Tímový projekt

ÚRK FEI STU

LS 2019/2020

Bc. Andrej Chmurčiak

Bc. Viktor Christov

Bc. Ivan Kenický

Bc. Bálint Sallay

ZADANIE

Študijný program: ***Robotika a kybernetika***

Študijný odbor: ***Kybernetika***

Vedúci projektu: ***Ing. Michal Adamík (1. tím), Ing. Miroslav Kohút (2. tím)***

Miesto vypracovania projektu: ***Ústav robotiky a kybernetiky***

Riešitelia: Bc. Andrej Chmurčiak, Bc. Viktor Christov , Bc. Ivan Kenický, Bc. Bálint Sallay

Názov projektu: Inteligentné plánovanie pohybu robota

Špecifikácia zadania:

Cieľom zadania je vytvoriť inteligentné metódy riadenie robotického systému podľa pokynov vedúceho projektu. Očakáva sa vytvorenie aplikácie, ktorá je zameraná predovšetkým na riadenie pohybu samotného robota, t. j. polohovanie a navigácia.

Úlohy:

1. Špecifikujte úlohy, ktoré si osvoja jednotliví členovia tímu na robotickom systéme.
2. Zdokumentujte plánovaný postup práce a projektovú dokumentáciu.
3. Analyzujte možné metódy riadenia a vylepšenia robotického systému.
4. Implementujte vybrané metódy a vylepšenia do robotického systému. V prípade nutnosti zakúpenia HW alebo SW dostatočne vopred komunikujte s vedúcim projektu.
5. Výsledky úloh spracujte a vyhodnoťte. Pripravte si prezentáciu Vami dosiahnutých výsledkov.
6. Vypracujte odborný článok o dosiahnutých výsledkoch.

Termín odovzdania projektu: 15.5.2020

V Bratislave dňa 17.2.2020

prof. Ing. Jarmila Pavlovičová PhD.

garantka študijného programu

PLÁN PRÁCE

Pravidelné testovanie

Stretávanie sa a komunikácia za účelom riešenia úloh a problémov

Zapojenie a spojazdnenie grippera

Definícia a pripojenie standu

Testovanie programu

Úpravy programu za cieľom jeho zlepšenia

Úpravy programu za cieľom opravy potenciálnych chýb

Oboznámenie sa so zadaním tímového projektu

Identifikácia možných nedostatkov

Analýza možností vylepšenia, odstránenia nedostatkov

Vytváranie rešeršu

Reorganizácia práce vzhľadom na situáciu, zmena spôsobu stretávania a komunikácie

Demonštrácia výsledkov rešeršu

Vytváranie odborného článku

Analýza a zhodnotenie záverečného stavu

Vytváranie prezentácie

Vytváranie posudku

Obhajoba – máj 2020

Zápisnica zo dňa 6.8.2019

Prítomní:

Bc. Andrej Chmurčiak, Bc. Bálint Sallay

Úlohy:

1. Zapojenie grippera
2. Definícia vstupov a výstupov pre ovládanie grippera
3. Postavenie podstavca – štandu

Dosiahnuté výsledky:

Gripper sa nám úspešne podarilo zapojiť na manipulátor.

Nadefinovali sme všetky potrebné vstupy a výstupy na ovládanie gripperu, čo je prvým krokom k úspešnému stavaniu steny.

V rámci tohto dňa sme tiež postavili štand, na ktorom sa manipulátor bude nachádzať. Týmto sme zabezpečili jeho stabilitu ale aj vyhradili bezpečnú zónu práce.

Neskôr bude potrebné tento štand vymodelovať aj pre Rhinoceros aby sme ho vedeli využívať pri simuláciách.

Zápisnica zo dňa 9.8.2019

Prítomní:

Bc. Andrej Chmurčiak, Bc. Bálint Sallay

Úlohy:

1. Definícia štandu (vytvorenie 3D modelu) pre program Rhinoceros
2. Pripojenie k okolitému prostrediu (vytvorenie vstupov, výstupov) a uloženie do Grasshoper premennej
3. Spojenie modelu s programom robota - získanie informácie o existencii tejto prekážky
4. Testovanie programu

Dosiahnuté výsledky:

V rámci tohto stretnutia sa nám úspešne podarilo vymodelovať (zadefinovať) spomínaný podstavec do Rhinocerosu, pripojiť k okolitému prostrediu ako aj uložiť ho do potrebnej formy a teda Grasshoper premennej.

Po spojení modelu s programom sme prešli na testovanie tohto programu. Zistili sme, že naša konfigurácia grippera má krátky dosah a preto ju bolo nutné zmeniť.

Konfiguárciu sme upravili, čím sme dosiahli zväčšenie pracovného priestoru (dosahu) manipulátora, nakoľko pri kolmej konfigurácii toto nie je možné.

Zápisnica zo dňa 14.8.2019

Prítomní:

Bc. Andrej Chmurčiak, Bc. Bálint Sallay

Úlohy:

1. Otáčanie gripperu
2. Pridanie medzibodu

Dosiahnuté výsledky:

V tento deň sme mali v pláne vyriešiť otáčanie gripperu ako aj pridanie medzibodu avšak toto sa nám nepodarilo.

Ošetrenie natáčania gripperu aby sme predchádzali singularitám, nám svojou náročnosťou zabralo viac času ako sme očakávali a preto sme po úspešnom vyriešení prvého bodu, presunuli bod druhý na nasledujúce stretnutie

Zápisnica zo dňa 15.8.2019

Prítomní:

Bc. Andrej Chmurčiak, Bc. Bálint Sallay

Úlohy:

1. Pridanie medzibodu
2. Odstránenie trhavého pohybu

Dosiahnuté výsledky:

Dnes sa nám podarilo pridať medzibod, čo bolo našim plánom už na predchádzajúcom stretnutí. Pridaním tohto medzibodu, sme umožnili stavanie steny vo viacerých kvadrantoch.

Ďalší problém, ktorý sme videli bol trhavý pohyb pri stavaní steny. Po úvahách sme sa zhodli na tom, že náš aktuálny spôsob pohybu - lineárny - *moveL*, by sme mohli zmeniť na kĺbový - *moveJ*. Našu úvahu sme aplikovali a zistili sme, že bola správna a pohyb je oveľa plynulejší.

Zápisnica zo dňa 18.2.2020

Prítomní:

Ing. Michal Adamík, Bc. Andrej Chmurčiak, Bc. Viktor Christov, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Stretnutie s vedúcim tímového projektu
2. Oboznámenie sa so zadaním
3. Voľba vedúceho tímu
4. Vytvorenie predbežného plánu, dodatok k predchádzajúcemu plánu
5. Predbežné prerozdelenie úloh

Dosiahnuté výsledky:

Po úvodnom zoznámení sa, sme sa oboznámili so zadaním a jeho úlohami. Pre lepšiu organizáciu práce sme si zvolili vedúceho tímu, ktorým sa jednohlasnou voľbou stal Bc. Ivan Kenický.

Vzhľadom na úlohy zo zadania a už dosiahnuté výsledky, sme si vytvorili predbežný plán práce.

Taktiež sme si predbežne prerozdelili úlohy a dohodli sa na pravidelnom stretávaní sa s možnými obmenami podľa potreby.

Zápisnica zo dňa 5.3.2020

Prítomní:

Ing. Michal Adamík, Bc. Andrej Chmurčiak, Bc. Viktor Christov, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Oboznámenie sa nových členov (Christov, Kenický) s manipulátorom
2. Stavanie standu
3. Testovanie programu z výstavy
4. Identifikácia nedostatkov
5. Pojednanie o možných vylepšeniach

Dosiahnuté výsledky:

Členov Christov a Kenický sme oboznámili s manipulátorom a potom sme spoločne stavali už skôr spomínaný stand. Následne sme testovali program, s ktorým sme boli na výstave, za účelom demonštrácie ale i identifikácie možných nedostatkov.

