Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Martin Cvetkov

Robotovo pojasnjevanje svojih odločitev

MAGISTRSKO DELO

ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2017

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Martin Cvetkov

Robotovo pojasnjevanje svojih odločitev

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2017

University of Ljubljana
Faculty of Computer and Information Science

Martin Cvetkov

Making a robot explain its decisions

MASTER'S THESIS

SECOND-CYCLE STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

Supervisor: acad. prof. dr. Ivan Bratko

Ljubljana, 2017

Povzetek

Naslov: Robotovo pojasnjevanje svojih odločitev

Namen tega magistrskega dela je omogočiti uporabnikom in širši javnosti boljše razumevanje robotov. Natančneje, magistrsko delo si prizadeva najti način, ki bi omogočil robotom pojasniti svoje akcije in na ta način inženirjem, uporabnikom in širši javnosti olajšal razumevanje dejanj robota. Drug problem, na katerega se to delo osredotoča je povezan s tem, da naloge robota niso povsem natančno določene vnaprej. V takih primerih se naloge ne morejo rešiti s fiksnim, vnaprej določenim planom, zato v svojem magistrskem delu prikažem, kako uporabnikom omogočiti oblikovanje specifičnih, prilagojenih scenarijev, ki jih bo robot uspešno rešil.

Oba omenjena problema se rešujeta z uporabo planiranja. To pomeni, da so začetno stanje robota in cilji, ki jih je potrebno doseči, določeni s strani uporabnika. Robot nato z uporabo planiranja z regresiranjem ciljev ustvari plan za doseganje ciljev. Vsako akcijo iz plana se pojasni z uporabo algoritma. Algoritem upošteva neposredne in posredne cilje, ki jih akcije dosegajo.

Rezultat tega magistrskega dela je razvoj računalniškega programa, ki simulira ter obenem nadzoruje robota s šestimi stopnjami prostosti. Program ustvarja dinamične plane za uporabniško določene cilje ter obenem robotu omogoča, da uporabniku prek zvočnika pojasni vsako svojo akcijo.

Ključne besede

robotika, planiranje, inverzna kinematika, umetna inteligenca, obrazložitev

Abstract

Title: Making a robot explain its decisions

This master thesis is concerned with making robots more understandable to the user and broader audience. More exactly, the problem that this thesis tackles is to develop a way for robots to explain their actions in order to make it easier for engineers, users and for the audience to understand what the robot is actually trying to achieve. Another problem that this thesis tackles is that the tasks of the robots may not be precisely defined upfront. In such cases, tasks cannot be solved by a fixed, predefined task plan. Thus, I also show how to enable the user to create custom scenarios that the robot successfully resolves.

These two problems are resolved by using planning. More specifically, the initial state of the robot and the goals that need to be achieved are defined by the user. After that, the robot, using the goal regression algorithm, creates a plan in order to achieve the goals. Each action of the plan is explained using an explanation algorithm. The algorithm takes into account the immediate and indirect goals the actions achieve.

As a result of this master thesis, I developed a program that simulates and can control a robot with six degrees of freedom. The program creates dynamic plans for user defined goals. Also, the program enables the robot to explain each action it makes to the user, using the speakers.

Keywords

robotics, planning, inverse kinematics, artificial intelligence, explanation

ACKNOWLEDGMENTS

I would like to sincerely thank my mentor prof. dr. Ivan Bratko of the Faculty of Computer and Information Science at University of Ljubljana. Prof. Bratko was always ready to help whenever I ran into a trouble spot or whenever I had a question about my research or writing. He was full with ideas and suggestions on how to improve my master thesis and steer it in the right direction.

I would also like to express gratitude to my family and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Martin Cvetkov, 2017

Contents

Extended abstract (in Slovene)

Extended abstract (in Slovene) Robotovo pojasnjevanje svojih odločitev

Eden izmed ciljev umetne intelegence je razvoj robotov, ki so sposobni ustvarjati načrte in delovati samostojno, kot ljudje. Cilj tega magistrskega dela je zagotoviti robotsko roko s šestimi prostostnimi stopnjami, ki je sposobna manipulirati s predmeti z uporabo inverzne kinematike ter samostojno planirati reševanje danih nalog. Hkrati, med izvajanjem načrta, robot pojasni vsako svojo odločitev oz. akcijo, ki jo izvede.

Motiv za to magistrsko delo je to, da pogosto vidimo, kako roboti opravljajo dejanja, ki jih ljudje ne razumemo dobro. S tem, da robot razloži svoje odločitve, bodo inženirji, uporabniki in širša javnost, lahko lažje razumeli, kaj robot dejansko poskuša doseči. Drugi motiv je omogočiti, da robotovi plani za reševanje nalog niso vnaprej podani, temveč jih robot sestavi sam po potrebi. Torej, da se plani formirajo dinamično.

V tej magistrski nalogi bom pokazal, kako omogočiti, da robot samostojno ustvarja dimanični načrt, in ga med izvajanjem razlaga. Natančno pozicioniranje in manipuliranje predmetov se doseže z uporabo inverzne kinematike, inverzna kinematika se uporablja za premikanje robotske roke na željeno lokacijo in z željeno orientacijo. Načrtovanje se doseže z uporabo planiranja s sredstvi in cilji (angl. “means-ends planning“). Uporabil sem način planiranja z regresiranjem ciljev. Vsaka akcija ima svojo osnovno razlago. Po oblikovanju načrta se vsaka akcija pojasni s cilji, bodisi glavnimi bodisi vmesnimi, ki jih akcija doseže. Glavne cilje določi uporabnik. Razlaga je ustvarjena s knjižnico, ki pretvarja pisno besedilo v govor, ki se za uporabnika predvaja prek zvočnikov.

