FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Robot NAO matematikem
Student:	Richard Stanko
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Proveďte rešerši dostupných algoritmů a knihoven pro čtení matematických výrazů z obrázku nebo z tahů na tabletu. Na základě rešerše vyberte nejspolehlivější algoritmus a jeho implementaci. Navrhněte a realizujte aplikaci pro robota NAO, ve které bude robot sledovat bílou tabuli, bude z ní číst napsané matematické vzorce a rovnice a s pomocí programu Wolfram Mathematika bude tyto rovnice vyhodnocovat a sdělovat slovně výsledky, případně u rovnic a odvození kontrolovat jejich správnost. Zaměřte se zejména na předzpracování dat z kamery pro vámi vybraný rozpoznávací algoritmus matematických vzorců a jeho návaznost na program Mathematica. Rozsah implementace upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

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

V Praze dne 9. února 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalárska práca

Robot NAO matematikem

Richard Stanko

Vedúci práce: Ing. Miroslav Skrbek Ph.D.

11. mája 2018

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb, autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k používaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo používať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, ale len pre nezárobkové účely. Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 11. mája 2018 .....

Odkaz na túto prácu

Stanko, Richard. *Robot NAO matematikem*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

STANKO, Richard. *Robot NAO matematikem.* [Bakalárska práca]. České vysoké učení technické v Praze. Fakulta informačních technologií; Katedra číslicového návrhu. Vedúci práce: Ing. Miroslav Skrbek Ph.D. Stupeň odbornej kvalifikácie: bakalár. Praha: FIT ČVUT, 2018. 53s.

Cieľom práce je navrhnúť aplikáciu na rozpoznávanie a výpočet matematických príkladov, ktorá beží na humanoidnom robotovi NAO od firmy Softbank Robotics. Práca je rozdelená do troch celkov. Prvý predstavuje robota NAO, jeho technickú a programovú výbavu. Ďalej poskytuje stručnú históriu technológie OCR, oboznamuje so samotnou technológiou a princípmi, ktoré používa. Popísané sú aj jednotlivé použité funkcie knižnice OpenCV a matematického aparátu, ktorý sa za nimi ukrýva. Uvedený je aj zoznam použitých nástrojov a podmienky, ktoré musia byť splnené, aby aplikácia fungovala. V závere prvej časti je vyobrazený a popísaný vývojový diagram návrhu aplikácie. Druhý celok je venovaný podrobnému popisu programu, ktorý je vyvinutý v grafickom prostredí Choregraphe a v jazyku Python. Hlavné bloky kódu sú podrobne popísané a vysvetlené sú aj vstavané funkcie a balíčky jazyka Python. Robot dokáže rozpoznať a vypočítať príklady od rovníc až po limity a integrály. Posledný celok je venovaný testovaniu a výsledkom. Aplikácia je riadená a ukončená slovnými príkazmi, teda nemusí byť opätovne spušťaná. Projekt je možné využiť na riešenie príkladov z tabule, papiera alebo na kontrolu správnosti. Do budúcnosti je možné rozšírenie na vysvetlenie konkrétnej problematiky, generovanie náhodných príkladov alebo použitie OCR časti na čítanie dokumentov, kníh a textov.

Klíčová slova Nao, OCR, OpenCV, Matematika, Wolfram Alpha, Výpočet príkladov

Abstract

STANKO, Richard. *Robot NAO as a mathematician.* [Bachelor's thesis]. Czech technical university in Prague. Faculty of Information Technology; Department of Digital Design. Supervisor: Ing. Miroslav Skrbek Ph.D. Professional qualification level: Bachelor degree. Prague: FIT CTU, 2018. 53p.

The aim of this thesis is to develop an application suited for equation recognition and solving. This application is designed to run on the humanoid robot NAO, developed by Softbank Robotics. The thesis is divided into three main sections. The first section introduces the robot, NAO his hardware and software specifications. Later on it deals with the history of OCR, the technology and its principles. OpenCV and its functions are described in detail including the mathematics involved. We give a list of tools used for this thesis and some requirements to make the application run properly. At the end of the first section a flowchart diagram is used to describe the design of the program. The second section is dedicated to a detailed description and explanation of the program which is developed in the Choregraphe suite with additional Python scripts. The main blocks of the code are explained in detail including some built-in functions and packages of the Python language. NAO is able to recognize and solve problems including basic equations, multiple variable equations, limits and integrals. The application is controled by words using speech recognition, so it does not have to be run over and over. The last section addresses testing and results. This project can be used to solve problems from the blackboard or written on a paper, or to check whether they are correct or not. For future development it is possible to add an explanation module, which would explain the topic in more detail, a random problem generator for the selected topic or use the OCR engine to create a document, book, text reader.

Keywords Nao, OCR, OpenCV, Math, Wolfram Alpha, Equation solving

Obsah

Zoznam obrázkov

Úvod

Technológia OCR je čím ďalej tým viac používaná na úradoch ako aj v osobnej sfére v podobe mobilných aplikácií. Hlavné využitie nachádza pri skenovaní dokumentov, rozpoznávaní jednotlivých častí na úradných spisoch a listoch, ako sú napríklad meno, priezvisko, rodné číslo. Ďalej ako identifikácia poznávacích značiek vozidiel na fotografiách zhotovených dopravými kamerami. V podobe mobilných aplikácii ich nachádzame ako matematické kalkulátory, konvertory dokumentov do digitálnej podoby alebo interaktívne prekladače do rôznych jazykov. Medzi takéto mobilné aplikácie patria napríklad Google Translate, Scan & Translate+, PDF Scanner+.

Roboty tiež nie sú žiadnou novinkou v domácnostiach za posledné roky. Do domácností asi najviac prenikli roboty of firmy iRobot. Prvým z nich je robot Roomba, ktorého hlavnou a jedinou úlohou je vysávanie. Vyznačuje sa charakteristickým okrúhlym tvarom. Podobný mu je robot Scooba, ktorý k vysávaniu pridáva aj umývanie podlahy. Ďalšie roboty od tejto firmy sú napríklad robot Looj, ConnectR a iné. Vtipným príkladom je robotický budík s kolieskami, ktorý sa po prvom zazvonení ukryje a keď ho chcete vypnúť musíte ho najprv nájsť. Humanoidní roboti sú samostatnou kategóriou, ktorú ale v domácnostiach stretneme asi len veľmi zriedka. Ako už ich názov naznačuje tvarom sa podobajú ľuďom. Najznámejšími príkladmi sú robot Atlas od firmy Boston Dynamics, ktorá je taktiež známa svojimi štvornohými robotmi, Asimo od Hondy a roboti Romeo a NAO od Aldebaran Robotics.

V našej práci sa budeme venovať už spomenutému robotovi NAO a jeho spojením s technológiou OCR na riešenie matematických príkladov. Robot NAO je vyvíjaný firmou Aldebaran Robotics (dnes už Softbank Robotics) a prvýkrát bol uvedený na trh v roku 2006. Jeho asi prvým úspechom bolo, že nahradil vtedajšieho robota Aibo v robotickej futbalovej lige[8]. Od jeho prvého vydania v roku 2006 prešiel už niekoľkými verziami a v súčasnej dobe je jeho najnovšou verziou Evolution V5. NAO je 58 cm vysoký robot disponujúci až 25 stupňami voľnosti. Robot je dostupný v dvoch základných farebných prevedeniach, a to v červenej a modrej s možnosťou vyhotovenia vo vlastnej

farbe. Možnosti typu projektov na NAO-vi sú obmedzené len kreativitou ich autorov.

Práca je rozdelená do troch celkov. Prvý celok s názvom Analýza a Návrh je venovaný popisu robota NAO, stručnej histórii a charakteristike technológie OCR, nástrojom použitým na vývoj a v závere je načrtnutý návrh aplikácie na vývojovom diagrame. Druhý celok s názvom Realizácia pojednáva o konkrétnej implementácii programu v jazyku Python, kde podrobne popisujeme jednotlivé časti kódu a vstavané bloky prostredia Choreographe. Posledná kapitola je venovaná testovaniu, ukážke jednotlivých testovaných príkladov a výsledkov.

Kapitola 1

Cieľ práce

Cieľom našej práce je navrhnúť aplikáciu na robotovi NAO, ktorá mu umožní počítať, vyhodnocovať alebo kontrolovať rôzne matematické formule napísané na tabuli, papieri alebo na čomkoľvek, čo si robot NAO dokáže odfotografovať. Takáto aplikácia by umožnila rýchlu kontrolu vypočítaného príkladu a pomoc s neznámymi či náročnými matematickými úlohami bez prítomnosti akéhokoľvek pedagogického personálu, stačí zapnúť robota, pustiť aplikáciu a počítať spolu s NAO-m. Využitie je možné, či už vo vzdelávacích zariadeniach, kde robot okamžite dokáže skontrolovať a príadne opraviť odpoveď študenta alebo v domácnosti pri plnení domácich úloh, či samoštúdiu. Rozsah a náročnosť vypočítateľných príkladov sa pohybuje od základnej aritmetiky, cez rovnice až po diferenciálny a integrálny počet. Ponúka sa nám možnost pridať do repertoára aplikácie aj teoretické poznatky k daným témam, teda spraviť z robota nie len matematika, ale taktiež ako prednášajúceho matematiky. Nevýhody a úskalia tejto problematiky sú slabá interaktivita medzi NAO-m a študentom a množstvo poznatkov, ktoré by robot musel mať uložené. Je to však jedna z možností ako projekt do budúcna rozšíriť alebo upraviť.

Nemenej dôležitým cieľom je zoznámenie sa a oboznámenie čitateľa s technológiou OCR, ktorá má široké využitie v rôznych sférach ako napríklad preklad textu z obrázku v reálnom čase, rozpoznanie textu, matematických symbolov a práca s nimi. OCR čím ďalej tým viac zasahuje do každodenného života. Najbežnejšie sa s touto technológiou stretneme pri automatickom skenovaní dokumentov a ich konverziou do konkrétneho formátu, rozpoznávaním poznávacích značiek aút alebo ako pomoc zrakovo postihnutým. Jedná sa väčšinou o offline proces, teda dáta sa nemenia v reálnom čase, technológia zaoberajúca sa týmto typom sa nazýva on-line character recognition alebo dynamic character recognition. Je na mieste zmieniť, že OCR sa venuje tlačiteľným znakom, rukopisom sa zaoberá technológia inteligent character recognition(ICR). S rozpoznávaním textu a jednotlivých znakov ide ruka v ruke s úpravou daného obrázku do stavu, kedy sú znaky v dobrom kontraste s pozadím a majú jasné kontúry. Na túto úlohu využijeme voľne dostupnú knižnicu OpenCV pre

jazyk Python, knižnica je dostupná aj pre iné jazyky. Knižnicu samotnú, jej algoritmy, ktoré použijeme si predstavíme bližšie v sekcii analýza a návrh. Cieľom predstavenia OpenCV je pochopenie fungovania funkcií a matematiky ukrývajúcou sa za jednotlivými algoritmami ako sú napríklad konvolučné matice, rôzne transformácie a iné.

