, o kterém by měl robot právě hovořit.

Výstupem tohoto programu je zpráva obsahující informaci, zda se na slajdu nachází laserové ukazovátko.

Pokud laserové ukazovátko bylo nalezeno, vypíše se na příkazovou řádku, kde se nachází, a zobrazí se slajd s dokresleným laserovým ukazovátkem a vstupní snímek vyfotografovaný robotem pro porovnání.

Pokud laserové ukazovátko nalezeno nebylo, vypíše se tato informace na příkazovou řádku. Případně, pokud se nepodařilo nalézt slajd, aplikace se v tomto momentu ukončí a napíše důvod do příkazové řádky.

Pro spuštění je potřeba mít nainstalované balíčky „NumPy“, „SciPy“ a „OpenCV“. Program lze spustit na linuxovém operačním systému s nainstalovaným Pythonem ve verzi 2.7.

Použití tohoto programu je následující:

```
usage: DetectLaser.py [-h] photo slide

positional arguments:
  photo               path to input photo
  slide               path to input slide

optional arguments:
  -h, --help          show this help message and exit
```

#### 6.1.2.5 Transformace bodu ze slajdu do snímku

Pro otestování transformace bodu definovaném na slajdu do vyfoceného snímku slouží program „TransformPoint.py“, který je uložen ve složce „testing“ na přiloženém CD.

Program na vstupu očekává cesty ke dvěma souborům - snímek vyfotografovaný robotem a slajd, o kterém by měl robot právě hovořit. Dále pak očekává bod definovaný hodnotou x a y, který se má transformovat do snímku.

Výstupem programu je pozice na snímku, kde se transformovaný bod nachází. Dále je pak bod nakreslen do obou vstupních obrázků, a ty jsou zobrazeny uživateli.

Pokud se z nějakého důvodu nepodaří bod transformovat (například není nalezen slajd na vyfoceném snímku nebo je bod definovaný mimo dimenzi obrázku), tato informace se vypíše na příkazové řádce.

Pro otestování je možné použít snímky vyfotografované robotem, které jsou uložené ve složce „testing/data/photo“, odpovídající slajdy prezentace jsou uloženy ve složce „testing/data/slides“.

Pro spuštění je potřeba mít nainstalované balíčky „NumPy“, „SciPy“ a „OpenCV“. Program lze spustit na linuxovém operačním systému s nainstalovaným Pythonem ve verzi 2.7.

Použití tohoto programu je následující:

```
usage: TransformPoint.py [-h] photo slide x y

positional arguments:
  photo               path to input photo
  slide               path to input slide
  x                   x of point to be transformed
  y                   y of point to be transformed

optional arguments:
  -h, --help          show this help message and exit
```

## 6.2 Možné rozšíření aplikace

Jednotlivé části práce by mohly být dál rozvíjeny. Příhodné by bylo naučit robota, aby si sám mohl zapínat a vypínat laserové ukazovátko. Nyní totiž robot drží laserové ukazovátko pořád zapnuté, což není úplně praktické. Kromě toho, že se můžou v průběhu přednášky vybít baterie v laserovém ukazovátku, může také robot nechtěně namířit ukazovátko do publika.

Vylepšená by mohla být i část porovnávání slajdů. Metoda pomocí střední kvadratické chyby sice dokáže správně určit, že dva slajdy s různým rozložením jsou rozdílné. Ale pokud bude potřeba porovnat dva slajdy, na obou bude výčtový seznam jenom s rozdílným textem, tyto slajdy budou označeny za stejné. Protože je ale díky aplikaci zajištěno, že bude promítána správná prezentace, měla by být metoda pomocí střední kvadratické chyby dostačující.

Určitě by stálo za to vyzkoušet aplikaci i na robotovi Pepper. Robot Pepper je také dílem společnosti SoftBank Robotics, který působí reprezentativněji, proto by se mohl hodit na přednášení víc než robot NAO. Další výhodou by mohlo být plynulejší otáčení robota Pepper ke slajdům a zpět k posluchačům, neboť nemá nohy jako robot NAO, ale jezdí na kolečkách. U robota NAO se rychle přehřejí motory na nohách a potom nemůže prezentaci dokončit.

V aplikaci by nemělo být třeba dělat mnoho úprav, pouze změnit některé nastavení kloubů, neboť je robot Pepper vyšší. Proto by například nepotřeboval pro nalezení slajdu tolik zaklánět hlavu.

# Závěr

Cílem této práce bylo vytvořit aplikaci, díky které bude robot NAO schopný přednášet podle zadaných textů v prezentaci.