Dva scenarija (scenarij z žogami in scenarij s kockami), v tem magistrskem delu služita kot primera domen, v katerih ilustriramo delovanje planiranja in razlage robotovih akcij.

I Orodja in metodologija

Za namen tega magistrskega dela sem razvil program, ki lahko nadzoruje in manipulira robotsko roko s šestimi prostostnimi stopnjami. Poleg nadzora in manipuliranja robotske roke sem kot del magistrske naloge razvil vizualno simulacijo robotske roke, ki posnema gibanje robotske roke v realnem času. Hkrati sem implementiral program za planiranje z regresiranjem ciljev za manipuliranje robotske roke z žogami ali kockami.

Program je napisan v programskem jeziku C# v .NET framework, z uporabo integriranega razvojnega okolja “Microsoft Visual Studio 2015“. Uporabniški vmesnik je oblikovan z uporabo Windows Presentation Foundation (WPF). Tridimenzionalne podobe robotske roke so narisane v “AutoCAD 2010“, programska aplikacija za 2D in 3D računalniško oblikovanje, in dodatno izpopolnjene s programsko opremo za animacije in 3D grafiko »Blender«. 3D podobe robotske roke so importirane v program in animirane z uporabo WPF animacijskih razredov in knjižnice HelixToolkit. Robotska roka, za katero je program napisan, ima šest prostostnih stopenj in je v lasti Fakultete za elektrotehniko in informacijske tehnologije Univerze “Sv.Kiril i Metodij“, Skopje.

II Planiranje

Pri izboru, kateri planer uporabiti za zgoraj omenjena scenarija, sem se odločil uporabiti planer z regresiranjem ciljev, ker generira optimalne plane. Za regresijskega planiranja, sem uporabil algoritem opisan v knjigi “Prolog Programming for Artificial Intelligence“ (4.izdaja, Pearson Education, 2012), avtor Ivan Bratko.

III Scenarij s sortiranjem barvnih žog

V tem scenariju uporabnik lahko določi število žog in njihove barve (rdeča, zelena, modra). Žoge so lahko pozicionirane na štirih lokacijah, ki jih tu imenujemo “centri“: začetni center, v katerem so žoge prvotno postavljene, in trije “barvni centri“, kamor robot žoge premakne iz začetnega centra. Ideja je, da robot sortira žoge glede na njihove barve v ustrezne barvne centre, na način ki ga določi uporabnik. Uporabnik lahko določi pot (trajektorijo) k centru, kot želi, oziroma uporabnik lahko specificira točke, ki naj jih robotska roka obišče. To je koristno, ker lahko uporabnik glede na situacijo in ovire, s katerimi se sooča robot, določi alternativne poti za različne scenarije.

Akcije, ki se lahko izvajajo za reševanje tega tipa nalog, so: zgrabi žogo, izpusti žogo, premakni roko z dane lokacije na začetni center, premakni žogo v dani barvni center.

IV Scenarij z manipulacijo kock

V tem scenaruju lahko uporabnik določi število kock, kot tudi njihov relativni položaj glede na druge kocke in “centre“. Možni so štirje centri. Uporabnik določi začetni položaj kock in cilje, ki jih je treba doseči. Začetno stanje se določi tako, da se za vsako kocko navede njena pozicija. Vsaka kocka je lahko nameščena na eni od drugih kock ali enem od centrov. Planer poišče najkrajše zaporedje premikov kock, s katerim doseže zahtevane cilje.

V tem scenariju so možne le akcije tipa MOVE(kocka, objekt 1, objekt 2); pri tem robotska roka prestavi kocko z objekta 1 na objekt 2, pri čemer je objekt 1 in objekt 2 lahko center ali kocka.

V Razlaga akcij, ki jih izvaja robot

Motivacija za razlago akcij izhaja iz tega, da pogosto gledamo robote, kako opravljajo dejanja, ki jih ne razumemo. Razlage pomagajo inženirjem, uporabnikom in splošni publiki, da razumejo, kaj točno robot poskuša doseči.

Pri generiranju razlage vsake akcije upoštevamo regresirani cilj, ki ga skuša akcija doseči, pa tudi osnovne cilje naloge, ki jih je podal uporabnik.

Osnovna razlaga akcije se oblikuje v času planiranja. Akcija dejansko lahko izpolni več regresiranih ciljev, vendar se za osnovno razlago uporabi samo regresirani cilj, za čigar izpolnjevanje je planer to akcijo uvrstil v plan. Če regresirani cilj ni glavni, uporabniško določen cilj, bo razlaga akcije dopolnjena. Dopolnitev se opravi z algoritmom, ki deluje na naslednji način: Algoritem najprej preveri, ali je regresirani cilj glavni, uporabniško določen cilj; če je, se razlaga ne dopolnjuje. V primeru pa da ni tako, algoritem najprej preveri, če so kateri literali, ki ji ta akcija dodaja, del glavnih (uporabniško določenih) ciljev. Če obstaja tak literal, ki je glavni cilj, se razlagi doda naslednje besedilo: "This also achieves the main goal" in opis glavnega cilja. V primeru, da akcija ne vpliva na noben glavni cilj, algoritem išče glavne cilje, ki so z akcijo indirektno povezani. To pomeni, da akcija indirektno prispeva k izpolnitvi glavnega cilja. V primeru, da sta največ dva glavna cilja povezana z akcijo, se opis teh glavnih ciljev doda v razlago akcije. Če pa je z akcijo povezanih več glavnih ciljev, se k razlagi akcije dodajo le zaporedne številke teh, da razlaga ne bi bila preveč zapletena. V primeru, da je akcija povezana z vsemi glavnimi cilji, se razlagi doda naslednje besedilo: "This action affects all the main goals".