Kapitola 2

Analýza a návrh

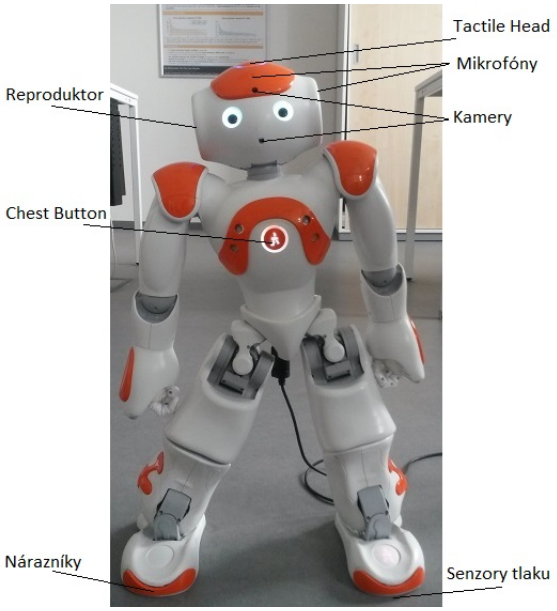
2.1 Analýza

V tejto časti bakalárskej práce sa zamierame na analýzu robota NAO, technológie OCR a knižnice na prácu s obrazom OpenCV. V sekcii NAO si ukážeme, že NAO je vhodný na takýto typ úlohy, t.j. ako "matematik" a pomocník pri výpočtoch. Pomôže mu pri tom jednak vstavané API v jazyku Python a balíčky OpenCV, ktoré sa do Pythonu jednoducho doinštalujú. Ďalej v sekcii OCR si povieme niečo o histórii tejto technológie, technológii samotnej a aplikácii ktorá nám poskytne samotný rozpoznávací aparát pre našu prácu. Posledná sekcia je venovaná knižnici OpenCV, ktorú použijeme na úpravu a prípravu obrázkov pred ich odoslaním na rozpoznanie. Sú v nej uvedené a popísane niektoré postupy či algoritmy použité v našom projekte.

2.1.1 Robot NAO

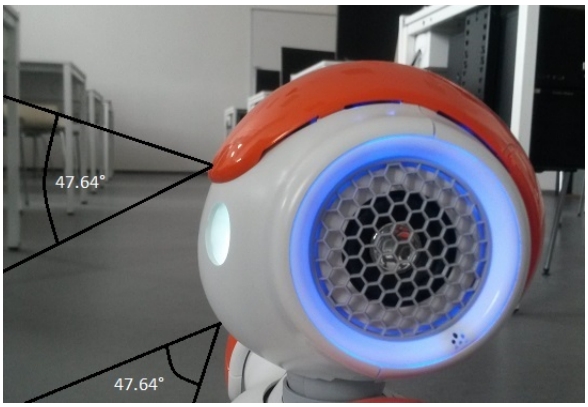
2.1.1.1 Technická špecifikácia

Robot NAO je humanoidný robot vyvinutý firmou Aldebaraan(Softbank) Robotics v roku 2006. V priebehu rokov si prešiel už niekoľkými verziami, v súčasnosti sa nachádza vo verzii 5, my budeme používať verziu v4. NAO je 58 centimetrov vysoký a disponuje 25 stupňami voľnosti a to vďaka motorom ktoré mu umožnujú pohybovať nohami, kolenami, členkami, rukami, zápästiami a hlavou. Toto mu umožnuje prispôsobovať sa okoliu, udržovať stabilitu, alebo rozpoznať, či práve sedí, leží. Čo sa našej práce týka využijeme jeho štyri mikrofóny a reproduktory na komunikáciu s užívateľom, ale zaujímajú nás hlavne kamery, ktoré sú na robotovi dve a obe sú umiestnené v hlave robota. Kamery majú rozlíšenie 1280x960 pri 30 obrazoch za sekundu a dokážu identifikovať objekty v diaľke. My budeme používať tú kameru, ktorá má v zornom poli celý náš objekt, v našom prípade matematickú formulu, ktorej upravenú fotografiu potom pošleme na rozpoznanie a následný výpočet.



Obr. 2.1: Rozmiestnenie jednotlivých senzorov a pohybových nástrojov

Ako je vyobrazené na obrázku 2.1, NAO disponuje dotykovými senzormi umiestnenými na hlave a rukách, senzormi tlaku a nárazníkmi na nohách a ďalej v každom kĺbe senzorom teploty, stuhnutosti a polohy. NAO je postavený na architektúre Intel Atom s procesorom Z530 o frekvencii 1.6 GHz, pamäťou RAM o veľkosti 1 GB, flash pamäťou o veľkosti 2 GB a možnosťou pridať 8 GB micro SDHC kartu. Toto vybavenie nám bude stačiť na uskutočnovanie operácií s obrázkom a jeho uloženie do pamäte. Je taktiež vybavený bezdrôtovým prístupom na internet, čo nám umožní odoslanie a výpočet na vzdialenom serveri a následný príjem výsledku[6].



Obr. 2.2: Rozmiestnenie kamier a ich zorné polia

2.1.1.2 Programové vybavenie

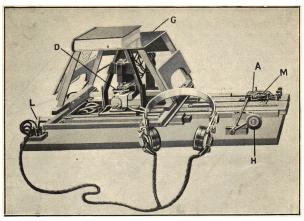
NAO disponuje operačným systémom NAOqi OS, čo je distribúcia GNU/Linux založená na Gentoo. Je to distribúcia vstavaného typu vyvinutá špecificky pre potreby robota. Bežia na ňom rady programov a knižníc medzi nimi aj všetky potrebné pre NAOqi[6]. K systému je možné sa pripojiť vzdialene prostredníctvom SSH a pristupovať k súborom cez FTP.

Aplikácie na robota navrhujeme v programe Choregraphe, čo je nástroj na návrh rôznych choreografií, chovaní a interakcií s užívateľom. Tento program umožnuje tvoriť aplikácie prostredníctvom pridávania modulov reprezentujúcich jednotlivé senzory, predpripravené animácie a pohyby typu: postav sa, sadni si, choď do predu a pod. Môžeme si samozrejme pridať aj vlastný modul, do ktorého vložíme vlastný pythonovský kód. Robot podporuje jazyky Python, C++, Java a Javascript, pričom Python má podporu ako v Choreograph, tak priamo na robotovi cez pythonovský balíček NAOqi. Jazyk C++ je použitelný prostredníctvom vývojových prostredí Visual Studio, Eclipse, QtCreator a XCode, do ktorých je nutné doinštalovať NAOqi framework, ktorý potom komunikuje s robotom cez sieť.

2.1.2 OCR

2.1.2.1 História

Na úvod si povieme niečo z histórie technológie OCR. Za touto skratkou stojí pojem Optical Character Recognition, čiže optické rozpoznávanie znakov. Počiatky technológie OCR môžeme datovať do roku 1914, kedy Emanuel Goldeberg, izraelský fyzik a vynálezca, vyvinul stroj, ktorý čítal znaky a konvertoval ich na štandardný telegrafický kód. Súčasne s ním Edmund Fournier d'Albe, vyvinul Optophone, ručný skener, ktorý čítal znaky a vytváral tóny, ktoré korešpondovali s konkrétnými naskenovanými písmenami[1]. V roku 1930 Emanuel Golberg opäť zasiahol do histórie OCR a to prístrojom, ktorý nazval "Statistical Machine". Tento stroj prehľadával archívy mikrofilmov, na čo používal systém rozpoznávania optického kódu. Goldberg obrdžal v roku 1931 patent a tento patent neskôr odkúpila firma IBM. "Statistical machine" bol zjednodušený a 40 násobne zrychlený profesorom z MIT Vannerbarom Bushom v roku 1938. Presunieme sa o pár rokov dopredu do roku 1951, kedy americkí kryptoanalisti David H.Shepard a Harvey Cook Jr. vytvorili takzvané "Gizmo", stroj, ktorý dokázal nahlas čítať napísane znaky a interpretovať Morseho kód[1]. Na záver preskočíme do roku 2000, kedy je technológia OCR sprístupnená ako online služba WebOCR pre cloud computing ako aj pre mobilné aplikácie. V roku 2005 je spoločnosťou Hewlett-Packard vydaný voľne dostupný cross-platformový engine OCR s názvom Tesseract. Od roku 2006 je vývoj Tesseractu sponzorovaný spoločnosťou Google a bol považovaný za najpresnejší voľne dostupný OCR engine. V súčasnosti je už dostupná verzia 4, tohoto nástroja a podporuje ropoznávanie vo vyše 100 jazykoch s možnosťou učenia sa novej znakovej sady[5].



Obr. 2.3: Optophone. Obrázok je naskenovaný z časopisu *Vetenskapen och livet* z roku 1922. Odkaz: `http://runeberg.org/vetlivet/1922/0113.html`

2.1.2.2 Technológia

V súčasnosti sa OCR vyvinul do rôznych špecifických aplikácií ako napríklad rozpoznávanie dokladov, faktúr, šekov alebo dokumentov. Používajú sa hlavne na nasledujúce úkony:

- Automatické rozpoznanie ŠPZ
- Automatizácia spracovania formulárov
- Automatické extrahovanie poistných údajov
- Extrahovanie osobných údajov do kontaktov
- Konverzia rukopisu na ovládanie počítača

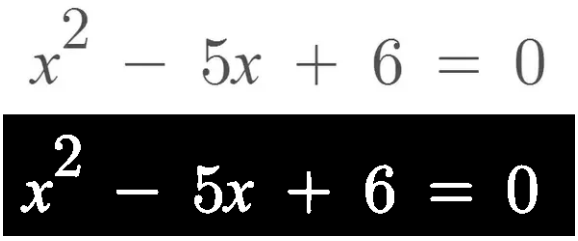
- Asistencia zrakovo postihnutým

Rozlišujeme niekoľko druhov tejto technológie a to podľa typu znakov, ktoré rozpoznáva. Najčastejšie hovoríme o OCR, ktorá rozpoznáva tlačené znaky alebo slová a to prevažne offline, teda nie v reálnom čase. Na rozpoznávanie v reálnom čase slúži technológia Dynamic Character Recognition alebo Online Character Recognition. Ďalším typom sú takzvané ICR a IWR, teda Intelligent Character, resp. Word Recognition. Tieto technológie rozpoznávajú ručne písané znaky a slová. Väčšina ICR enginov má neurónové siete, ktoré sa dokážu samostatne učiť a aktualizujú databázu znakov vždy, keď rozpoznajú nový typ rukopisu.