Zistili sme, že nedostatkom nášho programu je používanie príkazov *moveJ*, *moveL* spoločne s jedným medzibodom, čo obmedzuje pracovný priestor robotického manipulátora na dva kvadranty. Keďže chceme využiť pracovný priestor robota optimálne, je vhodné zmeniť pôvodný program tak, že budeme používať príkazy *moveJ* a *moveL* bez medzibodu a medzibod budeme používať spolu s príkazom *moveC*, ktorý zabezpečí že nedôjde ku kolízii robotického manipulátora a zabezpečí aj zväčšenie pracovného priestoru na všetky štyri kvadranty.

Zápisnica zo dňa 26.3.2020

Prítomní:

Ing. Michal Adamík, Bc. Andrej Chmurčiak, Bc. Viktor Christov, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Zmena spôsobu komunikácie
2. Pojednanie o aktuálnom stave riešenia
3. Reorganizácia práce

Dosiahnuté výsledky:

Dňa 26.3. sme sa po istej odmlke vzhľadom na aktuálnu situáciu opäť stretli, no tentoraz cez tele-konferenciu. Dohodli sme sa, že pre efektívnu komunikáciu budeme používať *Microsoft Teams*, ktorý umožňuje tele-konferenčný hovor a v prípade potreby aj ďalšie elektronické spôsoby dorozumievania sa.

V rámci tohto hovoru sme taktiež pojednali o aktuálnom stave riešenia. Následne sme trochu pozmenili plán a spôsob práce so zreteľom nemožnosti reálneho testovania programu a zhoršenej komunikácie, vzhľadom na už spomínanú výnimočnú situáciu.

Zápisnica zo dňa 14.4.2020

Prítomní:

Ing. Michal Adamík, Bc. Viktor Christov

Úlohy:

1. Pojednanie o výsledkoch rešeršu o aktuálne dostupným možnostiach Git riešení
2. Určenie konkrétnych produktových požiadaviek

Dosiahnuté výsledky:

Odprezentovali sme výsledky prieskumu aktuálne dostupných produktov na trhu.

Následne boli určené požiadavky, podľa ktorých je potrebné nájsť najvhodnejší produkt, analyzovať jeho možnosti a navrhnúť ho ako možné riešenie.

Požiadavky boli najmä: Možnosť písať vývesky

Časová os

Nízka, ideálne nulová cena

Zápisnica zo dňa 17.4.2020

Prítomní:

Ing. Michal Adamík, Bc. Viktor Christov

Úlohy:

1. Demonštrácia práce s Git
2. Tvorenie nových vetiev

Dosiahnuté výsledky:

V rámci dnešného stretnutia sa nám podarilo opísať hlavné časti práce s Git, na základe predošlého prieskumu.

Postupne sme prešli funkciami ako napríklad *checkout* na porovnanie obsahov repozitárov a identifikáciu chýbajúcich súborov repozitári pracovnom, či *commit* na vytvorenie novej verzie obsahu.

Taktiež sme sa venovali ďalším príkazom ako aj tvoreniu nových vetiev a problémami s tým spojenými.

Zápisnica zo dňa 27.4.2020 (1)

Prítomní:

Ing. Michal Adamík, Bc. Viktor Christov

Úlohy:

1. Vyhodnotenie výsledkov prieskumu trhu
2. Prezentácia navrhovaného riešenia podľa predchádzajúcich požiadaviek
3. Demonštrácia klienta

Dosiahnuté výsledky:

Na základe dostupných možností sme sa rozhodovali nad typom úložiska a voľbou klienta. Vzhľadom na cenové požiadavky a druh vzdialeného úložiska, zároveň pod vplyvom karanténnej situácie sme sa rozhodli pre riešenie GitHub, ktoré poskytuje všetko požadované a zároveň s ním už máme aj predchádzajúce skúsenosti.

S ohľadom na už skôr dohodnuté požiadavky na klienta, padla naša voľba na klienta GitKraken, ktorý ich spĺňa – umožňuje časovú os, nástenku a je pre študentov a pedagógov zadarmo.

Zápisnica zo dňa 27.4.2020 (2)

Prítomní:

Ing. Michal Adamík, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Pojednanie o tvorení odborného článku

Dosiahnuté výsledky:

V rámci tohto stretnutia sme si ujasnili akú formu článku by sme chceli, aké dôležité súčasti by mal článok obsahovať a zároveň aký je očakávaný rozsah.

Zápisnica zo dňa 30.4.2020

Prítomní:

Ing. Michal Adamík, Bc. Andrej Chmurčiak, Bc. Viktor Christov, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Zhodnotenie aktuálneho stavu
2. Záverečné úpravy
3. Tvorba prezentácie

Dosiahnuté výsledky:

Keďže sme už boli prakticky na konci projektu, stretli sme sa aby sme vyhodnotili aktuálny stav. Ten bol pomerne potešujúci a dolaďovali sme už len drobnosti a formality.

Zároveň sme začali zaoberať tvorbou prezentácie a to akou formou ju budeme tvoriť a prezentovať.

Zápisnica zo dňa 13.5.2020

Prítomní:

Bc. Andrej Chmurčiak, Bc. Viktor Christov, Bc. Ivan Kenický, Bc. Bálint Sallay

Úlohy:

1. Zhodnotenie práce oponentského tímu
2. Vytvorenie posudku

Dosiahnuté výsledky:

Po prečítaní práce oponentského tímu sme spoločne určili nájdené pozitíva a negatíva tejto práce. Tieto poznatky sme následne spracovali do posudku, ktorého súčasťou sú aj pripomienky a otázky ku práci.

Obsah

Úvod

V rámci tohto tímového projektu by sme chceli vyhotoviť, respektíve vylepšiť naše riešenie na inteligentné plánovanie pohybu robota. Takéto riešenie je následne využiteľné aj v praxi, napríklad pri stavaní budov.

Počas riešenia budeme musieť spolupracovať v tíme, komunikovať, identifikovať problémy, navrhovať možné riešenia, čo nám umožní sa zlepšiť aj v oblasti tímovej spolupráce.

Postupne si prejdeme časťami ako definícia efektora, ktorý použijeme ako koncový pracovný nástroj na stavenie; definíciou využívaného manipulátora, ktorého trasu budeme plánovať.

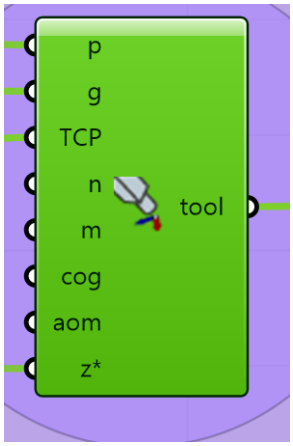
Následne sa povenujeme navrhovaným a aplikovaným vylepšeniam.

V poslednej časti pojednáme o vhodných možnostiach zdieľania súborov, zdrojových kódov a iného – riešení Git. Taktiež nájdeme aj konkrétne pre nás vhodné riešenie.

1 Virtuálne robotizované pracovisko

1.1 Definícia efektora

TacoABB v Grasshopperi obsahuje programový blok, ktorý umožní užívateľovi definovať si nástroj, ktorý je namontovaný na manipulátor. Jedným vstupom spomínaného bloku je model nástroja ako premenná “Mesh”. Použili sme CAD modely poskytnuté výrobcom ktoré sme upravili pre použitie v prostredí Grasshopper.