Oglejmo si primer razlage plana v svetu kock. Naj bo začetno stanje določeno takole: (on(red block, center1), on(green block, center3), on(blue block, red block), clear(blue block), clear(green block), clear(center2), clear(center4)). Naj bo seznam ciljev: on(red block, green block), on(green block, blue block), on(blue block, center2). Potem se generira plan s tremi akcijami ter njihovo razlago takole:

- Akcija move(blue block, red block, center2). Osnovna razlaga te akcije je naslednja: "Moving blue block from red block to center2 so that red block is clear." Ta razlaga se dopolni s stavkom "This also achieves the main goal blue block on top center 2". Razlaga se dopolni ker akcija posredno dosega enega izmed glavnih ciljev.

- Akcija move(green block, center3, blue block). Osnovna razlaga te akcije je: "Moving green block from center 3 to blue block so that green block on top of blue block." Ni potrebna dopolnitev razlage, zato ker akcija neposredno doseže glavni cilj: on(green block, blue block).

- Akcija move(red block, center 1, green block). Osnovna razlaga za to akcijo je: "Moving red block from center 1 to green block so that red block on top of green block." Ni potrebna dopolnitev razlage, zato ker akcija neposredno doseže glavni cilj: on(red block, green block).

VI Zaključek

Razlaga akcije je pomembna, ker omogoča inženirjem, uporabnikom in splošnemu opazovalcu, da bi bolje razumeli, kako in zakaj robot izvaja posamezne akcije. Upoštevajoč sorodno delo, ki je bilo že opravljeno na področju razlage robotskih akcij, lahko rečem, da sem z uporabo planiranja kot osnovo za razlago uspel razviti alternativen pristop razlag akcij, ki jih opravlja robot, in se tem narediti obnašanje robota bolj razumljivo za opazovalce.

Razviti program se lahko uporablja za vodenje katere koli robotske roke s šestimi prostostnimi stopnji z vrtljivimi sklepi. Treba je le določiti ustrezne inverzne enačbe in ustrezne dimenzije robotske roke.

Kot nadaljnje delo bi razvil generiranje planov z razlago za druge domene robotskih nalog. Nadaljnje raziskovanje je lahko usmerjeno na skrajšanje razlag za kompleksne plane zato, da bi razlage bile uporabniku še bolj razumljive. Algoritem planiranja z regresiranjem ciljev bi bilo mogoče nadalje optimizirati za gradnjo bolj kompleksnih planov.

Chapter 1

Introduction

One of the goals of Artificial Intelligence is to design robots that are able to make plans and act autonomously as humans. As robots get more and more involved in everyday life, it is important for people to establish certain level of trust in robots. Namely, there are two factors that significantly impact human trust: predictability and a mechanism for social exchange. However, both of them are frequently on low level or not present at all in robotic systems [1].

The goal of this master thesis is to enable a robotic arm with six degrees of freedom, to manipulate objects using inverse kinematics, to make plans and to use them to resolve a certain scenario. Also, while executing the plan, the robot should explain each decision (action) it makes.

The motivation behind this master thesis is that often we see robots taking particular actions that we do not truly understand. Research has shown that if operators do not trust the automated system, they will not use them, no matter how useful these systems might be [2, 3]. By developing a way for the robot to explain its decisions it will be easier for engineers, users and for the audience to understand what the robot is actually trying to achieve. Another reason is that in many scenarios the movements of the robots are defined upfront and not automatically constructed dynamically. In this master thesis, I show how to enable the robot to actually dynamically create a plan and explain it to the user while executing it.

The correct positioning and manipulation of objects is achieved through inverse kinematics, more exactly inverse kinematics is used to move the robotic arm to a desired location and with desired orientation. The planning was achieved by defining a planner with states and actions. The implemented goal regression planner is able to always find an optimal plan for resolving the chosen example scenarios. Each action has its own primary explanation. After creating the plan, each action is explained according to the immediate goal it tries to achieve and according to the main goals that are affected by this action. The main goals are defined by the user. The actual explanation is done through text to speech library and then broadcasted to the user by the speakers.

There are two example scenarios created for this master thesis in order to showcase the planning, robot manipulation and action explanation. In the first scenario (Figure 1.1), the user can define the number of balls and their colors (red, green, blue). The balls can be positioned at four locations that we here call "centers": one at which the balls are initially positioned, and three color centers where the balls are moved to afterwards. The path to the centers can have obstacles and that is why there is an option for defining the path that the robot should take in order to get to the appropriate center. The idea is that the robot repositions each ball according to its color to the appropriate color center, following the path that was defined by the user. Each ball color can be thought of as a different type of product. The initial center can be thought of as a place where all the products are mixed up, and each color center can be thought of as a place where all of the products of certain type are kept.