Pred samotným rozpoznávacím procesom sa musí obrázok upraviť do podoby, aby boli znaky v dobrom kontraste s pozadím, vyhladené, vyrovnané. Hovoríme o takzvanom pre-procesingu[2]. Ďalej sa odstraňujú neznakové údaje, škrvny a nežiadúce bodky a je tu aj možnosť, že sa vytvorí viacero obrázkov z pôvodného, každý reprezentujúci jeden znak. Obrázok je konvertovaný z modelov RGB, CMYK a iných do modelu Grayscale, je teda zbavený farby a každý pixel nesie informáciu len o svojej intenzite svetla. Na tieto techniky sa využívajú knižnice, ktoré pracujú s obrazom a videom, v našom prípade použijeme voľne dostupnú knižnicu OpenCV, ktorú si popíšeme neskôr.

Po príprave obrázku prichádza na rad samotné rozpoznávanie znakov. Povieme si o dvoch základných algoritmoch, ktoré vytvárajú rebríček kandidátov na príslušný rozopoznávaný znak. Prvým z nich je takzvaný Matrix matching, teda matice jednotlivých obrázkov sa porovnávajú na úrovni pixelov. Táto technika je najúčinnejšia pri tlačenom texte a bola použitá v prvotných OCR strojoch. Úskaliami tejto metódy je vysoký počet pixelov a teda aj vstupných dát na klasifikáciu. Použitie rôznych typov písma môže viesť k nesprávnym klasifikáciam.

Druhým algoritmom je extrakcia určitých smerodajných vlastností samotného znaku. Obrázok je najprv upravený takzvaným thresholdingom do binárnej podoby a v tomto už binárnom obrázku sú vyhľadané spojité komponenty. Tie sú potom rozdelené na skupiny podľa vlastností ako sú čiary, uzavreté krivky, smery čiar a a ich prieniky. Tieto vlastnosti sú potom porovnané s príslušnými znakmi a je vybratá najbližšia zhoda na základe nejakého algoritmu akým je napríklad algoritmus k-najbližších susedov. Vo väčšine prípadov sú tieto vlastnosti predané neurónovej sieti, ktorá na základe všetkých vstupných vlastností klasifikuje vstup do niektorých výstupných kategórii znakov z rozpoznávanej abecedy. Podobne ako pri rozpoznávaní reči sa zvyknú používať skryté Markovove modely, ktoré pomáhajú pochopiť vstup ako celok[2]. Tieto modely sú najprv inicializované takzvanou tréningovou sadou (training dataset) a metódou učenia sa s učiteľom (supervised learning). Pri tejto metóde je modelu prezentovaný vstup a požadovaný výstup. Problémom pri tomto type učenia je takzvané preučenie, kedy model neni schopný dobre zovšeobecňovať a klasifikovať vstupné data.



Obr. 2.4: Ukážka spracovania obrázku, kde výsledkom sú biele znaky na čiernom pozadí, ktoré sú ľahšie rozpoznateľné

Na záver sa rozpoznané znaky a slová zobrazia buď ako obyčajný text alebo sofistikovanejšie programy dokážu zachovať pôvodný obrázok a doplniť do neho rozpoznaný text. Efektivitu rozpoznania dokážu zvýšiť napríklad pridanie slovníka alebo implementácia gramatiky príslušného jazyka. Prítomnosť slovníka presne určuje slová, ktoré sa v texte vyskytujú a teda môžu byť aj rozpoznané. Väčšinou sa jedná o slovnú zásobu alebo znakovú sadu príslušného jazyka. Minimalizuje sa tak rozpoznanie nezmyslených slov a znakov.

2.1.3 OpenCV

2.1.3.1 O knižnici

Za skratkou OpenCV sa skrýva názov Open Source Computer Vision Library, čiže voľne dostupná knižnica pre účely počítačového videnia a strojového učenia. Táto knižnica obsahuje viac než 2500 optimalizovaných algoritmov pre prácu s obrazom, videom a pre strojové učenie. Algoritmy môžu byť využité na rozpoznávanie tvárí, rôznych objektov a tvarov, extrakciu 3D objektu zo scény, sledovanie pohybujúcich sa ľudí a objektov na videu, ba dokonca dokážu klasifikovať jednotlivé činnosti, ktoré tieto objekty vykonávajú. Obsahuje aj algoritmy potrebné na úpravu obrazu ako sú konverzie farebných priestorov, detekcie hrán, zaostrovanie a rozostrovanie a iné[7].

2.1.3.2 Popis použitých funkcií

V tejto sekcii si popíšeme použité funkcie na predprípravu obrázku a matematiku ukrytú za jednotlivými funkciami.

```
cvtColor(InputArray src, OutputArray dst, int code, int
    dstCn=0)
```

Funkcia cvtColor mení farebný priestor obrázku *input* na priestor reprezentovaný premennou *code* a uloží ju do *output*. Premenná *dstCn* slúži na určenie počtu kanálov výstupu. Kedže v našej práci používame výhradne konverziu do grayscal-u popíšeme si len túto časť. Intenzitu Y spočítame ako váženú sumu jednotlivých zložiek RGB a to nasledovne: $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$.

```
adaptiveThreshold(InputArray src, OutputArray dst,
    double maxValue, int adaptiveMethod, int
    thresholdType, int blockSize, double C)
```

Ďalšou funckiou je *adaptiveThreshold*, ktorá mení obrázok *src* z grayscalu do binárneho obrázku a ukladá ho do *dst*. *maxValue* je hodnota, ktorá sa priradí pixelu ak je vyšší alebo rovný ako prahová hodnota, *adaptiveMethod* je kód metódy, ktorá sa použije na získanie príslušnej prahovej hodnoty pre jednotlivé pixely za pomoci okolia príslušného pixelu, ktorého veľkosť je udaná ako $blockSize \times blockSize$. Z tohoto bloku pixelov sa počíta priemer resp. vážený priemer. Premenná *thresholdType* určuje, či sa jedná o binárnu alebo inverznú konverziu. Konštanta C je odčítaná od priemeru, resp. váženého priemeru. Pri použití binárnej koverzie sa každému pixelu priradí *maxValue* ak je jeho hodnota vyššia ako odpovedajúca prahová hodnota alebo 0 v opačnom prípade. V inverznej konverzii sú tieto priradenia opačné. OpenCV podporuje dve metódy výpočtu a to metódu priemeru a Gaussovskú metódu. V metóde priemeru sa hodnota spočíta ako priemer príslušného blocku pixelov a odčíta sa od neho konštanta C. Ak použijeme Gaussovskú metódu je použitý vážený priemer za použitia Gaussovského okna, ktoré odpovedá veľkosti bloku, pre okno tohoto bloku sa použije štandardná odchýlka.

```
GaussianBlur(InputArray src, OutputArray dst, Size ksize
    , double sigmaX, double sigmaY=0, int borderType=
    BORDER_DEFAULT)
```

Podobne ako v predchádzajúcom príklade táto funkcia mení obrázok *src* a to tak, že ho rozostruje. Na to použije Gaussovský kernel o veľkosti $ksize \times ksize$, ktorý je potom použitý na konvolúciu s okolím každého pixelu. Môžeme to chápať tak, že príslušný pixel a jeho okolie "prekryjeme" naším kernelom a jednotlivé prekrývajúce sa hodnoty spolu vynásobíme a sčítame, čím dostaneme novú hodnotu pixelu. Hodnoty *sigmaX* a *sigmaY* nám určujú štandardnú odchýlku v smere X-ovej resp.Y-ovej osy a *borderType* nám vraví ako sa budú extrapolovať pixely na okrajoch obrázka.

```
findContours(InputOutputArray image, OutputArrayOfArrays
    contours, OutputArray hierarchy, int mode, int
    method, Point offset=Point())
```

Funkcia, ktorá nájde kontúry binárneho obrázku *image* a uloží ich do *contours*. Pole *hierarchy* je naplnené hierarchiou v závislosti na zvolenom móde hodnotou *mode*. Premenná *method* určuje aká metóda sa použije na získanie kontúr. Medzi tieto metódy patrí zisk všetkých kontúr bez kompresie, s kompresiou alebo za použitia Teh-Chinovho algoritmu [3]. Pomocou algoritmu [4] sú tieto kontúry extrahované a uložené. Tieto kontúry sú užitočným nástrojom na analýzu a rozpoznanie tvarov, objektov a je možné vykresliť ich funkciou *drawContours*, ktorá podľa príslušnej hierarchie vykreslí buď len vonkajšie, vnútorné alebo všetky kontúry.

2.1.4 JSON

JSON alebo JavaScript Object Notation je spôsob uloženia dát, ktorý nie je závislý na cieľovej platforme a slúži prevažne k výmene dát. Jeho hlavnou výhodou je ľahká čitateľnosť pre ľudí a jednoduché spracovanie a generovanie pre stroje. Vyznačuje sa ukladaním dát vo formáte "meno": hodnota oddelených čiarkami a uzavretých medzi zložené zátvorky. Hodnoty polí sú uzavreté medzi hranaté zátvorky a oddelené čiarkami. V jazyku Python budeme s týmto formátom pracovať prostredníctvom balíčka *json*.

2.1.5 Prerekvizity

Aby bol zaručený bezchybný chod aplikácie je potrebné splniť niektoré prerekvizity pre jazyk Python, rozpoznávač rovníc a aj pre Wolfram Alpha. Prvou podmienkou je pripojenie k internetu.

2.1.5.1 Python

V práci je využitá verzia 2.7 jazyka Python a budeme potrebovať nasledujúce balíčky:

- request - na odoslanie požiadavku rozpoznávaču a jeho prijatie
- json - na obsluhu JSON objektov
- base64 - na kódovanie reťazcov z binárnej do textovej podoby
- re - na prácu s regulárnymi výrazmi
- cv2 - knižnica OpenCV, na úpravu obrázkov
- numpy - obsahuje funkcie lineárnej algebry a rôzne transformácie
- naoqi - poskytuje rozhranie na komunikáciu s príslušenstvom robota

2.1.5.2 Rozpoznávanie

Na rozpoznávanie formúl v našej aplikácii použijeme program Mathpix (dostupný na `www.mathpix.com`), ktorý poskytuje svoje API aj pre iné aplikácie. Mathpix je desktopová aplikácia, ktorá konvertuje matematické formule do formátu LaTeX. Aby sme toto API mohli používať, je potrebné založenie si účtu, prostredníctvom ktorého užívateľ obdrží dvojicu kľúčov, ktoré sú potrebné na autentizáciu. O rozpoznanie formule sa žiada prostredníctvom požiadavku HTTP POST, ktorý obsahuje názov obrázku, dvojicu autentizačných kľúčov a JSON objekt kam sa májú rozpoznané data uložiť.