Nejprve jsme se seznámili s technickým přehledem robota NAO. Ukázali jsme si, jak robot NAO vypadá, jaké má komponenty a jaké klouby mu umožňují pohyb. Potom jsme se podívali na podrobný přehled dvou kamer umístěných na hlavě robota. Nakonec byl popsán systém, který na robotovi běží a díky kterému ho můžeme ovládat. Byly podrobně popsány některé z předpřipravených modulů, díky kterým lze jednoduše s robotem pohybovat či získávat data z robota.

Potom jsme se podívali na některé knihovny Pythonu, které by mohly být využity pro vytvoření požadované aplikace. Hlavními knihovnami jsou knihovny Tkinter a OpenCV. Tkinter umožňuje vytvořit jednoduché grafické uživatelské rozhraní, které bylo využito pro zadávání textů uživatelem k jednotlivým slajdům. Knihovna OpenCV slouží především ke zpracování obrázků. Tato knihovna je v aplikaci hojně využívána například pro hledání slajdu robotem nebo pro detekci laserového ukazovátka na slajdu.

V další kapitole byl ukázán návrh aplikace. Nejprve bylo popsáno, jaké hodnoty jsou potřeba uživatelem zadat, aby mohla být prezentace spuštěna. Potom byly vyjmenovány všechny činnosti, které musí při přednášce robot zastávat. Nakonec bylo navrženo třídní rozdělení výsledné aplikace.

V další části práce je popsána realizace aplikace. Pro spuštění prezentace musí být správně vyplněná políčka v grafickém uživatelském rozhraní. Tyto hodnoty musí uživatel před spuštěním přednášky vyplnit. Pak probíhají kontroly vstupu, které vyhodnotí, zda má robot veškeré potřebné informace pro to, aby mohl začít s přednáškou.

Samotné přednášení robotem je rozděleno do několika činností.

Na začátku se robot postaví a požádá o laserové ukazovátko. Potom pro každý slajd prezentace zkontroluje, zda je na plátně promítaný slajd. To znamená, že se musí k prezentovanému slajdu natočit, najít ho a nakonec porovnat, zda se slajdy shodují.

Když robot zhodnotí, že je slajd správný, může o něm začít hovořit. Pokud se dostane do situace, že by měl ukázat laserovým ukazovátkem na slajd, musí se opět nasměrovat ke slajdu a namířit na dané místo laserovým ukazovátkem.

Na konci prezentace se robot posadí a aplikace je ukončena po zavření editačního okna.

Kromě celé aplikace lze její jednotlivé části testovat pomocí krátkých programů. Tyto programy lze spustit i v případě, že zrovna není robot NAO k dispozici, což může testování usnadnit.

V této práci se povedlo vytvořit aplikaci spustitelnou na linuxových operačních systémech. Aplikace obsahuje grafické uživatelské rozhraní pro editaci textu ke slajdům pro robota. Po spuštění prezentace robot pohovoří o každém slajdu, u každého slajdu zjistí, zda se jedná o očekávaný slajd, a případně ukáže laserovým ukazovátkem na slajd.

Některé části aplikace by mohly být vylepšeny, především možnost zapínat a vypínat laserové ukazovátko robotem. Stálo by za to vyzkoušet aplikaci i na robotovi Pepper, neboť by nemělo být třeba v aplikaci toho mnoho měnit.

# Literatura


[1] LEHÁR, Adam a Ondřej ŠMÍD: NAO učitel [online]. 2015, [cit. 2018-01-06]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-IVS/labs/mi_ivs_prj/nao_teacher

[2] SoftBank Robotics: Ask Nao — New Way of Teaching? [online]. 2019, [cit. 2018-01-06]. Dostupné z: https://asknao.aldebaran.com/

[3] GRIFFITHS, Andrew: The robot teacher connecting with autistic children [online]. 2014, [cit. 2018-01-06]. Dostupné z: https://www.telegraph.co.uk/technology/news/10632937/The-robot-teacher-connecting-with-autistic-children.html

[4] SoftBank Robotics: Company: Who is SoftBank Robotics? [online]. 2019, [cit. 2019-01-04]. Dostupné z: https://www.softbankrobotics.com/emea/en/company

[5] SoftBank Robotics: NAO: the humanoid robot [online]. 2019, [cit. 2019-01-04]. Dostupné z: https://www.softbankrobotics.com/emea/en/nao

[6] SoftBank Robotics: Links — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/family/nao_technical/links_naov6.html

[7] SoftBank Robotics: Technical overview — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/home_nao.html

[8] Gigabotics: Features of Nao Robot [online]. 2017, [cit. 2019-01-05]. Dostupné z: http://gigabotics.com/robotics/features-of-nao-robot/

[9] SoftBank Robotics: Joints — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/family/nao_technical/video_naov6.html

[10] SoftBank Robotics: Video cameras — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/family/nao_technical/links_naov6.html

[11] Aldebaran Robotics: NAOqi Framework — NAO Software 1.14.5 documentation [online]. 2017, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/1-14/dev/naoqi/index.html