In the second scenario, the robot works with blocks (Figure 1.2). The user can define the number of blocks and their relative position according to other blocks and centers. In this scenario, there are four centers. The user can define the initial configuration of the blocks and the configuration that needs to be achieved after the manipulation of the blocks. The config-

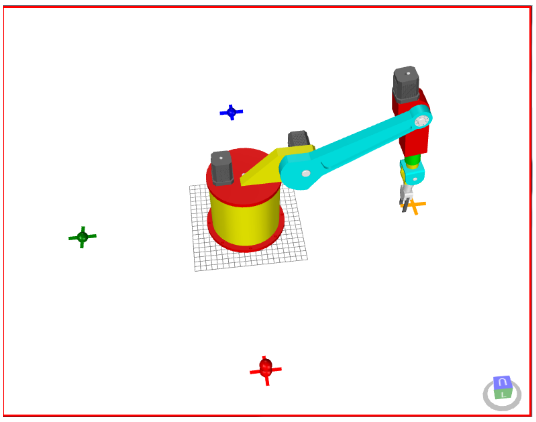


Figure 1.1: Example of the balls and the centers in the ball scenario.

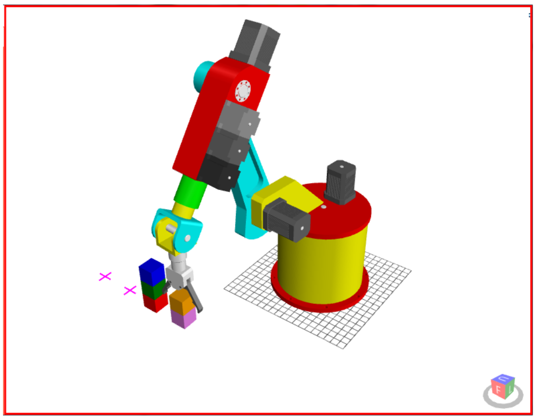


Figure 1.2: Example of the blocks and the centers in the blocks scenario.

uration consists of defining where each block is positioned. Each block can be positioned on a center or on top of another block which means that it is possible that the block can have a stack of blocks on top of it. The planner should find an optimal (shortest) sequence of block manipulations in order to achieve the configuration defined by the user i.e. the goal. The blocks can be thought of as objects in a warehouse that need to be repositioned.

The thesis continues as follows: in Chapter 2, I give a brief overview of related work, in Chapter 3, the tools and methodology used in this master thesis are explained, Chapter 4 explains how the robotic arm is moved to another position and with specific orientation. In Chapter 5, I explain how I designed the planner and how the actions are explained. In chapter 6, I give a final conclusion and review possibilities for future work.

Chapter 2

Related work

Previous research in the fields of artificial intelligence, data mining, and machine learning has sought to provide reasonable ways of having an autonomous system explain its decisions and subsequent actions.

Lomas et al. [4] in their work focus on enabling a robot to explain its actions in human understandable concepts and terms, including what action the robot took, what information it had about the environment at the time and the logic behind the decision. They do this by developing Explaining Robot Actions (ERA) system which includes a robotic world model and a query system to produce real-time human understandable answers.

To make robot's internal "thought processes" more observable to viewers, Leila et al. [5] turned to the practices of animators, who have a lot of experience in making inanimate objects come to life with readable actions. They focused specifically on pre and post action expressions of forethought and reaction as ways of helping people to understand when the robot is "thinking of acting." To test the hypotheses that these forethought and reaction cues would make robot actions more readable, they conducted a controlled experiment where they screened animated clips of a robot trying to accomplish a variety of tasks, with and without forethought or reaction, asking viewers to interpret and rate the clips. The results from their experiment show that perceptions of robots are influenced by robots showing forethought, noting the task outcome (success or failure), and showing goal-oriented reactions to those task outcomes.

Robotic actions have also been explained through visual timelines made up of action trees that are used for describing the robot's recent actions [6].

Lemon et al. (2001) [7] focused on generation of speech for interaction with users. The paper describes an interface that combines natural language commands and dialog with a computer-based map interface. This system allows the user and the robot to agree on pronoun referents without specific names, such as the command "Go here," coupled with a click on the map interface for disambiguation.

Cvetkov [8] in his thesis demonstrates the procedure for inducing inverse kinematics equations for a robot with six degrees of freedom with rotational joints which can be used to manipulate the robot. Further on, he induces the inverse kinematics equations for a concrete robot named "Makedon". I use these inverse kinematics equations in my thesis to control the movement of the robotic arm.

Chapter 3

Tools and methodology

For the purpose of the master thesis, I developed a program that is able to control and manipulate a robotic arm with six degrees of freedom. Beside controlling and manipulating the robotic arm, as a part of the master thesis I also developed a visual simulation of the robotic arm that emulates the real-time movement of the robotic arm. Also, a goal regression planner was implemented to plan the actions of the robotic arm in order to carry out the task of repositioning the objects (balls or blocks) at the right positions. While the robotic arm is carrying out the actions from the plan, it also explains them accordingly.

The whole program was written in the C# programming language under the .NET framework using the integrated development environment (IDE) "Microsoft Visual Studio 2015". I created the user interface using Windows Presentation Foundation (WPF). The three dimensional representations of the robotic arm were drawn in "Autocad 2010" which is a software application for 2D and 3D computer-aided design (CAD) and then further refined with the open source 3D graphics and animation software "Blender". The 3D representations of the robotic arm were imported into the program and animated using WPF animation classes and the HelixToolkit library. The robotic arm for which the program was created and tested on has six degrees of freedom and it is in ownership of the Faculty of Electrical Engineering and

Information Technologies at University ”Ss Cyril and Methodius”, Skopje.