2.1.5.3 Výpočet

Výpočet v našej aplikácii zaisťuje internetová stránka Wolfram Alpha (dostupná na `www.wolframalpha.com`). Táto stránka založená na jazyku Wolfram Mathematica poskytuje rôznorodé informácie od matematiky cez informatiku, geografiu až po nutričné hodnoty jedál. Pri výpočte matematiky nám vypočíta nie len výsledok ale poskytne aj rôzne interpretácie zadanej požiadavky ako sú napríklad grafy, rady, transformácie a iné. Rovnako ako v predchádzajúcej sekcii, vyžaduje registráciu, ktorá užívateľovi priradí unikátny autentizačný kľúč. Týmto kľúčom sú identifikované požiadavky odoslané na servery Wolfram Alpha. Výsledok je doručený vo formáte XML alebo JSON podľa preferencie.

2.2 Podobné projekty

V tejto sekcii si v krátkosti povieme o projektoch na robotovi NAO, ktoré súvisia s matematikou alebo OCR.

2.2.1 Robot NAO ako učiteľ

V tomto projekte z univerzity v Plymouth, Spojené Kráľovstvo sa autori zaoberajú štúdiou vplyvu robota-učiteľa na výkony študentov. Jedná sa o mladistvých žiakov vo veku od 11-12 rokov. Autori uvádzajú, že dosiahnutie lepších výsledkov na robotovi je z toho dôvodu, že žiaci považujú robota za fascinujúceho a zábavného. Robot bol v tomto prípade naprogramovaný ako

interaktívny učiteľ, ktorý pomocou rečových výplní, prestávok, gest a sledovaní tváre napodobňuje ľudské chovanie. Dokáže vysvetliť problematiku na rôznych príkladoch, vysvetliť a poskytnúť pomoc keď je potreba.[9]

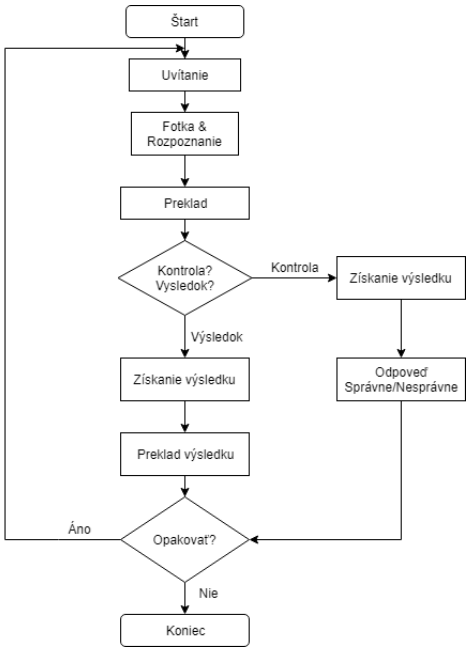
2.2.2 NAO a OCR

Projektov súvisiacich s OCR je mnoho. Prevažná časť sa venuje rozpoznávaniu znakov v rôznych jazykoch za pomoci už vyššie spomenutej knižnice Tesseract v spolupráci s OpenCV. Výsledky z týchto rozpoznaní sú potom predané iným funkciám ako napríklad písanie, hovorenie, a iné. Uvedieme si dva príklady:

- **Optické a rukou písané rozpoznanie číslic pomocou neurónovej siete na robotovi NAO**. Autori používajú neurónovú sieť na detekciu extrakciu znakov z obrázkov. Táto sieť je najprv trénovaná pomocou tréningovej sady dát. Odkaz: https://www.researchgate.net/publication/31-1589353_Optical_and_handwritten_digit_recognition_by_using_neural_network_on_NAO_humanoid_robot.
- **larics/nao-ocr** je projekt zo stránky GitHub, ktorý pomocou knižníc Tesseract a Leptonica implementuje rozpoznávanie znakov na robotovi. Je dostupný ako vzdialený alebo lokálny modul spustiteľný na robotovi. Odkaz: `https://github.com/larics/nao-ocr`

2.3 Návrh

Návrh aplikácie popisuje nasledujúci vývojový diagram. Aplikácia je vyhotovená v grafickom návrhovom prostredí Choreographe, v ktorom využijeme predpripravené bloky na hovorenie, rozpoznávanie reči a vyhotovenie fotografie. Ďalšia potrebná funkcionalita je implementovaná v jazyku Python a komunikuje so vstavanými blokmi. Medzi kľúčové vlastnosti patria: riešenie rovníc jednej a viacerých neznámych, výpočet určitých a neurčitých integrálov, derivácií a limít. Ďalej riešenie logaritmov a matíc. Každá táto operácia má v jazyku LaTeXunikátnu reprezentáciu, čo nám umožňuje jednoduché rozpoznanie a preklad danej operácie. Ďalej musí jasne prezentovať, čo má vypočítať a aký je výsledok daného problému, ďalej aby užívateľovi ponúkol možnosť opätovného výpočtu bez potreby znova nahrávať a spúšťať program. Prvý problém vyriešime tak, že NAO bude prekladať matematiku do českého jazyka a vždy po rozpoznaní formule alebo výsledku povie o aký príklad sa jedná. Druhá vlastnosť je zaistená jednoduchou otázkou na konci programu, či chce užívateľ pokračovať alebo ukončiť program.



Obr. 2.5: Vývojový diagram aplikácie

2.3.1 Popis blokov

V bloku *Uvítanie* robot privíta užívateľa a požiada ho, aby ho umiestnil pred matematický príklad, ktorý je treba vypočítať alebo skontrolovať. To znamená, že kamera umiestnená na hlave by mala byť v rovnakej alebo približnej

výške ako je rozpoznávaná matematická formula. Pokračovanie je signalizované pohladením robota po senzoroch na hlave.

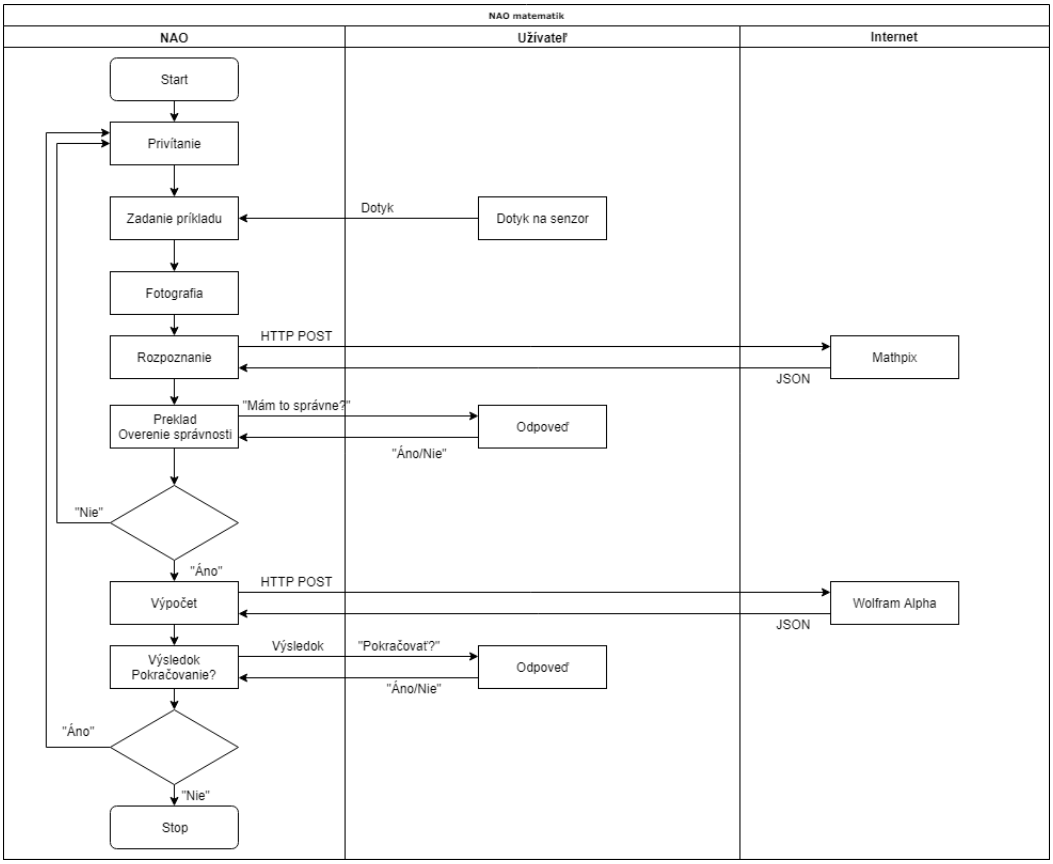
Blok *Fotka & Rozpoznanie* má na starosti vyhotovenie fotografie, jej úpravu pomocou funkcií OpenCV do inverznej binárnej podoby. Následne upravenú fotografiu odošle na rozpoznanie aplikácií Mathpix, ktorá rozpoznaný obrázok vráti vo formáte JSON. Položka "latex" v prijatom objekte obsahuje preloženú matematickú formulu do latex-u.

Získaný latex-ový reťazec je v bloku *Preklad* preložený do českého jazyka a potom je odoslaný do vstavaného bloku Text-to-Speech, kde NAO túto matematickú formulu vysloví. Preklad je realizovaný pomocou regulárnych výrazov, ktoré sa aplikujú na dodaný latex-ový reťazec. Následne sa robot opýta, či chce užívateľ zadaný príklad skontrolovať alebo vypočítať. Na túto otázku sa očakáva slovná odpoveď. Rozpoznanie odpovede sa prevedie vo vstavanom bloku Voice Recognition.

O výpočet samotnej formule sa stará blok *Získanie výsledku*. Výsledok je získaný prostredníctvom API nástroja Wolfram Alpha. Rovnako ako internetová stránka aj API vracia výsledok vo viacerých podobách. Nás bude zaujímať len plain-text forma výsledku. Výsledok bude následne opäť preložený do českého jazyka a robot ju vysloví. Na záver sa NAO opýta užívateľa, či chce opakovať výpočet alebo sa má ukončiť.