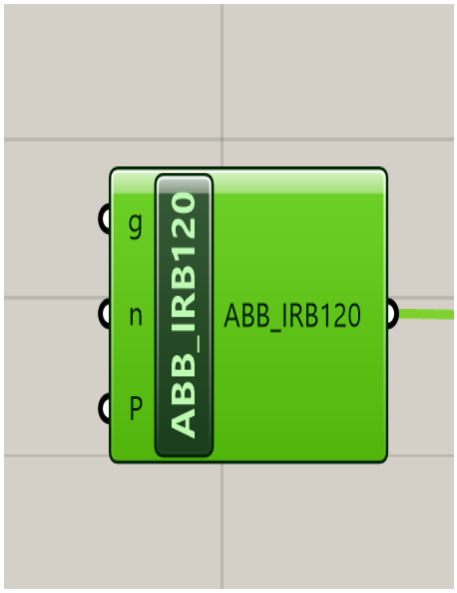


Obrázok 1 Programový blok na definíciu nástroja

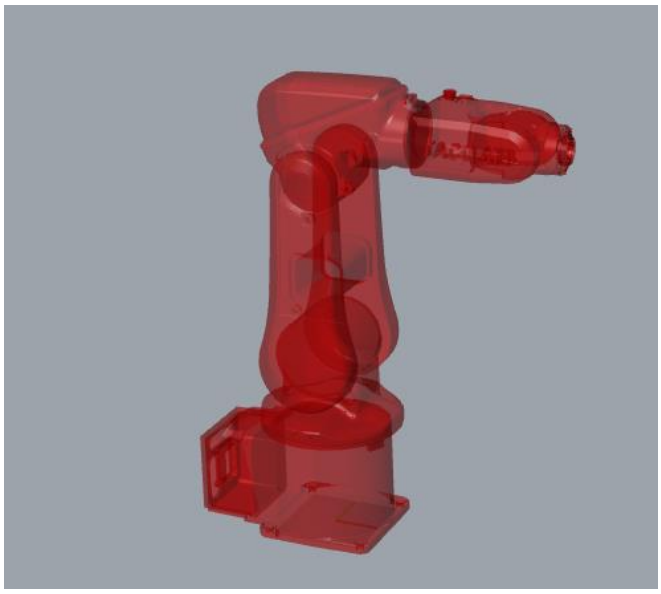
Blok *Tooldata Definition* zobrazený na obrázku 1 má osem vstupov a jeden výstup. Prvý vstup označený písmenom p je rovina, ktorá reprezentuje plochu kde sa spojí posledná os robota s efektorom. Druhý vstup označený písmenom g je premenná typu Mesh, ktorá obsahuje model efektora. Tretí vstup označený TCP je rovina, ktorá reprezentuje koncový bod efektora a jeho orientáciu v priestore. Vstup n je názov nástroja, ktorý bude zobrazený v programovom kóde. Vstup *m* je váha nástroja v kilogramoch. Vstupy *cog* a *aom* sú nepovinné, a obsahujú ťažisko a moment zotrvačnosti nástroja. Posledný vstup, ktorý je označený *z** ako Zone Data opisuje veľkosť generovanej rohovej trajektórie.

1.2 Definícia manipulátora

Pre vytvorenie definície manipulátora sme použili programovaný blok ABB IRB120. Ako môžeme vidieť, blok má tri vstupy a jeden výstup. Vstupy sú nepovinné, predstavujú dodatočné modely, meno robota a pozíciu robota v priestore. Výstupom bloku je model robota ABB IRB120.



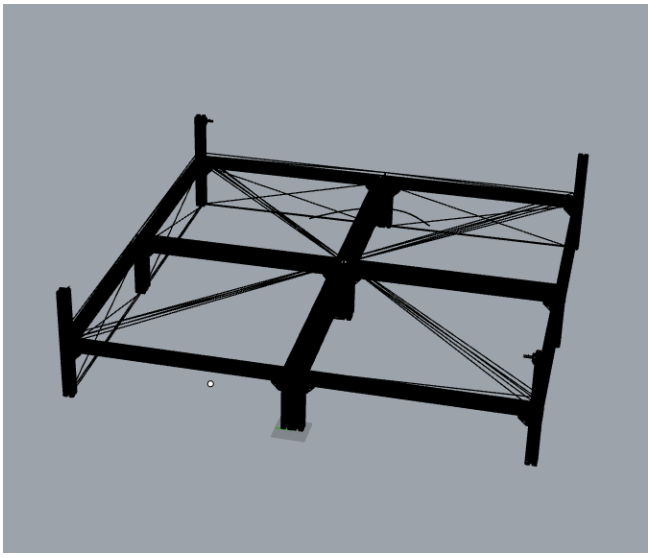
Obrázok 2 Blok na definíciu manipulátora



Obrázok 3 Definovaný manipulátor

1.3 Definícia štandu

Súčasťou robotizovaného pracoviska je aj štand, na ktorý je upevnený manipulátor. Slúži na vyhradenie bezpečnej vzdialenosti okolo manipulátora, taktiež poskytuje rovný povrch pre stavanie steny. Model štandu sme namodelovali v prostredí Rhinoceros3D a uložili do formy "Mesh" na použitie v prostredí Grasshopper. Model štandu sa použije ako vstup bloku na generovanie pohybových trajektórií.

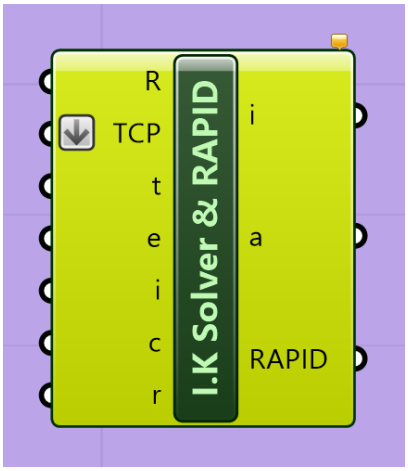


Obrázok 4 Model štandu v Rhinoceros3D

2 Generovanie trajektórií manipulátora a spracovanie vstupných a výstupných signálov

2.1 Simulácia trajektórií a generovanie RAPID

Pre simuláciu trajektórií použijeme blok Inverse kinematics solver&RAPID.

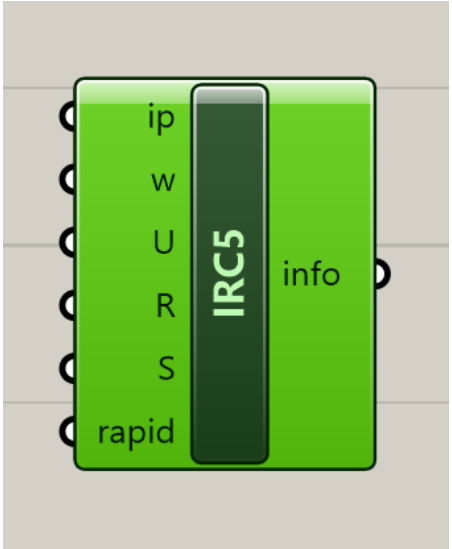


Obrázok 5 Blok na simuláciu trajektórií

Blok má sedem vstupov a tri výstupy. Prvým vstupom je definícia robota, ktorú tvorí výstup z bloku ABB IRB120 ktorú vidíme na obrázku 2 . Druhý vstup TCP, tvoria plochy ktoré reprezentujú bod v priestore a natočenie nástroja. Poradie plôch na vstupe určí trajektórie robota. Tretí vstup je definícia nástroja, ktorú tvorí výstup z bloku *TooldataDefinition*, ktorú vidíme na obrázku 1 . Štvrtý vstup reprezentuje prostredie v ktorom sa manipulátor nachádza a tvorí ho model štandu vo forme premennej “Mesh”. Dáta prostredia sa využívajú pri detekcií zrážok. Piaty vstup je číslo, ktoré určí ktorý krok simulácie sa má zobraziť v Rhinoceros3D. Pripojením vstupu *i* sa taktiež v Rhinoceros3D objaví možnosť spustenia simulácie. Vstupom *c* je možné pridať dodatočné príkazy pre manipulátor. Vstup *r* je vstup typu boolean, určuje či blok vygeneruje výstupný kód RAPID alebo nie. Pre účel simulácie nie je potrebné použiť ani jeden z výstupov. Pri testovaní na manipulátore však použijeme výstup RAPID, ktorý obsahuje program, vykonaný robotom. Blok generuje program pomocou príkazov *MoveL*, čiže lineárny pohyb, čo môže spôsobiť problémy so singularitou.