3.1 C# programming language

C# is an object-oriented, general-purpose programming language created by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). C# is a high-level language that closely follows traditional high level languages C and C++ and has a strong resemblance to Java [9]. C# runs on a special environment called the Common Language Runtime (CLR). The CLR is a part of the .NET framework. The main purpose of the CLR is to enable portability, that is once a program is written in C#, it can function on different hardware platforms and operating systems, although C# programs are most commonly executed on Microsoft Windows operating systems, Windows mobile phones and other portable devices based on Windows. C# programs can also be run on other operating systems using other frameworks, but this is not officially supported by Microsoft. Because of the above mentioned reasons as well as because C# is very simple and easy to learn, today C# is one of the most popular programming languages.

When evaluating which programming language to use, I took into consideration the following programming languages: Visual Basic, C++ and C#. I did not take into consideration other languages because the library for the motion control card only supported these three languages, and writing the drivers for it myself would have been time consuming and after all this was not the goal of the thesis. Further on, I decided to work with C# because of the support for the manipulation of visual objects and because of the relative ease of creating rich user interface programs. Moreover, I am most familiar with this programming language.

3.2 .NET Framework

As previously said, the C# programming language is distributed as part of the Microsoft .NET Framework platform. More specifically, the .NET framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. .NET Framework consists of an environment for development and execution of programs that can be developed in any of the languages that are compatible with .NET. Some of those programming languages are: C#, VB.NET, C++, F# and others. .NET framework provides language interoperability (each language can use code written in other languages) across several programming languages. The key components from which the .NET framework consists are the following:

- The programming languages (C#, VB.NET, C++, F#).
- Set of development tools which turn programs written in some language into intermediate code which is then understandable for the CLR.
- Runtime environment called Common Language Runtime (CLR).It provides an environment to run all the .Net Programs. The code which runs under the CLR is called Managed Code. CLR provides memory and thread management for the programs, as well as other services such as security and error handling.
- Set of standard libraries for easier program development. There are libraries for work with databases, for communication frameworks and work with protocols such us HTTP, JSON REST, SOAP and many others.

In the thesis I use .NET version 4.5.2.

3.3 WPF and Helix Toolkit

Windows Presentation Foundation (or WPF) is a graphical subsystem of the .NET framework for rendering user interfaces in Windows-based applications. WPF uses XAML (XML-based language) to define, connect, and interact different interface elements. WPF goal is to unify the common user interface elements, such as 2D/3D rendering, fixed and adaptive documents, typography, vector graphics, runtime animation, and pre-rendered media. All of these elements can then be linked and manipulated based on various events, user interactions, and data bindings.

In the program I created for the master thesis, I use WPF for creation of the user interface, to animate the 3D visualized robotic arm, and to enable interaction with the robotic arm so that the user can have a 360 degrees view of it.

Helix Toolkit is an open source 3D library that is licensed under the MIT license [10]. It is based on .NET and more specifically the WPF platform. The main goal of the library is to make it easy to work with 3D in WPF and provide features that are not included in the standard WPF 3D visual model.

I use the Helix Toolkit library as an addition to the WPF classes in order to achieve easier animation and interaction. I also use Helix Toolkit to import ".obj" drawings because WPF does not provide a way to do that.

3.4 Autocad and Blender

AutoCAD is a software application for 2D and 3D computer-aided design (CAD) and drafting.

Blender is the free and open source 3D creation suite. It supports modeling, creation and animation [11].

I use AutoCAD to draw the initial three dimensional visualization of the robotic arm. AutoCAD enables drawing with high precision, as a result of that the drawings are with the original dimensions of the robotic arm. After the initial drawings are created, I use Blender in order to improve them and then export the drawings as ".obj" files which can then be imported in the program by the Helix Toolkit library. After importing, the drawings can be manipulated in 3D using WPF and Helix Toolkit.

3.5 SQLite database

SQLite is an in-process library that implements a transactional SQL database engine that is self-contained, does not need a server and needs no configuration. The code for SQLite is public and as such is free for use in private or commercial purposes. SQLite, by some accounts, is the most widely deployed database in the world, with huge number of applications and a lot of high-profile projects [12].

SQLite is an embedded SQL database engine. In comparison with most of the other SQL databases, SQLite does not have a separate server process. SQLite works by reading and writing directly to ordinary disk files. A complete SQL database, including tables, indices, triggers, and views, is contained in a single disk file. Another advantage is that the database file format is cross-platform, which means that the database can be freely transfered between 32-bit and 64-bit systems or between any architectures. These features make SQLite a popular choice as an Application File Format.

SQLite is a compact library. When all of the features are enabled, the library size can be less than 500KiB. If the optional features are not included, the size of the SQLite library can be reduced to be under 300KiB. SQLite, if necessary, can also be made to run in minimal stack space of 4KiB and minimal heap space of 100KiB. Because of this, SQLite is very popular database engine choice on memory constrained devices such as cellphones, PDAs, and MP3 players. Of course, there is a tradeoff between memory usage and speed, that is SQLite runs faster if it is given more memory to work with. Nevertheless, the performance is satisfactory even in low memory environments.

The main advantages of SQLite are:

- It is serverless, which means that there is no need to install anything at the user's system.

- It consists of one file that can be copied on any system and architecture.

The main disadvantage of SQLite is that it does not provide user management which means that it does not provide the ability to set access privileges to the database and tables.

Therefore, it is best to use SQLite in embedded applications, applications that need portability and one user applications. SQLite is not recommended for usage in Multi-user applications.