2.3.2 Komunikácia

Komunikácia vrámci našej aplikácie je vyobrazená na nasledujúcom obrázku. Medzi jednotlivými blokmi v robotovi sú predávané reťazce alebo len signály na prechod do ďalšieho bloku. Komunikáciu môžeme rozdeliť na tri hlavné oblasti a to oblasť robota NAO, užívateľa a internetových aplikácií.



Obr. 2.6: Vývojový diagram komunikácie vrámci aplikácie.

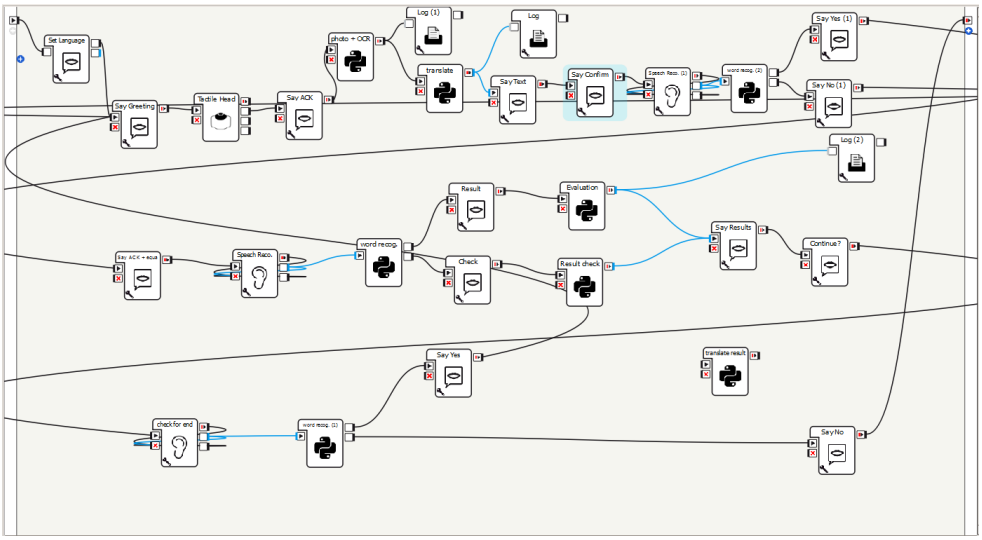
Diagram znázorňuje chod a komunikáciu v priebehu aplikácie. Medzi užívateľom a robotom prebieha komunikácia slovne. Rozumie sa tým predvolená množina z českého jazyka. Konkrétne sú to slová: Äno", "Ne". Týmito slovami užívateľ riadi beh programu a dáva robotovi znamenie na pokračovanie alebo ukončenie chodu programu, poprípade na signalizáciu, že daný príklad bol zle rozpoznaný. Požiadavky na internetové aplikácie v našom programe sú zaistené cez HTTP POST požiadavky poskytované balíčkom *requests*. V prípade odoslania upravenej fotografie na rozpoznanie, sa do hlavičky požiadavky špecifikuje aplikačný kľúč, email a návratový formát. Mathpix potom odpovedá objektom vo formáte JSON, ktorý popisuje jednotlivé vlastnosti rozpoznaného obrázku. Tento objekt obsahuje informácie o presnosti rozpoznania, pozície, či obrázok obsahuje graf, tabuľku, geometriu a samozrejme samotný rozpoznaný text. V druhom prípade, kedy sa odosiela požiadavka na výpočet, sa všetky informácie predávajú v tele požiadavky vo formáte "názov-"hodnota", oddelené znakom '&'. Pre Wolfram Alpha je nutné špecifikovať návratový formát ako JSON, pretože implicitne je nastavený na XML. Výsledok je totožný ako

na internetovej stránke Wolfram Alpha a je formátovaný do takzvaných **podov**. Každý **pod** má svoj názov, ak to charakter príkladu povoľuje tak graf a zoznam **sub-podov**. Práve v týchto **sub-podoch** je uložený aj výsledok v podobe plaintextu. Nie je predom známe, v ktorom **pode** je umiestnený výsledok, preto prejdeme postupne všetky a hľadáme prvý výskyt znaku '='. Niektoré úlohy majú výsledok v **pode** s názvom Result. Rovnice majú svoj výsledok uložené v **pode** s názvom Solution.

Kapitola 3

Realizácia

V tejto kapitole pojednávame o samotnej realizácii blokov popísaných v časti návrh. Popísané sú samostatné časti zdrojového kódu, použité funkcie z rôznych balíčkov jazyka Python. Pri každom bloku je uvedený zdrojový kód alebo jeho podstatná časť, pokiaľ je v rozumnom rozsahu. Vstavané funkcie jazyku Python z použitých balíčkov sú popísané pri ich prvom výskyte v zdrojovom kóde. V popise blokov sa vyskytuje pojem vstavaný balíček, takto označujeme predpripravený blok kódu, ktorý je dostupný v prostredí Choreographe hneď po jeho spustení. Na obrázku môžeme vidieť konečné schéma zapojenia



Obr. 3.1: Schéma zapojenia jednotlivých blokov v grafickom prostredí Choregraphe

blokov. Vlastné bloky sú zobrazené ikonou jazyka Python. Ikona ústa reprezentuje bloky Text-to-Speech, ktoré umožňujú robotovi rozprávať. Do bloku sa

doplní text v českom jazyku, ktorý sa má vysloviť. Bloky so značkou ucha sú určené na Speech Recognition, teda rozpoznávanie reči. Tieto bloky umožňujú komunikáciu s robotom a jeho riadenie prostredníctvom hovorených príkazov. V hornej ľavej časti môžeme vidieť blok Tactile Head reprezentujúci senzory hlavy, ktorý sa aktivuje pri dotyku jedného z troch senzorov umiestnených na hlave. Zvyšné bloky so značkou papiera sú takzvané Logy a sú určené na zachytávanie ladiacich výpisov z blokov.

3.1 Realizácia jednotlivých blokov

3.1.1 Blok Uvítanie

Uvítací blok je z hľadiska realizácie rozdelený do dvoch častí. Prvá časť je samotné uvítanie užívateľa, v ktorom sa robot predstaví a v stručnosti povie o čom naša aplikácia je a čo robí. V druhej časti požiada užívateľa, aby ho umiestnil pred matematickú formulu, ktorú ma vyriešiť a pohladil ho po hlave, čím sa signalizuje prechod do fotografickej a rozpoznávacej časti. Oba bloky sú realizované vstavanými blokmi Text-to-Speech, ktorým predchádza blok nastavenia jazyka, ktorý nastaví jazyk pre celú aplikáciu na český. Po detekcii dotyku na senzoroch, ktoré sú umiestnené vo vrchnej časti hlavy vyobrazených na Obr. 2.1, nastáva odoslanie signálu do nasledujúceho bloku a jeho spustenie.

3.1.2 Blok Fotografia & Rozpoznanie

Po aktivácii tohoto bloku sa vytvára spojenie s modulom kamery robota cez volanie funkcie *ALProxy*. Na použitie týchto funkcií potrebujeme importovať balíček *naoqi*.

```
cam = ALProxy("ALPhotoCapture", "10.10.48.252", "9559")
```

Funkcia požaduje tri argumenty. Prvým je názov modulu, ku ktorému sa chceme pripojiť, v našom prípade sa jedná o modul *ALPhotoCapture*. Tento modul poskytuje pre programátora funkcie oboch dostupných kamier. Druhým argumentom je IP adresa, na ktorej požadovaný modul beží a posledný argument je port. Následne cez volanie funkcie *takePicture* vytvoreného objektu *cam* odfotografujeme matematickú formulu.

```
cam.takePicture(folderPath, fileName)
```

Argumenty *folderPath* a *fileName* sú reťazce, ktoré určujú cestu kam a pod akým názvom sa má obrázok uložiť. Možné je doložiť tretí, nepovinný argument, ktorý keď je nastavený na hodnotu True, prepíše súbor s rovnakým názvom. V prípade ak je hodnota tohoto argumentu False a exstuje súbor s rovnakým menom, je vyhodená výnimka. Návratovou hodnotou je pole o jednom prvku obsahujúce meno uloženej fotografie.

Nasledujú úpravy obrázku prostredníctvom funkcii z knižnice OpenCV. Na ich sprístupnenie je potrebné importovať balíček *cv2*. Začneme načítaním obrázku do premenej *img*.

```
img = cv2.imread(filename, 0)
```

Prvý argument je meno obrázku, ktorý sa má otvoriť, druhý argument určuje spôsob. Spôsoby načítania obrázku sú: farebný obrázok, je to implicitné nastavenie a táto možnosť načíta farebný obrázok, ale ignoruje alfa kanál, druhou možnosťou je čiernobiely spôsob a na záver nezmenený, ktorý načíta obrázok tak ako je. V našom prípade použijeme čiernobiely spôsob, ktorý sa najlepšie hodí na ďalšie úkony. Konverzia do binárnej podoby je zaistená funkciou *adaptiveThreshold()*, ktorá je podrobnejšie popísaná v sekcii Analýza. Ako spôsob konverzie je použitý Gaussovský priemer.

```
th = cv2.adaptiveThreshold(img, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 3, 2)
```

Úprava obrázku je dokončená invertovaním čiernej a bielej farby, čím získame biely text na čiernom pozadí, ktorý je ľahšie rozpoznateľný. V tomto tvare je obrázok následne uložený.