2.2 Komunikácia prostredia Grasshopper s ovládačom manipulátora IRC5 compact

Po úspešnej simulácií a vytvorení programu je potrebné vygenerovaný program preniesť do ovládača robota. Použijeme blok z TacoABB s názvom IRC5, rovnako ako názov ovládača robota.



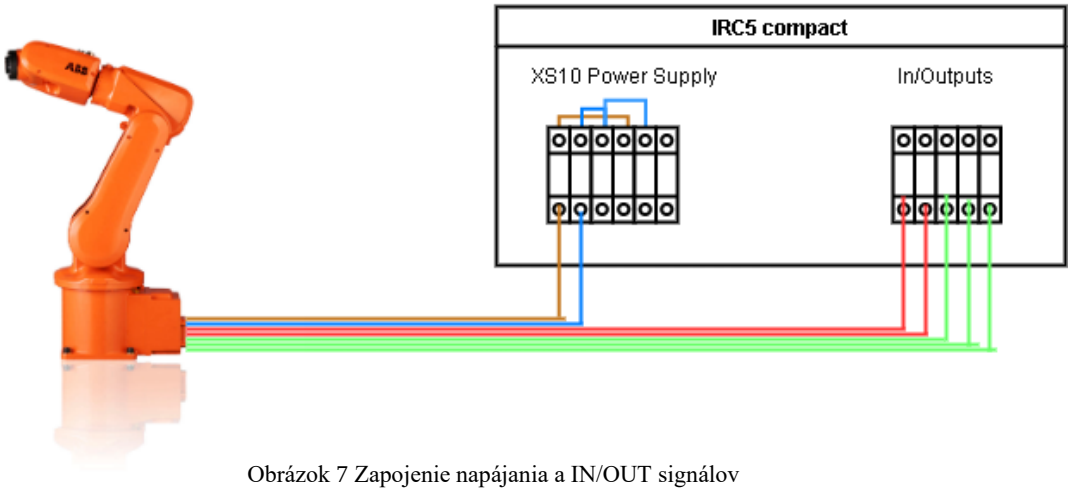
Obrázok 6 Blok na prenos programu

Ako vidíme z obrázku 6 blok má šesť vstupov a jeden výstup. Prvý vstup určuje IP adresu IRC5 compact ovládača robota. Druhý vstup je nepovinný, umožňuje výpis vlastných upozornení pri prvom nahraní kódu RAPID do ovládača. Ďalšie tri vstupy sú typu boolean a pri hodnote 1 umožňujú nahranie programu do ovládača (Upload), spustenie programu (Run) alebo zastavenie programu (Stop). Posledný vstup je samotný program ktorý sa má nahrať do ovládača. Jediným výstupom bloku sú informácie o stavu a úspešnosti nahrávania.

2.3 Zapojenie napájania efektora a vstupno-výstupných signálov

Robotizované pracovisko okrem štandu, manipulátora a ovládača obsahuje aj niekoľko snímačov, konkrétne dva indukčné snímače na efektory, pomocou ktorých vieme určiť, či je efektor otvorený alebo zatvorený. Taktiež používame optoelektrický snímač upevnený na podávači tehál, pomocou ktorého manipulátor kontroluje, či má k dispozícii tehly na stavanie.

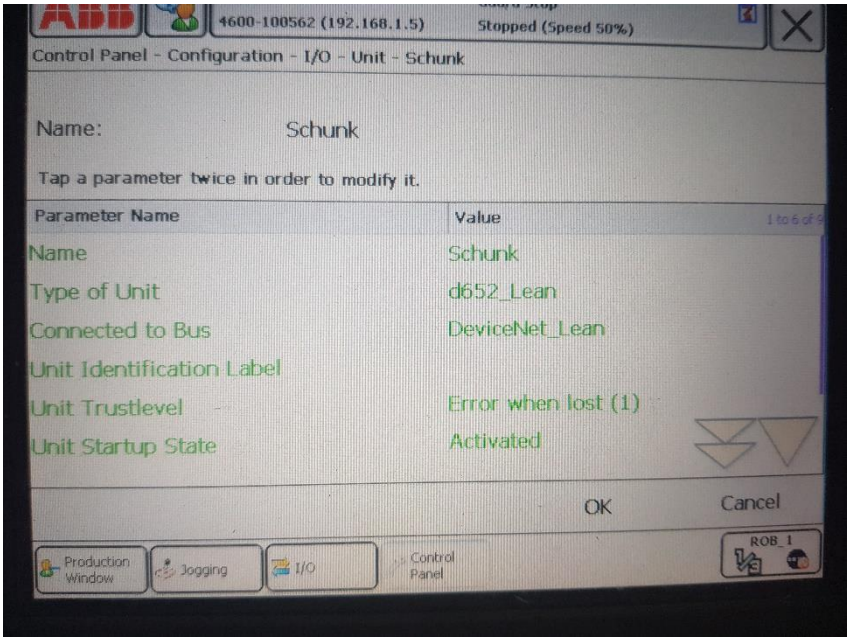
Každý snímač predstavuje jeden vstupný signál. Na ovládanie efektora sme potrebovali dva výstupné signály, jeden pre otvorenie a jeden pre zatvorenie. Všetky signály sme zapojili na vstupno-výstupné rozhranie IRC5 compact ovládača pomocou 44 pinového konektora, ktorý zároveň zabezpečuje aj napájanie snímačov. Pre napájanie efektora je potrebné napätie 24V, ktoré čerpáme zo XS10 zdroja ovládača. Ukážku zapojenie napájania a vstupných, výstupných signálov vidíme na obr. 7.



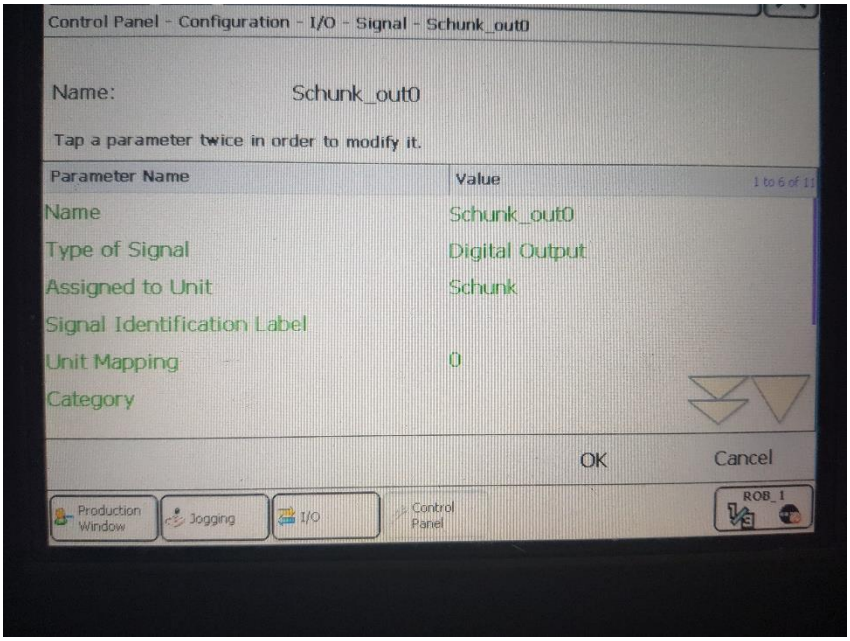
Obrázok 7 Zapojenie napájania a IN/OUT signálov

2.4 Definícia vstupno-výstupných signálov

Aby sme mohli vytvoriť signály, ktoré bude môcť vysielať a prijímať manipulátor cez svoj ovládač museli sme v operačnom systéme manipulátora vytvoriť novú premennú typu „Unit“. Zadefinovanie našej premennej typu Unit môžeme vidieť na obrázku 8 . Vyberieme typ a meno premennej a nastavíme zbernicu ku ktorej je pripojená. Taktiež je potrebné zadať adresu na ktorej sa pripája k želanej zbernici. Táto premenná nám umožní zadefinovať si nové signály, ktoré budú vstupy a výstupy efektora. Podobne postupujeme aj pri definovaní signálov. Vytvoríme novú premennú typu „Signal“, ktorá sa vloží do našej premennej typu „Unit“. Zadáme meno a typ signálu a zadáme adresu signálu zo vstupno-výstupného modulu ovládača manipulátora. Signály môžu byť vstupné alebo výstupné.