In the application developed for this master thesis, I use SQLite database in order to enable the user to create a path as a series of points that the robotic arm should follow.

Chapter 4

Robotic Arm Control

The robotic arm for which the program was developed has six degrees of freedom. The sixth axis as an end effector has a gripper attached to it. The purpose of the gripper is to catch and release the objects. The name of the robotic arm is Makedon. The robot did not come with any software for manipulation and that is why I had to develop one.

4.1 Stepper motors

Stepper motors (Picture 4.1) are DC motors that move in discrete steps. They consist of multiple coils that are organized in groups called ”phases”. If each phase is energized in sequence the motor will rotate, one step at a time. The step size depends from the construction of the motor, and can vary from less than one degree to values in the range of one to ten degrees and even to hundreds of degrees.

According to their power, stepper motors can be divided as follows:

- Low power motors (between 10mW and 1W) usually used in measurement equipment.
- Average power motors (between 1W and 100W), for example used in printers.

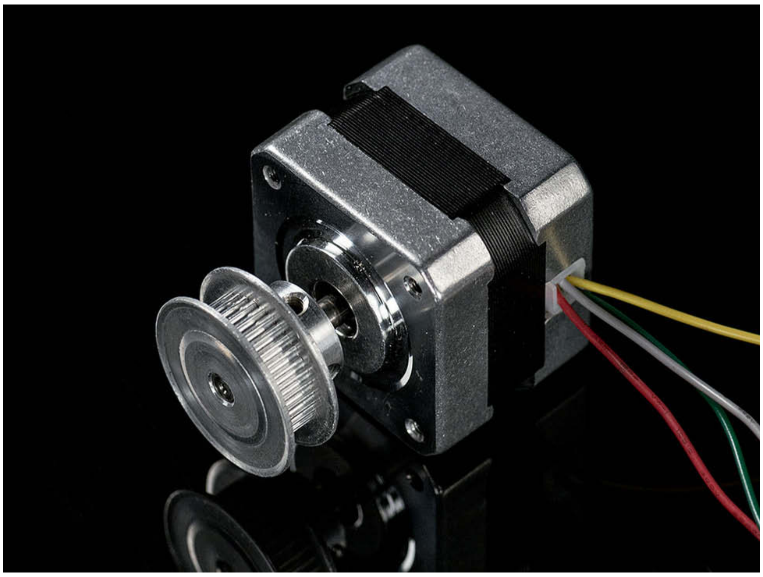


Figure 4.1: Stepper motor

- High power motors (between 100W and 1KW) which are used in machines that work with high load.

Depending on the way the phases are energized, stepper motors can be unipolar or bipolar. Unipolar stepper motors always energize the phases in the same way, while bipolar actually reverse the current flow through the phases.

The advantages of stepper motors are:

- Since they move in repeatable steps, stepper motors are very good at precise positioning.
- Stepper motors enable precise rotational speed control, which is very important in robotics.

The disadvantage of using stepper motors is that they are inefficient and that their power consumption is independent of the load. Also, stepper motors provide no feedback for the position. This can result in unexpected behavior and imprecision.

For each of the axes of the robotic arm used in this master thesis, there is a stepper motor that is responsible for moving the corresponding axis.

4.2 Sensors

A sensor is an object that detects events or changes in its environment and provides a corresponding output. Sensors in robotics are used to provide information about the inner state of the robot, as well as the position of the robot in correspondence to the outer environment. The robot controller knows the state of each joint of the robot by the information provided from the sensors. Robots can be equipped with different kind of sensors, such as: color sensors, touch sensors, cameras and others.

4.2.1 Color sensors

Color sensors are used to detect the color of a surface. The sensors cast light (red, green and blue) on the objects to be tested, calculate values from the reflected radiation and compare them with previously stored reference colors. If the color values are within the required range, a corresponding color is detected.

4.3 Rotary encoders

A rotary encoder is an electro-mechanical device that converts the angular position or motion to an analog or digital code.

There are two main types of rotary encoders: absolute and incremental (relative).

The output of absolute encoders indicates the current position of the shaft, making them angle transducers.

The output of incremental encoders provides information about the motion of the shaft, which is then usually further processed into information such as speed, distance and position.

Rotary encoders are used in many applications that require precise shaft unlimited rotation—including industrial controls, robotics, special purpose photographic lenses, computer input devices (such as optomechanical mice and trackballs), controlled stress rheometers, and rotating radar platforms.

Because the stepper motors do not have feedback for their actual position, rotary encoders are used to correct the position of the motors.

4.4 Robot manipulator

The body of the robotic arm is shown in Figure 4.2. It consists of six axes. There are connections between the axes as follows:

- Connection between axis 1 and axis 2 with length of 196 millimeters
- Connection between axis 2 and axis 3 with length of 400 millimeters
- Connection between axis 2 and axis 4 with length of 13 millimeters
- Connection between axis 3 and axis 5 with length of 320 millimeters

The axes and the links between them are also shown in Figure 4.2.

All the equations in the following subsections are developed by Mihail Cvetkov [8] using the Denavit-Hartenberg representation of kinematic equations. Some of the symbols used in the equations are constants and have the values as shown in Table 4.1.

	θ_i	d_i	a_i
Axis 1	θ_1	325	196
Axis 2	θ_2	13	400
Axis 3	θ_3	0	0
Axis 4	θ_4	320	0
Axis 5	θ_5	0	0
Axis 6	θ_6	193	0

Table 4.1: Values of the symbols used in the kinematic equations.