Ďalšie kroky sú zamerané na odoslanie obrázku na rozpoznanie a prijatie rozpoznaného objektu. Na tieto operácie je potrebné importovať balíčky *json, requests, sys* a *base64*. Začneme zostavením URI, kde sa volá funkcia *b64encode*, ktorá zakóduje reťazec do textovej podoby.

```
image_uri = "" + base64.b64encode(
    open(file, "rb").read())
```

Toto vytvorené URI je použité ako hodnota indexu "src" v JSON objekte, ktorý tvorí datovú časť HTTP POST požiadavky. Tento objekt je vytvorený funkciou *json.dumps*. Hlavička je tvorená indexami "app_id" a "app_key", čo je dvojica identifikátorov obdržaných pri registrácii, ako je uvedené v sekcii Analýza. Spolu s adresou, na ktorú sa požiadavka posiela, tvoria data a hlavička argumenty funkcie *post*.

```
r = requests.post("http://api.mathpix.com/v3/latex",
data=json.dumps({'src': image_uri}),
headers={"app_id": "ID", "app_key": "key",
"Content-type": "application/json"})
```

Požiadavka vracia výsledok ako objekt *Response*. Pred načítaním samotnej rozpoznanej rovnice je nutná konverzia textu odpovede do formátu JSON a to funkciou *json.loads*. Výsledný získany objekt obsahuje mnoho indexov medzi nimi napríklad presnosť rozpoznania, údaj či sa jedná o matematickú formulu a iné. Nás zaujíma index "latex", ktorý obsahuje odfotografovanú formulu preloženú do latex-u.

```
data = json.loads(r.text)
```

```
equa = data['latex']
```

Pred uložením reťazca do pamäte je nutné ho konvertovať do kódovania UTF-8. Toto učínime volaním funkcie *encode*.

```
equa1utf = equa.encode("utf-8")
```

Na záver uložíme rozpoznaný latex-ový reťazec do pamäte robota. K tomu je potrebné vytvoriť spojenie s modulom *ALMemory* rovnako ako sme to robili v prípade kamery na začiatku tohoto bloku. Samotné uloženie je zaistené funkciou *insertData*, ktorá prijíma ako argumenty dvojicu reťazcov identifikátor, hodnota, ktoré reprezentujú túto premennú v pamäti.

```
mem = ALProxy("ALMemory", "10.10.48.252", "9559")
mem.insertData("equa1utf", equa)
```

Týmto sa blok rozpoznania končí a prechádza sa k samotnému prekladu matematickej formule do českého jazyka.

3.1.3 Blok Preklad

Hlavnou úlohou tohoto bloku je vytvoriť reťazec v českom jazyku, ktorý odpovedá odfotografovanej matematickej formuly a druhý reťazec, ktorý bude odovzdaný na výpočet. Preklad začína načítaním reťazca z pamäte. Opäť je potrebné vytvoriť spojenie s modulom pamäte a pomocou funkcie *getData* a názvu premennej obdržíme hodnotu reťazca.

```
mem = ALProxy("ALMemory", "10.10.48.252", "9559")
equa1 = mem.getData("equa1utf")
```

Na uľahčenie práce s reťazcom sú najprv odobraté medzery a biele znaky z reťazca.

```
equa1 = equa1.replace(' ', '')
```

Keďže majú matematické operácie jednoznačnú reprezentáciu v latex-u začneme overením, či sa nejedná o viacriadkové pole symbolov, teda o sústavu rovníc alebo maticu.

```
if(equa1.find("\\begin{array}") != -1 ):
  if(equa1.find("=") == -1):
    equa1 = getMatrix(equa1)
  else:
    equa1 = multiEqua(equa1)
```

Pred začatím prekladu sa uistíme či sa nejedná o deriváciu alebo limitu, ktoré je nutné ešte upraviť. Derivácia nájdeme podľa charakteristickej prípony `^{\prime}`, ku ktorej priradíme to to čo chceme derivovať. Pokiaľ v reťazci nájdeme znak '=' vieme, že sa jedná o sústavu rovníc a zavolá sa pomocná funkcia *multiEqua*, ktorá má za úlohu preložiť danú sústavu rovníc. V opačnom prípade vstupujeme do funkcie *getMatrix*, ktorá spracuje danú maticu

a operácie s ňou spojené. Obe tieto funkcie sú popísané podrobnejšie ďalej v texte.

V prípade, že sa nejedná o sústavu rovníc alebo maticu postupujeme ďalej v úprave reťazca. Rozdelíme si reťazec pomocou funkcie *split* podľa znaku '\', ktorý predchádza jednotlivým názvom matematických operácii. Kedže je znak spätného lomítka v jazyku Python považovaný za špeciálny, je potrebné pridať pred neho ešte jeden rovnaký znak '\', ktorý ho tejto vlastnosti zbaví.

```
equa = equa1.split(\\)
```

Ďalej je potrebné nahradiť výraz `operatorname`, ktorým latex popisuje niektoré funkcie. Tento výraz nahradíme menom popisovanej funkcie. Celé pole prechádzame v cykle a pomocou regulárneho výrazu a funkcie *search*, zistíme, či reťazec obsahuje túto sekvenciu znakov. Môžeme si všimnúť, že regulárny výraz je obalený v zátvorkách, týmto si v prípade úspechu funkcia *search*, zapamätá nájdený reťazec a uloží. V prípade neúspechu je návratovou hodnotou *None*. Upravené reťazce sa ukladajú do nového poľa.

```
fixedEqua = []
for subword in equa:
  m = re.search(r"operatorname\{([\+\-\*\/\(\)\^A-Za-z0
      -9]+)\}", subword)
  if(m != None):
    oper = m.group(1)
    equa2 = subword.replace("operatorname{"+oper+"}",
        oper)
    fixedEqua.append(equa2)
  else:
    fixedEqua.append(subword)
```

Na záver hlavného prekladového bloku sa nové pole upravených reťazcov prechádza v cykle a volá sa na ne pomocná funkcia *toCZ*. Po ukončení cyklu sa odstráni uložený latex-ový reťazec z pamäte robota. Preložený reťazec je odoslaný ako výstup do vstavaného bloku Text-to-Speech.

```
equaWord = ""
for word in fixedEqua:
  equaWord += toCZ(word)
mem.removeData("equa1utf")
```

3.1.3.1 Pomocná funkcia toCZ

Pomocná funkcia *toCZ*, je volaná na jednotlivé časti latex-ového reťazca. Základom je zložený príkaz **if..elif..else**, ktorý testuje reťazec na kľúčové slová (int, sum, lim, ...).

V prípade *int* sa jedná integrál a nasleduje zisťovanie medzí. V latexu je definovaný ako `\int_{lower}^{upper}`. Pokiaľ sa jedná o neurčitý integrál je

výsledkom regulárneho výrazu, ktorý ich vyhľadáva, objekt *None*. V opačnom prípade sa medze prečítajú a pridajú do konečného reťazca. Medzi načítanie jednotlivých medzí je pridaný príkaz *replace* aby sa zabránilo opakovanému rozpoznaniu hornej medze. Inicializácia reťazcov *lower* a *upper* je vynechaná.

```
if(word.find("int") != -1):
  intg = "integrál "
  m = re.search(r"\{([A-Za-z0-9_]+)\}", word)
  if(m != None):
   lower = m.group(1)
   word1 = word.replace('{' + lower + '}', '')
   m = re.search(r"\{([A-Za-z0-9_]+)\}", word1)
   upper = m.group(1)
   intg += "od " +lower + " do " + upper + " "
  intg += "z "
  return intg
```

Pokiaľ sa jedná o sumu, postup je podobný ako pri integráloch, rozdiel je len vo formulácii konečného reťazca. Suma je definovaná príkazom `\sum_{n=1}^{\infty}`. V konštrukcii regulárneho výrazu povoľujeme aj znak '=', ktorý sa vyskytuje za indexom sumy. Podobne postupujeme aj v prípade limity.

Zmena nastáva pri zlomkoch, ktoré sú definované ako `\frac{nom}{denom}`. Regulárnym výrazom sa extrahuje čitateľ a menovateľ. Tieto reťazce sú predané do pomocnej funkcie *replaceExpo*, ktorej jedinou úlohou je nahradenie číselného exponentu za slovný. V prípade nečíselného je nezmenený. O preklad čísel sa stará funkcia *numtoWord*, ktorá jednoducho vráti slovnú reprezentáciu čísla teda ak je exponent '1' návratovou hodnotou je reťazec 'prvou'.

```
elif(word.find("frac") != -1):
  newWord = replaceExpo(word)
  m = re.search(r"\{([ \^\(\)řěťíčšáA-Za-z0-9_
      \+\-\*\/]+)\}", newWord)
  num = m.group(1)
  num = replaceExpo(num)
  word1 = newWord.replace('{' + num + '}', '')
  m = re.search(r"\{([ \^\(\)řěťíčšáA-Za-z0-9\+\-\*\/]+)
      \}", word1)
  denom = m.group(1)
  denom = replaceExpo(denom)
  return num + " lomeno " + denom + ' ' + dif;
```

Premenná *dif* z posledného riadku predchádzajúceho kódu obsahuje znaky nachádzajúce sa za poslednou zátvorkou prekladaného reťazca. V prípade integrálu obsahuje reťazec diferenciálu 'dx' v ostatných prípadoch operátor nachádzajúci sa medzi jednotlivými prekladmi.

Funkcie typu logaritmus, sínus, kosínus, ..., sú preložené tak, že za meno funkcie je pridaný argument, získaní funkciou *getArg* a nahradené sú jeho exponenty za slovné. Pomocná funkcia *getArg*, pomocou jednoduchého regulárneho výrazu extrahuje argument nachádzajúci sa v zátvorkách.

```
def getArg( word ):
  m = re.search(r"\(([\{\}\^\(\)A-Za-z0-9\+\-\*\/]+)\)",
      word)
  argument = m.group(1);
  argument = replaceExpo(argument)
  return argument
```

3.1.3.2 Pomocná funkcia multiEqua

Úlohou tejto pomocnej funkcie je vyextrahovať sústavu rovníc z latex-ového zápisu. Robí to pomocou regulárnych výrazov, ktoré vyberajú odpovedajúce podreťazce v zložených zátvorkách obsahujúce znak '='. Na začiatku funkcie sú z reťazca odstránené nepotrebné znaky. Nasleduje cyklus, v ktorom sa extrahujú jednotlivé rovnice pomocou vyššie popísaného regulárneho výrazu.

```
def multiEqua( eq ):
  equa1 = re.sub("\{[A-Za-z0-9\+\-\*\/\^]+\}", "", eq)
  equas = []
  while(True):
    m = re.search("\{([A-Za-z0-9\+\-\*\/\=\^]+)\}",equa1
        )
    if(m == None):
      break;
    equa2 = m.group(1)
    equa1 = equa1.replace("{"+equa2+"}", "")
    equas.append(equa2)
  equasWord = "Soustava␣rovnic␣"
  for word in equas:
    equasWord += toCZ(word)
    equasWord += ",␣"
  return equasWord[:-2]
```

Každá rovnica z rozpoznávanej sústavy sa pridá do poľa reťazcov. Toto pole je potom v cykle prechádzané a je preložené pomocou funkcie *toCZ*. Výsledkom je preložená sústava rovníc.

3.1.3.3 Pomocná funkcia getMatrix

Spolu s ďalšou pomocnou funkciou *getVals* popisuje táto funkcia preklad matice do českého jazyka aj pre potreby Wolfram Alpha. Funkcia *getVals* pomocou regulárneho výrazu získa jednotlivé riadky matice a oddelí ich čiarkou.