Obrázok 8 Definícia premennej typu unit



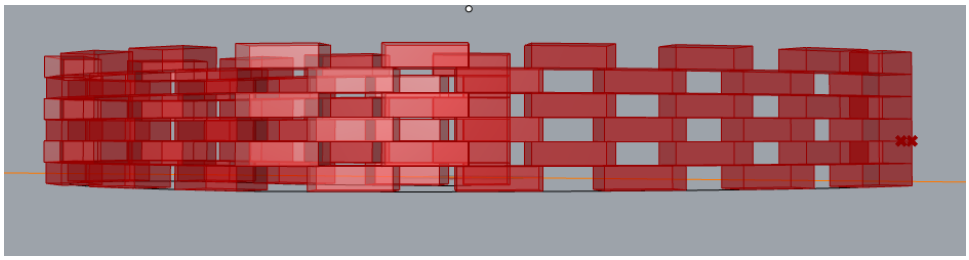
Obrázok 9 Definícia premennej typu signal

3 Optimalizácia generovania trajektórii pre robotický manipulátor v programe Rhinoceros Grasshopper

V pôvodnej aplikácii stavania steny bolo možné postaviť stenu len v dvoch kvadrantoch pracovného priestoru z dôvodu použitia jedného pomocného bodu, cez ktorý musel koncový bod robotického manipulátora prejsť aby nenastala kolízia robotu so samým sebou. Táto aplikácia obsahovala v rapid kóde iba príkazy *moveJ* a *moveL*, čo robotický manipulátor značne obmedzilo. Preto sme sa rozhodli implementovať vhodnú kombináciu príkazov rapid kódu a to konkrétne *moveJ, moveL* a *moveC* a tým rozšíriť možnosti tejto aplikácie.

3.1 Oboznámenie sa s generátorom trajektórii

Náš program je vytvorený a naprogramovaný v prostredí Rhinoceros Grasshoper v grafickom programovacom jazyku. V tomto programe máme tri základné parametre – krivka, ktorá slúži ako pôdorys steny, parameter offset, teda rozostup medzi dvomi tehlami v stene a parameter výška steny. Na základe týchto parametrov sa vygeneruje postupnosť bodov, medzi ktoré je potrebné vložiť body pre zdvihnutie a položenie tehly. Tieto body majú v sebe zakomponovanú aj informáciu o správnom natočení gripperu robotického manipulátora. Z týchto bodov sa vygeneruje rapid kód iba s príkazmi *moveL*. Pre plynulejší pohyb robotického manipulátora pri presúvaní medzi *pick and place* bodmi používame príkaz *moveJ*.

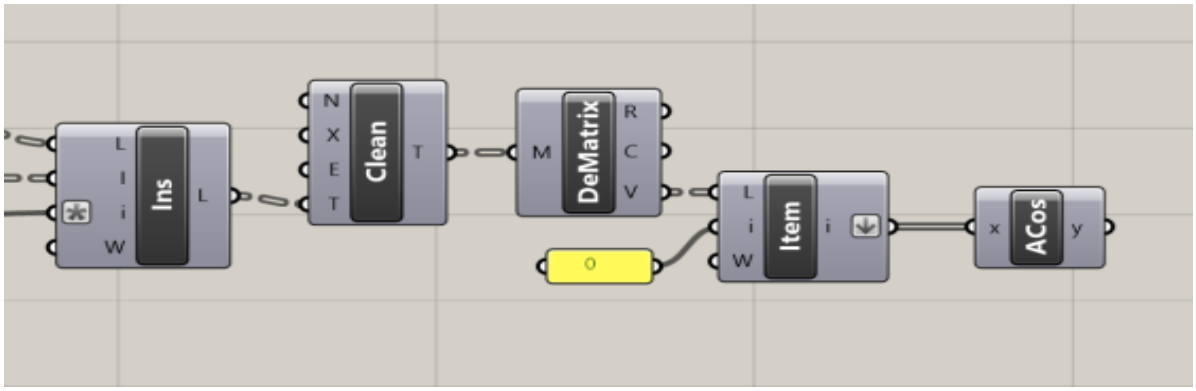


Obrázok 10 Stena vygenerovaná v prostredí Rhinoceros3D

Na obrázku vyššie môžeme vidieť vygenerovanú stenu v prostredí Rhinoceros-u. Ako už bolo spomenuté, každá jednotka - tehla, z ktorej je táto stena vybudovaná obsahuje

informáciu o súradniciach ťažiska tehly a o potrebnom natočení gripperu robotického manipulátora.

Keďže ide o aplikáciu stavania steny, musíme do tejto postupnosti bodov pridať aj bod odkiaľ bude robotický manipulátor tehly brať. Na každú druhú pozíciu do postupnosti tehiel vložíme tento bod pre funkciu pick. Tento bod je definovaný kartézskymi súradnicami a zároveň natočením, ktoré musí robotický manipulátor dodržiavať. Aby sme zabezpečili požadovanú trajektóriu koncového bodu robotického manipulátora, musíme pridať taktiež body v istej výške nad pick bodom a nad ťažiskami všetkých tehiel, teda pripočítame k ***z*** súradnici všetkých týchto bodov konštantu. Táto konštanta musí byť dostatočne veľké číslo na to, aby robot nenarazil do už postavenej steny a dostatočne malé číslo aby tieto body boli v jeho pracovnom priestore. Posledný krok, ktorý zostáva je získanie rotácii z tejto postupnosti bodov, ktoré potrebujeme pre natáčanie gripperu. Každý z bodov má priradenú svoju transformačnú maticu. Ide o rotačnú transformačnú maticu okolo osi z kde vieme, že v prvom riadku a prvom stĺpci tejto matice sa nachádza hodnota kosínusu uhla, ktorý potrebujeme. Pre získanie tejto hodnoty použijeme funkciu inverznú ku kosínusu a tým získame uhol natočenia, ktorý pri tejto aplikácii potrebujeme. Pri presúvaní sa medzi bodmi nad bodom pick a nad bodom ťažiska sme zvolili hodnotu natočenia nulovú, pretože na tieto body nemáme žiadne požiadavky - gripper môže byť natočený ľubovoľne.