The symbols present in the inverse kinematics equations in the following sections and their meanings are given in the next listing:

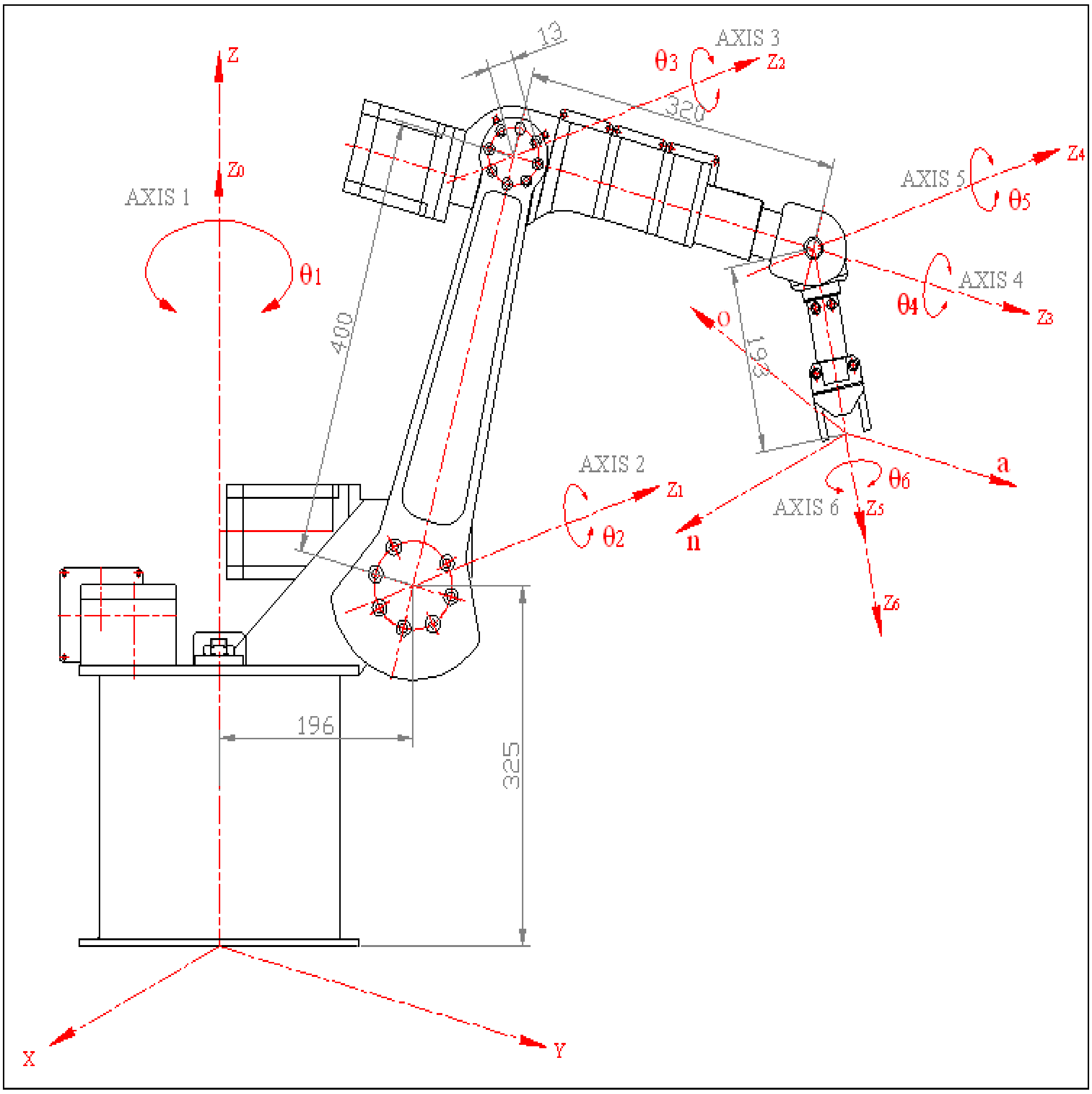


Figure 4.2: Robotic arm with six degrees of freedom. Used in this master thesis.

- Px, Py, Pz - these three symbols represent the desired x,y and z coordinate of the end effector. These are input parameters and are defined by the user.

- [nx ,ny, nz], [ox, oy, oz], [ax, ay, az] - represent the projections of the three unit vectors and are used for setting up the orientation of the end effector. The three unit vectors are mutually perpendicular and they are n (normal), o (orientation) and a (approach) vector. These parameters are input parameters and are defined by the user.

- Si, Ci where i can be from 1 to 6 represent the sine and cosine of the angle θ_i correspondingly.

- S23, represents the sine of the sum of $\theta_2 + \theta_3$

- C23, represents the cosine of the sum of $\theta_2 + \theta_3$

4.4.1 Axis 1

Axis 1 is moved by a stepper motor of type 34H2120-60-4Ak. It is produced by the Chinese company "MS Motor". Rotating Axis 1 for one degree corresponds to the stepper making 250 steps:

$$number\_of\_steps = 250 * number\_of\_degrees\_to\_rotate \tag{4.1}$$

In order to calculate the degrees Axis 1 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$\begin{gathered}
degrees = atan((d_2(Px - axd_6) - ((d_2(Px - axd_6))^2 - ((Px - axd_6)^2 \\
+ (Py - ayd_6)^2)(d_2^2 - (Py - ayd_6)^2))^{0.5})/((Px - axd_6)^2 + (Py - ayd_6)^2) \\
/(1 - ((d_2(Px - axd_6) - ((d_2(Px - axd_6))^2 - ((Px - axd_6)^2 + (Py - ayd_6)^2) \\
(d_2^2 - (Py - ayd_6)^2))^{0.5})/((Px - axd_6)^2 + (Py - ayd_6)^2))^{0.5}))
\end{gathered} \tag{4.2}$$