[12] SoftBank Robotics: ALMotion — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/motion/almotion.html

[13] SoftBank Robotics: ALTracker — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/trackers/index.html

[14] SoftBank Robotics: ALRobotPosture — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/motion/alrobotposture.html

[15] SoftBank Robotics: ALVideoDevice — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/vision/alvideodevice.html

[16] SoftBank Robotics: ALVideoRecorder — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/vision/alvideorecorder.html

[17] SoftBank Robotics: ALTextToSpeech — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/audio/altexttospeech.html

[18] SoftBank Robotics: NAOqi Interaction engines — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/interaction/index_interaction.html

[19] SoftBank Robotics: ALAutonomousLife — Aldebaran 2.8.5.10 documentation [online]. 2018, [cit. 2018-12-22]. Dostupné z: http://doc.aldebaran.com/2-8/naoqi/interaction/autonomouslife.html

[20] LUNDH, Fredrik: What's Tkinter? [online]. 2018, [cit. 2018-12-23]. Dostupné z: http://effbot.org/tkinterbook/tkinter-whats-tkinter.htm

[21] CLARK, Alex and Contributors: Overview — Pillow (PIL Fork) 5.4.0 documentation [online]. 2019, [cit. 2019-01-04]. Dostupné z: https://pillow.readthedocs.io/en/stable/handbook/overview.html

[22] BELVAL, Edouard: pdf2image - A python module that wraps the pdftoppm utility to convert PDF to PIL Image object [online]. 2018, [cit. 2018-12-23]. Dostupné z: https://github.com/Belval/pdf2image

[23] The SciPy community: What is NumPy? [online]. 2018, [cit. 2018-12-23]. Dostupné z: https://docs.scipy.org/doc/numpy-1.15.0/user/whatisnumpy.html

[24] SWEIGART, Al: Introduction — PyAutoGUI 1.0.0 documentation [online]. 2017, [cit. 2018-12-23]. Dostupné z: https://pyautogui.readthedocs.io/en/latest/introduction.html

[25] OpenCV team: About - OpenCV library [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://opencv.org/about.html

[26] OpenCV team: Geometric Image Transformations — OpenCV 2.4.13.7 documentation [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html

[27] OpenCV team: Image Filtering — OpenCV 2.4.13.7 documentation [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html

[28] OpenCV team: Structural Analysis and Shape Descriptors — OpenCV 2.4.13.7 documentation [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[29] OpenCV team: Feature Detection — OpenCV 2.4.13.7 documentation [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html

[30] CANNY, John: A computational approach to edge detection [online]. 1986, [cit. 2018-12-26]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf

[31] OpenCV team: Miscellaneous Image Transformations — OpenCV 2.4.13.7 documentation [online]. 2018, [cit. 2018-12-26]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html

[32] OpenCV team: *OpenCV: Feature Matching [online]*. [cit. 2018-06-19]. Dostupné z: https://docs.opencv.org/3.3.0/dc/dc3/tutorial_py_matcher.html

[33] ROSEBROCK, Adrian: How-To: Python Compare Two Images [online]. 2014, [cit. 2018-06-19]. Dostupné z: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/

[34] BLAŽEK, Rudolf B., Roman KOTECKÝ, Daniel VAŠATA, Jitka HRABÁKOVÁ a Petr NOVÁK: BI-PST – Pravděpodobnost a statistika - Přednáška 5: Náhodné veličiny III [online]. 2017, [cit. 2018-01-09]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PST/_media/lectures/bi-pst-lec05-handout.pdf

[35] BLAŽEK, Rudolf B., Roman KOTECKÝ, Daniel VAŠATA, Jitka HRABÁKOVÁ a Petr NOVÁK: BI-PST – Pravděpodobnost a statistika - Přednáška 12: Lineární regrese [online]. 2017, [cit. 2018-01-09]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PST/_media/lectures/bi-pst-lec12-handout.pdf

Příloha **A**

# Seznam použitých zkratek

**ERS** Electronic Rolling Shutter, Elektronická rolovací závěrka

**FPS** Frames Per Second, Snímky za vteřinu

**LED** Light-Emitting Diode, Elektroluminiscenční dioda

**MSE** Mean Squared Error, Střední kvadratická chyba

**OpenCV** Open Source Computer Vision

**PIL** Python Imaging Library

**SoC** System-on-a-Chip

Příloha **B**

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `testing` ..... zdrojové kódy a data pro testování
      - `gui` ..... zdrojové kódy pro testování uživatelského rozhraní
      - `data` ..... testovací data
        - `photo` ..... snímky vyfocené robotem
        - `slides` ..... slajdy ve formátu .png
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Turcajova_Gabriela_2019.pdf` ..... text práce ve formátu PDF
  - `DP_Turcajova_Gabriela_2019_zadani.pdf` ... zadání práce ve formátu PDF