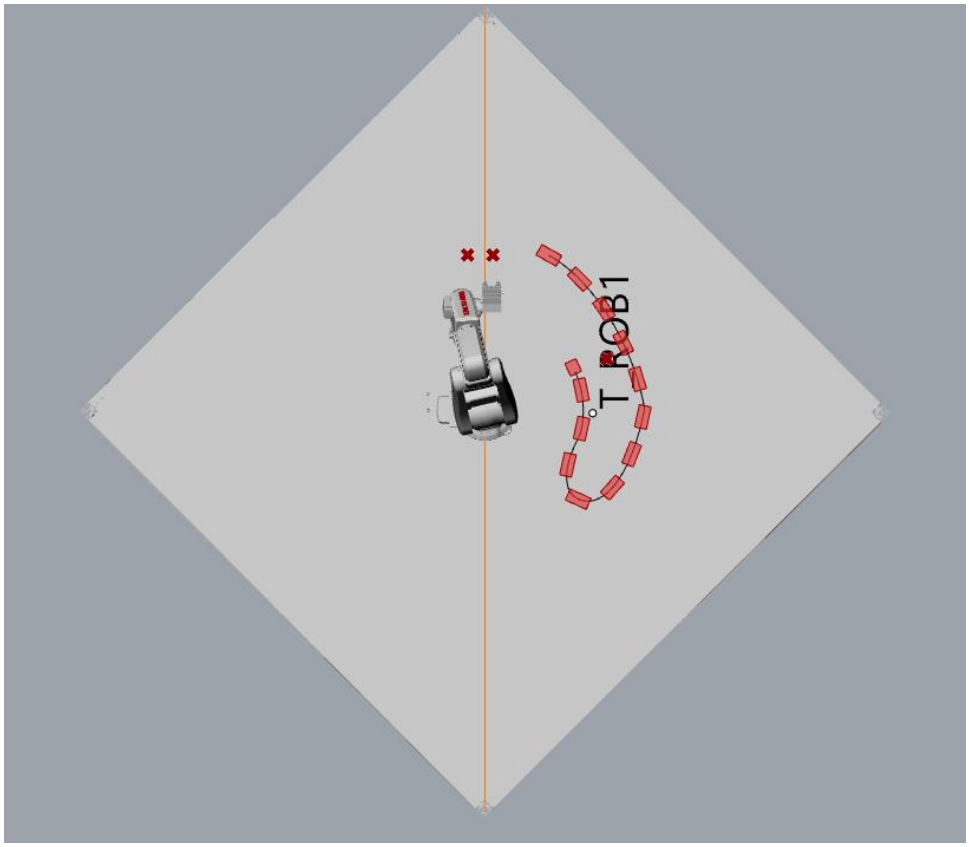
Obrázok 11 Ukážka programovania v prostredí Grasshopper - transformačné matice a získavanie uhlu natočenia

Všetko potrebné pre náš program máme pripravené. Po skompilovaní nám funkčný blok robota (obrázok číslo 5) vygeneruje rapid kód tvorený z príkazu *moveL*. Do tohto zoznamu musíme vložiť príkaz *moveJ* pred každý bod, ktorý sa nachádza v druhom kvadrante pracovného priestoru robota alebo inak povedané každý bod s ***y*** súradnicou menšou ako nula. Tento krok je pomerne jednoduchý, nakoľko poznáme poradové čísla týchto bodov a môžeme pred ne

jednoducho daný príkaz vložiť. Tento program má však isté obmedzenia, ktoré si ukážeme v nasledujúcej sekcii.

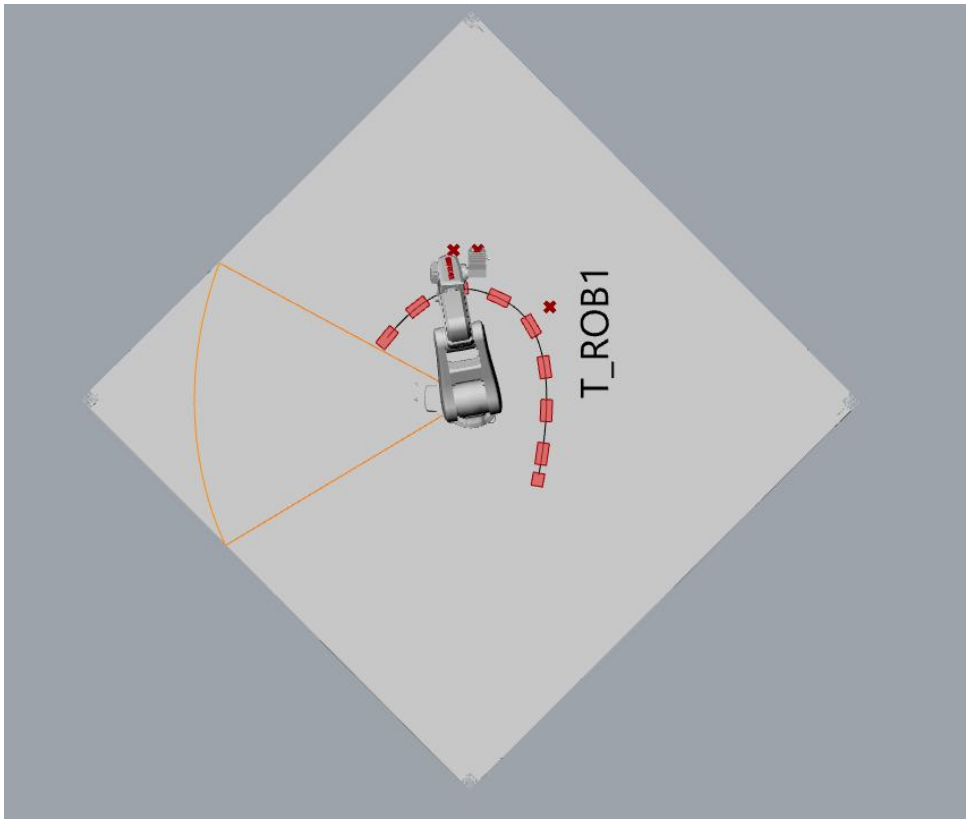
3.2 Obmedzenia pracovného priestoru a návrh riešenia

Na obrázku nižšie môžeme vidieť pracovný priestor, v ktorom pracuje tento generátor trajektórii spoľahlivo.



Obrázok 12 Spoľahlivý pracovný priestor generátora trajektórií

Ak by sme chceli zostať pri používaní *moveL* a *moveJ* príkazov, bolo by nutné pridať aspoň 3 ďalšie body a vhodne ich vložiť do postupnosti bodov, čo by bolo náročné na prevedenie. Zvolili sme si využitie príkazu *moveC*, vďaka ktorému nám na toto bude stačiť jeden jediný bod. Pracovný priestor robotického manipulátora tým zväčšíme tak, že budeme vedieť využiť celý.

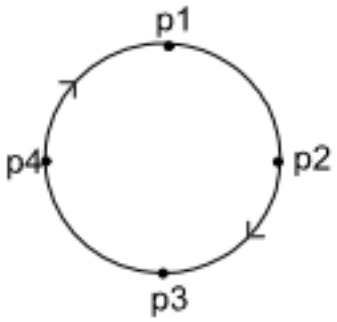


Obrázok 13 Nedosiahnuteľný pracovný priestor

Na obrázku vyššie môžeme vidieť vymedzenú časť pracovného priestoru, kde sa robotický manipulátor nedostane. Pri správnej implementácii príkazu *moveC* do rapid kódu budeme schopní teda využiť celý pracovný priestor. Príkaz *moveC* má však odlišnú syntax a používa sa iným spôsobom ako príkazy *moveL* a *moveJ*. Túto problematiku si bližšie vysvetlíme v nasledujúcej sekcii.

3.3 Syntax a použitie príkazov moveC v našom programe

Príkaz *moveC* sa používa na pohyb stredu nástroja po kružnici do nami zadaného bodu. Jeho použitie si môžeme vysvetliť na nasledujúcom obrázku.



Obrázok 14 Znázornenie moveC

Ak by sme sa chceli pohnúť do bodu **p1** pomocou príkazu *moveJ* alebo *moveL* vyzeralo by to nasledovne - *moveL* p1, v500, fine, tool1. Takýmto spôsobom by sme sa mohli postupne presúvať medzi jednotlivými vyznačenými bodmi. Nebola by však dodržaná požadovaná trajektória a taktiež by nastali problémy pri pohybe z bodu **p1** rovno do bodu **p3**, nakoľko priamočiara trajektória by viedla cez priestor ramena robotického manipulátora a mohlo by dôjsť ku kolízii.

Použitím už spomínaného príkazu ***moveC*** sa môžeme hýbať z ľubovoľného bodu kružnice do akéhokoľvek iného bodu napríklad nasledovným spôsobom: *moveC p2, p3, v500, z20, tool1;*. V tejto aplikácii sme sa rozhodli použiť tento príkaz nasledovne.