Vráti reťazec zložený z čísel príslušného riadku matice oddelených čiarkami až na posledné číslo, ktoré je bez čiarky.

```
def getVals( word ):
  vals = ""
  while(True):
    m = re.search("\{([A-Za-z0-9\-]+)\}", word)
    if(m == None):
      break
    vals += m.group(1) + ", ";
    word = word.replace("{"+m.group(1)+"}", "", 1)
  return vals[:-2]
```

Samotná funkcia *getMatrix* sa na začiatku pokúsi získať operátor spojený s maticou. Počet riadkov matice sa zistí na riadku 6. v kóde uvedenom nižsie a to regulárnym výrazom, ktorý extrahuje reťazec zložený zo znakov 'c' alebo 'r' a zistí jeho dĺžku. Táto dĺžka zodpovedá počtu riadkov matice. Následne sa z reťazca odstránia nepotrebné znaky a jednotlivé riadky sú rozdelené funkciou *split* do poľa. Toto pole sa prechádza v cykle a pomocou funkcie *getVals* sú jednotlivé riadky vyextrahované a zabalené do zátvoriek. Inicializácie reťazcov sú vynechané.

```
def getMatrix( equa1 ):
  equa1 = equa1.replace("\\\\", "\\")
  op = re.search(r"\operatorname{([A-Za-z]+)}", equa1)
  if(op != None):
    operation = op.group(1) + " "
  m = re.search("\{(c*|r*)\}", equa1)
  if(m != None):
    num = m.group(1)
  equa1 = re.sub("(.*)"+"{"+num+"}", "" , equa1)
  nums = equa1.split("\\")
  final = "matrix "
  for i in xrange(0, len(num)):
    final += "("
    final += getVals(nums[i])
    final += "), "
  final = final[:-2]
  return operation + final
```

3.1.4 Blok Získanie výsledku

Výsledok získame zo stránky Wolfram Alpha. Opäť sa volá funkcia *post* z balíčka *requests*. Argumentom tejto funkcie je reťazec reprezentujúci stránku spolu s jednotlivými požadovanými parametrami. Prvým z nich je parameter

Input, ktorý obsahuje samotnú matematickú formulu určenú na rozpoznanie. Ďalším parametrom v poradí je *App_ID* obsahujúci kľúč obdržaný pri registrácii. Posledným z parametrov je *Output*, ktorý špecifikuje v akom formáte má byť výsledok doručený. Implicitne je to formát XML, ale v našom prípade použijeme formát JSON. Pred odoslaním požiadavky je nutné upraviť latex-ovú formulu do správneho tvaru. Rovnako ako pri preklade odstránime kľúčové slovo `operatorname` a špeciálne znaky je nutné konvertovať do formátu s percentom. Na to využijeme balíček *urllib* a jeho funkciu *quote*, ktorú voláme v našej pomocnej funkcii.

```
import urllib as ul
def fixLatex( word ):
  fixed = re.sub(r"\\operatorname{([a-zA-Z0-9]+)}", "\g
     <1>", word)
  fixed = re.sub(r"\^\{([a-zA-Z0-9\+\-\*\/\^]+)\}", "^(\
     g<1>)", fixed)
  fixed = ul.quote(fixed)
  return fixed
```

Po úprave reťazca začneme zostavovať samotný odkaz a odošleme požiadavku. Skladá sa z vyššie popísaných parameterov a základu stránky Wolfram Alpha. Výsledok dostaneme vo formáte JSON, ktorý je potrebný načítať do premennej.

```
import requests
import json
import re
equa = False
if('=' in todo):
  equa = True
fixed = fixLatex(todo)
page = "http://api.wolframalpha.com/v2/query?"
inp = "input=" + "\"" + fixed + "\""
key = "appid=W4PYEL-399J7PAPV4"
out = "output=JSON"
full = page + inp + '&' + key + '&' + out
r = requests.post(full)
data = json.loads(r.text)
```

Zostáva už len získať samotný výsledok. Výstup z požiadavky na Wolfram Alpha je usporiadaný do takzvaných **podov**. Každý **pod** má svoj názov a **subpody** obsahujúce plaintext alebo obrázok. V prípade rovníc budeme hľadať cyklom **pod** s názvom "Solution". Inak ich prejdeme všetky a prvý, ktorý obsahuje znak '=' je náš výsledok.

```
if(equa):
  for pod in data['queryresult']['pods']:
```

```
    if(pod['title'] == 'Solution'):
      res = pod['subpods'][0]['plaintext']
else:
  for pod in data['queryresult']['pods']:
    if('subpods' in pod):
      if('plaintext' in pod['subpods'][0]):
        res= pod['subpods'][0]['plaintext']
        if('=' in res):
          break
```

Výsledok je uložený v premennej *res*. Uložíme ju do pamäte robota podobne ako v predchádzajúcich blokoch volaním modulu *AlMemory*.

3.1.5 Blok Preklad výsledku

Na rozdiel od prekladu v predchádzajúcom bloku, je preklad z plaintextu, ktorý sme získali z Wolfram Alpha realizovaný inak. Jedná sa o náhrady anglických názvov funkcií za české názvy (sin(x) sa zmení na sínus(x)), ďalej náhradu znaku '∧' (exponent) za slovo "na".

3.1.6 Ostané bloky

Zvyšné bloky sú tvorené vstavanými blokmi prostredia Choregraphe, tak ako sú popísané na začiatku tejto kapitoly. Bloky Text-to-Speech a Speech-Recognition umožňujú konunikáciu s užívateľom a riadenie chodu programu. Blok Tactile-Head signalizuje začiatok rozpoznávacieho procesu hneď ako je detekovaný dotyk na jednom z troch hlavových senzorov. Bloky typu Log sú určené na ladenie a odchytávanie informačných a ladiacich výpisov z blokov prekladu, rozpoznania a výpočtu.

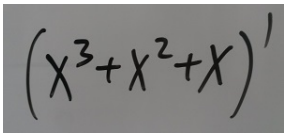
Kapitola 4

Testovanie

Kapitola popisuje a ukazuje jednotlivé testované príklady a ich výsledky. Výsledkami rozumieme jednak správne rozpoznanie príkladu ako aj dodanie správneho výsledku. Testované boli rôzne typy príkladov a to: integrály určité a neurčité, derivácie, limity, sústavy rovníc, rovnice, determinant matice.

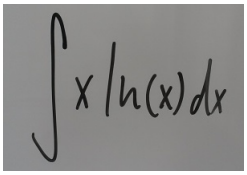
4.1 Príklady a výsledky

Nasleduje prehľad jednotlivých príkladov testovaných na robotovi NAO. Jednotlivé testované obrázky sú popísané odpovedajúcimi matematickými formulami. Odpovede a výsledky od robota sa nachádzajú vždy po príslušným obrázkom.



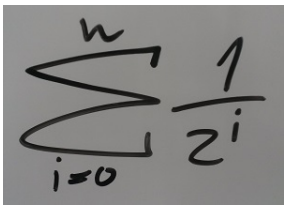
Obr. 4.1: Derivácia z $x^3 + x^2 + x$

Odpoveď robota: Derivace z x na třetí + x na druhou + x. Výsledok od robota: $4x + 1$.



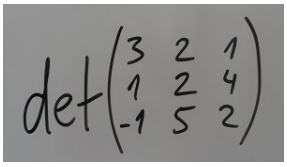
Obr. 4.2: Integrál $\int xln(x)dx$

Odpoveď robota: Integrál x přirozený logaritmus x dx. Výsledok od robota: $1/4x^2(2ln(x-1)) + constant$.



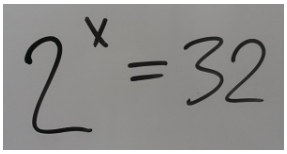
Obr. 4.3: Suma $\sum_{i=0}^{n} 1/2^i$

Odpoveď robota: Suma pro i rovná se 0 až n z 1 lomeno 2 na i-tou. Výsledok od robota: $2 - 2^(- n)$



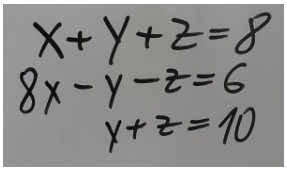
Obr. 4.4: Determinant matice

Odpoveď robota: -. Zlá detekcia matice.



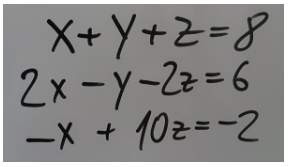
Obr. 4.5: Rovnica $2^x = 32$

Odpoveď robota: -. Zlá detekcia rovnice.



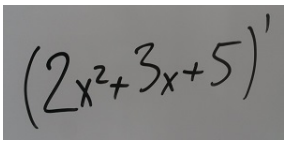
Obr. 4.6: Sústava rovníc

Odpoveď robota: soustava rovnic x+y+z rovná se 8, 8x-y-z rovná se 6, y+z rovná se 10. Výsledok od robota: -. Sústava nemá riešenie.



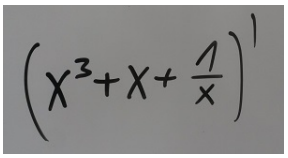
Obr. 4.7: Sústava rovníc

Odpoveď robota: soustava rovnic x+y+z rovná se 8, 2x-y-2z rovná se 6,-x+10z rovná se -2. Výsledok od robota: $x = 138/29, y = 86/29, z = 8/29$. Sústava nemá riešenie.



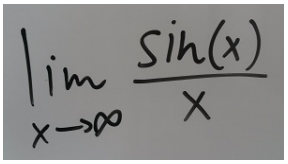
Obr. 4.8: Derivácia z $2x^2 + 3x + 5$

Odpoveď robota: -. Zlá detekcia. Odpoveď na druhý pokus: Derivace z 2x na druhou + 3x + 5. Výsledok na druhý pokus: $4x + 3$.



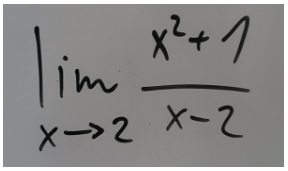
Obr. 4.9: Derivácia z $x^3 + x^2 + 1/x$

Odpoveď robota: -. Zlá detekcia.



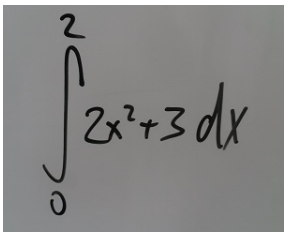
Obr. 4.10: Limita $\lim_{x\to\infty} \frac{sin(x)}{x}$

Odpoveď robota: -. Zlá detekcia. Odpoveď na druhý pokus: Limita pro x jedoucí k nekonečno z sinus(x) lomeno x. Výsledok na druhý pokus: 0.