Náš namodelovaný štand je rozdelený na štyri kvadranty, rovnako ako kartézska súradnicová sústava. Pre nás bude teraz postačovať rozdelenie podľa osi *x*, ktorá vodorovne rozdeľuje štand na dve polovice pri pohľade z hora. Bod pick sa nachádza v „hornej'' časti štandu čiže vieme, že jeho *y* súradnica bude vždy kladná. Vďaka tomuto umiestneniu je robotický manipulátor schopný sa z počiatočného bodu dostať do akéhokoľvek bodu nachádzajúceho sa v kladnej časti štandu. Avšak ak by sme vygenerovali stenu pre zápornú časť teda „spodnú'' časť nášho štandu, program by pozostával z príkazov *moveL* a *moveJ* a mohlo by dôjsť k už spomínanej kolízii. Preto je potrebné vložiť jeden medzibod, ktorého *y* súradnica bude v okolí nuly a *x* súradnica bude kladná. Tento medzibod reálne nevkladáme do programu, pretože by bolo náročné správne vloženie do už vygenerovanej postupnosti bodov. Ale vkladáme ho rovno ako príkaz rapid kódu nahradením tých príkazov *moveJ*, pre ktoré platí, že po vykonaní príkazu sa koncový bod robotického ramena nachádza v záporných *y* súradniciach. Pri príkaze *moveC* je veľmi dôležité správne poradie zadaných bodov, do ktorých

sa chceme dostať, takže tento príkaz musíme upraviť podľa smeru, ktorým chceme aby sa robotické rameno hýbalo.

Najskôr si teda nájdeme indexy tých príkazov v rapid kóde, ktoré pohybujú robotickým ramenom smerom od miesta pick teda príkazy *moveJ*, ktoré nám zabezpečujú pohyb nad pozíciu tehly, ktorú chceme práve položiť. V týchto prípadoch si rozdelením textu, teda príkazu napísanom v rapid kóde získame súradnicu a natočenie daného bodu. S týmito informáciami môžeme vďaka funkciám Grasshopperu vytvoriť príkaz *moveC* automaticky poskladaním častí textu dokopy. Zostáva nám spraviť rovnaký postup len pre príkazy *moveJ*, ktoré zabezpečujú pohyb robotického manipulátora zo záporných *y* súradníc nad stenou do bodu pick, teda začiatočného bodu. Keďže indexy týchto príkazov máme uložené, môžeme použiť funkčný blok Grasshopperu - **Replace items,** ktorý nahradí prvky v zozname na zadaných indexoch.

```
MoveJ [[-304.62,221.28,112.5],[0,-0.04661,0.998913,0],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

MoveL [[-304.62,221.28,12.5],[0,-0.04661,0.998913,0],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

WaitTime 1.5;
SetDO Stavbar_Gripper_close, 0;
SetDO Stavbar_Gripper_open, 1;
WaitTime 1;
SetDO Stavbar_Gripper_open, 0;
MoveL [[-304.62,221.28,112.5],[0,-0.04661,0.998913,0],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

MoveJ [[25,440,145.8],[-0.174713,0.907209,0.375778,0.072368],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

MoveL [[25,540,45.8],[-0.174713,0.907209,0.375778,0.072368],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

IF TestDI(Stavbar_opt) THEN
WaitDI Stavbar_opt, 0;
ENDIF
WaitTime 1.5;
SetDO Stavbar_Gripper_open, 0;
SetDO Stavbar_Gripper_close, 1;
WaitTime 1;
MoveL [[25,440,145.8],[-0.174713,0.907209,0.375778,0.072368],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

MoveJ [[-192.97,332.74,112.5],[0,0.062658,0.998035,0],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;

MoveL [[-192.97,332.74,12.5],[0,0.062658,0.998035,0],[0,-1,-1,0],[0,9E9,9E9,9E9,9E9,9E9]],v150,z20,custom_tool1\WObj:=WObj0;
```

Obrázok 15 Ukážka rapid kódu v programe Grasshopper

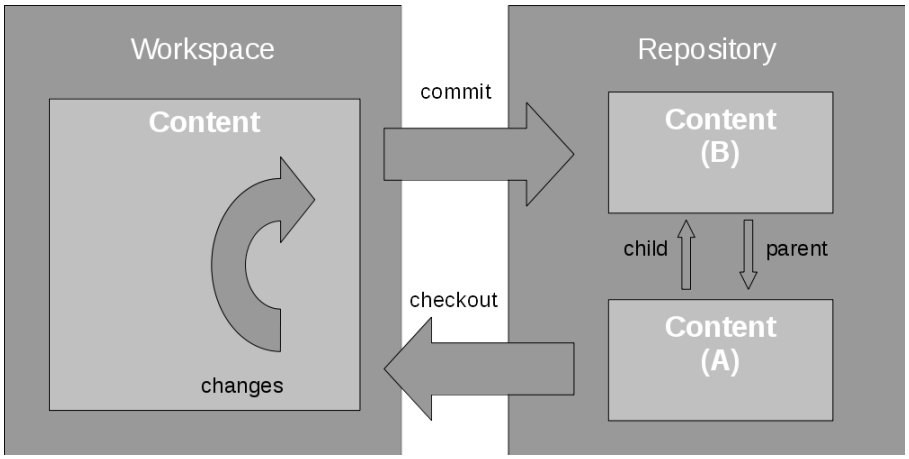
4 Git

Git je často používaný decentralizovaný repozitár pre zdrojové kódy za účelom skoordinovania práce medzi programátormi. Git bol vytvorený tvorcom Linux-u – Linusom Torvaldsom a je to open-source softvér distribuovaný pod GNU licenciou.

4.1 Opis princípu

Majme repozitár pozostávajúci zo zložiek a súborov. Tento repozitár obsahuje popri pracovnom adresári aj všetky predošlé verzie podôb súborov.

Funkcia *checkout* porovná obsah git repozitára a pracovného repozitára a zistí všetky chýbajúce súbory v pracovnom repozitári. Po vykonaní funkcie *commit* je vytvorená nová verzia obsahu z predošlej rodičovskej verzie.

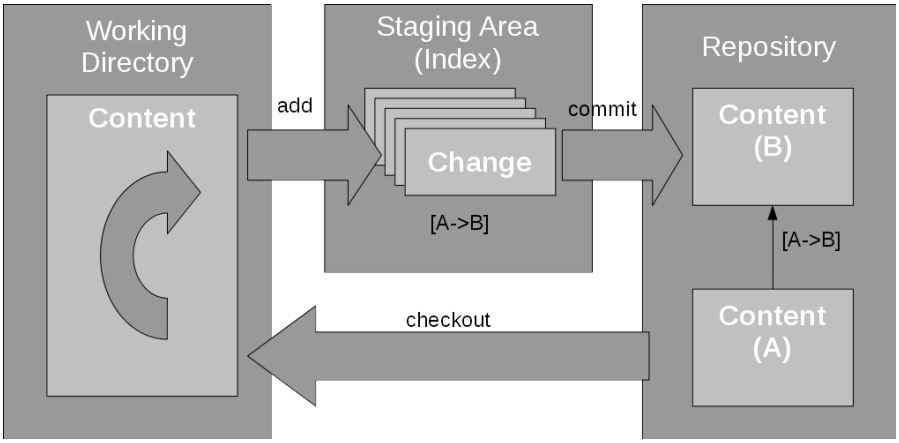


Obrázok 16 Grafické zobrazenie princípu nahrávania a kontroly

Obsah repozitára je uchovaný ako série verzií, ktoré nazývame *snapshot*, pospájané rodičovsko-detskými vzťahmi, ktoré boli vytvorené funkciu *commit*. Taktiež máme informáciu o tom čo sa medzi novou a predchádzajúcou verziu zmenilo. Túto informáciu voláme *change set*.

Na vytvorenie git repozitára sa používa príkaz *git init*. Ten vytvorí skrytý repozitár .git ale nepridá súbory do porovnávania. Na to aby sme v git-e mohli pracovať so súbormi je

potrebné použiť príkaz *git add*. Ten pridá zmeny v pracovnom úložisku do tzv. staging area, ktorú tiež voláme index. Tam sa sledujú rozdiely medzi predošlou formou súboru a aktuálnou formou súboru. Súbory, ktoré boli pozmenené sa dajú vypísať príkazom *git status*. Ak by sme chceli vidieť zmeny medzi súbormi tak je možné použi príkaz *git diff*, ktorý zobrazí zmeny súborov.



Obrázok 17 Grafické zobrazenie porovnania zmien

4.2 Ďalšie vetvy

Zatiaľ sa pri opise predpokladalo, že jeden rodič má iba jedno dieťa. To ale nemusí platiť a často ani neplatí. Ďalšiu vetvu je dobré vytvoriť ak je potrebné začať pracovať na novej funkcii ale zároveň je nevyhnutné pracovať aj na hlavnej vetve. Na toto nám v Git slúži funkcia *branch*.

Je dôležité uviesť, že na začiatku, kým nie sú vytvorené viaceré vetvy existuje len základná hlavná vetva, ktorú voláme *master*. Táto vetva je vždy považovaná za hlavnú a všetky ďalšie vetvy sú považované za vedľajšie.

Funkcia *branch* sa používa jednoducho. Ak použijeme príkaz *git branch <meno vetvy>* tak bude vytvorená nová vetva od poslednej verzie vetvy *master*. Ale ak pridáme na koniec tohto príkazu aj identifikačné číslo nejakej funkcie *commit*, čiže funkcia by vyzerala *git branch <meno vetvy> <commit ID>*, tak sa vetva vytvorí od tej verzie a tej vetvy, ktorej toto identifikačné číslo patrí.

Keď sa ukončí práca s určitou vetvou a vetva je pripravená na zlúčenie s hlavnou vetvou, použijeme príkaz *git merge <názov vetvy>*. Po zadaní tohto príkazu Git zlúči obsah hlavnej – master vetvy a vetvy, ktorú do nej chceme zlúčiť. Je zrejmé, že tu sa môžu vyskytnúť konflikty v súboroch. Existujú teda nasledujúce tri možnosti do akých sa môže zlučovanie dostať.

Prvá je nazývaná *fast forward merge*. Pri tomto zlúčení nastala taká situácia, že v hlavnej vetve nenastali zmeny od bodu kedy bola zlučovaná vetva vytvorená. V skutočnosti sa stane to, že zlučovaná vetva sa stane master vetvou.

Druhá možnosť je *bezkonfliktné zlúčenie* alebo *no-conflict merge*. Takéto zlúčenie nastane ak od bodu vytvorenia vetvy boli vykonané zmeny aj v zlučovanej, aj v hlavnej vetve ale iba v rôznych súboroch. Git potom automaticky urobí zmeny v týchto súboroch.

Posledná možnosť je *konfliktné zlúčenie* alebo *conflicting merge*. Toto zlúčenie nastane ak boli zmeny vykonané v oboch vetvách a aj v rovnakých súboroch a existujú konflikty. Tieto konflikty potom musí opraviť užívateľ.

Vyššie popisované bolo zatiaľ vykonávané iba na lokálnom úložisku. Samozrejme je možné posielať súbory aj na vzdialené úložisko. Príkazmi *git push* a *git pull* potom vieme posielať a sťahovať repozitáre na vzdialené úložisko.

4.3 Prieskum trhu

Potrebnou súčasťou so zreteľom na požiadavky vedúceho tímového projektu bol aj prieskum trhu a venujeme sa mu v tejto podkapitole.

Vzhľadom na definované nároky sme riešili dva problémy, prvým bol typ vzdialeného úložiska, druhým bola voľba klienta.

Na vzdialené úložisko sme mali ako hlavnú podmienku čo najnižšiu cenu. Uvažovali sme nad dvomi druhmi vzdialeného úložiska a to buď nad cloudovým úložiskom alebo nad tzv. self-hosted úložiskom, čo by bol server spustený v rámci školy. Nakoniec sme sa rozhodli, že budeme používať cloudové úložisko sa to z dôvodu karanténnej situácie, ktorá nastala.

Vyberali sme medzi tromi cloudovými úložiskami a to GitLab, Bitbucket a GitHub. Všetky ponúkajú možnosť používania zadarmo s nejakými obmedzeniami. Bitbucket sme

neskôr vylúčili z dôvodu obmedzenia počtu pracujúcich na jednom projekte iba na piatich, čo by pre náš konkrétny prípad stačilo ale ostatné dve možnosti takéto obmedzenie nemajú a preto sú vhodnejšie. GitLab a GitHub obe zadarmo ponúkajú všetky také funkcie aké sme vyžadovali. My sme vyžadovali iba základné funkcie ako kontrola verzii a nástenka s problémami. Výber preto ostal na preferencii a vybral sa GitHub z dôvodu už predošlých skúseností s týmto cloudovým úložiskom.

Na Git klienta sme mali taktiež ako jednu z hlavných požiadaviek cenu. Preferovali sme najlepšie zadarmo. Zároveň sme ale aj mali požiadavky také aby klient umožňoval tvorbu časovej osi a nástenku s úlohami. Vybrali sme štyri možnosti Gitea, SmartGit, Fork, GitKraken ale po bližšom preskúmaní bol jednoznačne GitKraken najlepšou voľbou pretože ponúka všetky uvedené požiadavky a zároveň je ho možné používať zadarmo pre študentov a učiteľov. Toto teda bola pre nás ideálna voľba a preto sme si ho aj vybrali.

Záver

V rámci tohto tímového projektu sa nám podarilo splniť jeho zadanie. Špecifikovali sme si úlohy, ktoré sme si postupne osvojili. Komunikáciou a rozdelením práce sme si našli vhodnú cestu akou spolupracovať, čo je v praxi veľmi dôležité.

Vyhotovili sme riešenie a následne naň aplikovali úpravy, ktoré teoreticky odstránili pozorované neduhy, či nedostatky. Tieto úpravy však nebolo možné prakticky otestovať na reálnom manipulátore vzhľadom na karanténny stav, no aplikáciou týchto úprav sme teoreticky zároveň náš systém vylepšili, čo je pozitívne.

Bohužiaľ už spomínané indukčné snímače sme nakoniec nepoužili, nakoľko sa nám nepodarilo ich vhodne umiestniť na gripper, tak aby boli funkčné.

Rovnako sme sa venovali aj možnostiam a voľbe vhodného riešenia na spoluprácu pri riešení takéhoto projektu.

Môžeme zhodnotiť, že práca na tomto projekte nám dala mnoho skúseností nielen v oblasti inteligentného plánovania pohybu robota ale aj v oblasti tímovej spolupráce.

Literatúra

[1] CHMURČIAK, Andrej – ADAMÍK, Michal: *Robotizovaná stavba steny v prostredí Grasshopper*. Bakalárska práca. 2019. 45 s

[2] SALLAY, Bálint – ADAMÍK, Michal: *Robotizované prekladanie objektov prostredí Grasshopper*. Bakalárska práca. 2019. 42 s

[3] FACHAT, Andre: *Learn the workings of Git, not just the commands* [online]. [cit. 2020-5-1]. Dostupné na internete: https://developer.ibm.com/technologies/web-development/tutorials/d-learn-workings-git/?fbclid=IwAR11zXFUwVFGxf7HyEnTL_VOM6wJJZHqK2gyzIahJ11IsNe22GXLyBS_VsA

Prílohy

Príloha A: Posudok na tímový projekt (priložený v archíve ako pdf)

Príloha B: Článok Inteligentné plánovanie pohybu robotického manipulátora, generovanie trajektórií a automatizované stavanie steny (priložený v archíve ako pdf)

Príloha C: Zdrojový kód a 3D model (externe - https://github.com/TPHugo/TimovyProjekt)