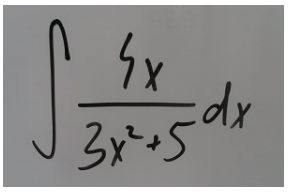
Obr. 4.11: Limita $\lim_{x\to 2} \frac{x^2+1}{x-2}$

Odpoveď robota: Limita pro x jedoucí k 2 z x na druhou + 1 lomeno x - 2. Výsledok od robota: $-\infty$.



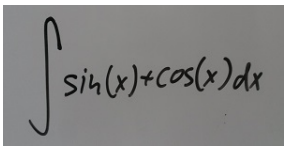
Obr. 4.12: Integrál $\int_0^2 2x^2 + 3dx$

Odpoveď robota: -. Zlá detekcia. Odpoveď na druhý pokus: Integrál od 0 do 2 z 2x na druhou +3dx. Výsledok na druhý pokus: 34.



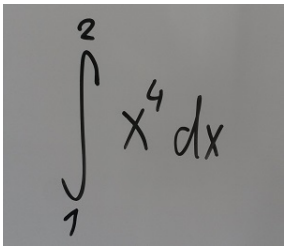
Obr. 4.13: Integrál $\int \frac{4x}{3x^2+5}dx$

Odpoveď robota: Integrál z 4x lomeno 3x na druhou +5dx. Výsledok od robota: $2/3log(3x^2 + 5) + constant$.



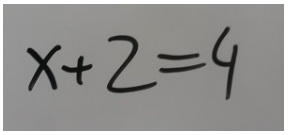
Obr. 4.14: Integrál $\int sin(x) + cos(x)dx$

Odpoveď robota: Integrál z sínus(x) + kosínus(x)dx. Výsledok od robota: $sin(x) - cos(x) + constant$.



Obr. 4.15: Integrál $\int_1^2 x^4 dx$

Odpoveď robota: -. Zlá detekcia. Odpoveď na druhý pokus: Integrál od 1 do 2 z x na čtvrtou dx. Výsledok na druhý pokus: 31/5.



Obr. 4.16: Rovnica $x + 2 = 4$

Odpoveď robota: x + 2 rovná se 4. Výsledok od robota: x rovná se 2.

4.2 Diskusia a zhodnotenie výsledkov

Testovanie bolo realizované v učebni 1048 budovy A, FIT ČVUT za denného svetla bez umelého osvetlenia. Robot NAO bol posadený na stole vzdialený od tabule približne 1.5 metra. Príklady boli písane čiernou fixou na biely povrch. Dohromady bolo testovaných 16 príkladov, z toho dva príklady na určitý integrál, po tri príklady na neurčitý integrál a derivácie. Dve sústavy rovníc o troch neznámych, jedna lineárna rovnica, jeden determinant matice, dve limity funkcie a exponenciálna rovnica. Pri niektorých príkladoch si môžeme všimnúť testovanie na druhý pokus. Pre účely tohoto nového pokusu bol robot presunutý na stoličku bližšie k tabuli. Zlepšila sa detekcia niektorých príkladov a tým aj poskytovanie výsledkov. Testy s rovnakými rovnicami boli vykonané aj na papieri, napísané modrým perom. V tomto prípade detekcia zlyhala iba na príklade determinantu matice, kde bola zle rozpoznaná operácia.

Obr. 4.17: Graf úspešnosti rozpoznania príkladov na tabuli

Hoci je veľkosť testovacej sady malá, z grafu môžeme vyvodiť, že keď sme robota presunuli bližsie k tabuli, tak úspešnosť rozpoznania sa zvýšila. Avšak správne a konzistentné rozpoznanie závisí na mnohých faktoroch. Osvetlenie a typ písma hrajú veľkú rolu pri správnom rozpoznávaní. Dôležité je, aby sa napísaný znak nedotýkal ostatných znakov alebo, aby jeho charakteristiky nepripomínali iné znaky.

Záver

Programovanie robota NAO je zaujímavejšia a zábavnejšia forma programovania v porovnaní s vývojom softwaru, pretože programátor má okamžitú odozvu, či už vo forme reči alebo rôznych činností. V našej práci sme navrhli a vytvorili aplikáciu, ktorá je schopná riešiť matematické príklady a to rovnice a sústavy rovníc, matice, integrály, derivácie a sumy. Aplikácia beží na humanoidnom robotovi NAO spolu s technológiou OCR, ktorá je zaistená aplikáciou MathPix a výpočtom zabezpečeným stránkou Wolfram Alpha. Navrhnutá a vyvinutá bola v grafickom prostredí Choregraphe spolu s doplnením vlastných blokov v jazyku Python. Pri vývoji sme použili sadu knižníc a funkcií na prácu s obrazom OpenCV. Verziu 2.7 jazyka Python a jeho balíčky *json, numpy, requests, re, system*. Veľkú časť práce odviedol práve balíček *re*, ktorý implementuje regulárne výrazy, pomocou ktorých sú nahradzované a rozpoznávané jednotlivé reťazce. Beh aplikácie je priamočiary a začína pozdravom robota a načítaním obrázku, ktorý je následne odoslaný na rozpoznanie. Rozpoznaný text je preložený a odoslaný na výpočet. Vypočítaný text je opäť preložený a vyslovený robotom. Aplikácia sa končí otázkou na zopakovanie výpočtu alebo ukončenie. Projekt je možné využiť na výpočet rôznych typov príkladov napísaných na tabuli či papieri alebo na kontrolu správnosti.

Pri tvorbe tejto práce sme sa oboznámili s technikami úpravy obrázkov ako sú napríklad: binarizácia obrazu (thresholding), konverzia z jedného farebného priestoru do iného, erózie a rozšírenia, pomocou ktorých sa zbavíme šumu v obrázku alebo jeho pozadí. Ďalej s funkciami ako sú extrakcie kontúr, ich hierarchie a aproximácia polygoniálnej krivky. Zoznámili sme sa aj s technológiou OCR, jej použitím a históriou. Tesseract je od roku 2005 open-source nástroj na rozpoznávanie znakov a slov vyvinutý firmou Hewlett-Packard a neskôr prevzatý spoločnosťou Google. Podporuje rozpoznanie viac ako 100 jazykov a môže byť vytrénovaný na rozpoznanie iných. Na záver sme sa okrajovo zoznámili s neurónovými sieťami, hlbokým a strojovým učením, ktoré sú niekedy využívané na tento typ úloh.

Konkrétne sme v práci využili funkcií OpenCV, a to thresholding na získa-

nie binárneho obrázku, z ktorého sme potom prostredníctvom erózie a rozšírenia (dilation) odstránili nečistoty. Tento obrázok je potom odoslaný prostredníctvom HTTP POST požiadavky na rozpoznanie, ktorá nám vracia JSON objekt s rozpoznaným textom. Použitý je externý rozpoznávač Mathpix, ktorý je dostupný aj vo forme mobilnej aplikácie. Text je vo formáte LaTeX, ktorý sme spracovali do českého jazyka a mierne upravili pre potreby Wolfram Alpha. Miernou úpravou sa rozumie náhrada niektorých kľúčových slov do prirodzeného jazyka a náhrada zložených zátvoriek za okrúhle. Využili sa k tomu funkcie balíčka *re*, ktorý implementuje regulárne výrazy. Funkcie vyhľadávania a náhrady vytvorili silný nástroj na analýzu a jednoduchú manipuláciu s reťazcami. V poslednej fázy aplikácie je výraz odoslaný na výpočet cez POST požiadavku, s možnosťou špecifikovať návratový formát a ďalšie parametre.

Úspešnosť rozpoznania sa pohybovala na 50% vo vzdialenosti 1.5 metra od bielej tabule. Úspešnosť sa zdvihla na 81.5%, keď sme presunuli robota do vzdialenosti < 1 meter. Výsledky vypočítané robotom boli správne až na prípad druhej limity, kedy robot vrátil výsledok jednostrannej limity, miesto konštatovania, že limita neexistuje.

Rozšírenie tohoto projektu do budúcna môže zahrňovať napríklad pridanie interaktívneho vysvetľovania danej problematiky, náhodné generovanie príkladov pre danú problematiku spolu s písaním na papier, implementácia vlastného OCR enginu, ktorý by rozpoznával matematiku rovno do českého alebo anglického jazyka prípadne do jazyka Wolfram Mathematica. Do úvahy prichádza aj využitie OCR na čítanie dokumentov, kníh alebo iných textov, ktorý by slúžil ako pomocník zrakovo postihnutým alebo spraviť z robota NAO interaktívny prekladač s výkladom.

Literatúra

[1] Schantz, H. F., *The history of OCR: optical character recognition*, Recognition Technologies Users Association (1982)

[2] Bishop, Ch. M., *Neural Networks for Pattern Recognition*, Claredon Press Oxford (1996)

[3] Teh, C.H. and Chin, R.T., *On the Detection of Dominant Points on Digital Curve.*, PAMI 11 8, pp 859-872 (1989)

[4] Suzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following.*, CVGIP 30 1, pp 32-46 (1985)

[5] Kay, A., *Tesseract: an Open-Source Optical Character Recognition Engine*[online], Published in 2007, From https://www.linuxjournal.com/article/9676, Accessed on 20.4.2018

[6] SoftBank Robotics Europe, *NAOqi Documentation*, From http://doc.aldebaran.com/2-1/legal/notice.html, Accessed on 26.3.2018

[7] OpenCV team, *Opencv Documentation*, From https://docs.opencv.org/2.4.13.6/, Accessed on 3.4.2018

[8] Melanson D., *Nao robot replaces AIBO in RoboCup Standard Platform League*[online], Published 16.8.2007, From https://www.engadget.com/2007/08/16/nao-robot-replaces-aibo-in-robocup-standard-platform-league/, Accessed on 1.5.2018

[9] Bhat A., Chojnacki A., Knapp E., *The future is Nao: Teaching Mathematics to young schoolchildren using humanoid robots*[online], Published November 2016, From https://www.researchgate.net/publication/310234284_The_Future_is_Nao_Teaching_Mathematics_to_young_Schoolchildren_using_Humanoid_Robots, Accessed on 1.5.2018

Dodatok **A**

Zoznam použitých skratiek

GUI Graphical User Interface
XML Extensible Mark-up Language
OCR Optical Character Recognition
API Application Programming Interface
SSH Secure Shell Protocol
OpenCV Open Computer Vision
FTP File Transfer Protocol
JSON JavaScript Object Notation
URI Uniform Resource Identifier
HTTP Hyper Text Transfer Protocol

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `project` ..... adresár obsahujúci projekt v programe Choregraphe
- `src`
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `BP_stankric.pdf` ..... text práce vo formáte PDF