4.4.2 Axis 2

Axis 2 is moved by a stepper motor of type 4H2160-50-8A. It is produced by the Chinese company "MS Motor". Rotating Axis 2 for one degree corresponds to the stepper making 112.5 steps:

$$number\_of\_steps = -112.5 * number\_of\_degrees\_to\_rotate \tag{4.3}$$

In order to calculate the degrees Axis 2 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$\begin{gathered}
degrees = atan(((Pz - azd_6 - d1)((a_2^2 - d_4^2 + (C1(Px - axd_6) \\
+ S1(Py - ayd_6) - a_1)^2 + (Pz - azd_6 - d_1)^2)/(2a_2)) - (C1(Px- \\
axd_6) + S1(Py - ayd_6) - a_1)((C1(Px - axd_6) + S1(Py - ayd_6) - a_1)^2 \\
+ (Pz - azd_6 - d_1)^2 - ((a_2^2 - d_4^2 + (C1(Px - axd_6) + S1(Py \\
- ayd_6) - a_1)^2 + (Pz - azd_6 - d_1)^2)/(2a_2))^2)^{0.5})/((C1 \\
(Px - axd_6) + S1(Py - ayd_6) - a_1)^2 + (Pz - azd_6 - d_1)^2))/(1 \\
(((Pz - azd_6 - d_1)((a_2^2 - d_4^2 + (C1(Px - axd_6) + S1(Py- \\
ayd_6) - a_1)^2 + (Pz - azd_6 - d_1)^2)/(2a_2)) - (C1(Px - axd_6) + S1 \\
(Py - ayd_6) - a_1)((C1(Px - axd_6) + S1(Py - ayd_6) - a_1)^2 + (Pz- \\
azd_6 - d_1)^2 - ((a_2^2 - d_4^2 + (C1(Px - axd_6) + S1(Py - ayd_6) \\
- a_1)^2 + (Pz - azd_6 - d_1)^2)/(2a_2))^2)^{0.5})/((C1(Px - ax \\
d_6) + S1(Py - ayd_6) - a_1)^2 + (Pz - azd_6 - d_1)^2))^2)^{0.5}
\end{gathered} \tag{4.4}$$

4.4.3 Axis 3

Axis 3 is moved by a stepper motor of type 34H295-48-8A. It is produced by the Chinese company "MS Motor". Rotating Axis 3 for one degree corresponds to the stepper making 83.333 steps:

$$number\_of\_steps = -83.33 * number\_of\_degrees\_to\_rotate \tag{4.5}$$

In order to calculate the degrees Axis 3 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$\begin{gathered} degrees = atan((C2(C1(Px - axd_6) + S1(Py - ayd_6) - a_1) + S2 \\ (Pz - azd_6 - d_1) - a_2)/(C23(C1(Px - axd_6) + S1(Py - ayd_6) - a_1) + \\ S23(Pz - az * d_6 - d_1))) \end{gathered} \tag{4.6}$$

4.4.4 Axis 4

Axis 4 is moved by a stepper motor of type 23H 252-062-4A. It is produced by the Chinese company "MS Motor". Rotating Axis 4 for one degree corresponds to the stepper making 177.777 steps:

$$number\_of\_steps = -177.777 * number\_of\_degrees\_to\_rotate \tag{4.7}$$

In order to calculate the degrees Axis 4 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$degrees = atan(S1ax - C1ay)/(C23C1ax + C23S1ay + S23az) \tag{4.8}$$

4.4.5 Axis 5

Axis 5 is moved by a stepper motor of type 23H 252-062-4A. It is produced by the Chinese company "MS Motor". Rotating Axis 5 for one degree corresponds to the stepper making 231.8524 steps. However, Axis 4 affects the movement of Axis 5 and has to be taken into consideration when evalauting the number of steps the stepper motor for Axis 5 has to make. We can see that in the following equation:

$$\begin{aligned} number\_of\_steps = 231.8524 * degrees\_to\_rotate\_axis\_5 \\ + number\_of\_steps\_axis\_4 \end{aligned} \tag{4.9}$$

In order to calculate the degrees Axis 5 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$\begin{aligned} degrees = atan(-(C23C1ax + C23S1ay + S23az) \\ /((S23C1ax + S23S1ay - C23az)/C4 \end{aligned} \tag{4.10}$$

4.4.6 Axis 6

Axis 6 is moved by a stepper motor of type 23H 252-062-4A. It is produced by the Chinese company "MS Motor". Rotating Axis 6 for one degree corresponds to the stepper making -177.777 steps:

$$\begin{aligned} number\_of\_steps = -177.777 * degrees\_to\_rotate\_axis\_6 \\ + num\_steps\_axis\_5 - 1.304 * 231.8524 * degrees\_axis\_5 \end{aligned} \tag{4.11}$$

In order to calculate the degrees Axis 6 needs to rotate in order to move the robotic arm to a certain position in space, the following kinematic equation is used:

$$\begin{aligned} degrees = atan((S4C23C1 - C4S1)nx + (S4C23S1 + C4C1)ny + S4 \\ S23nz)/((S4C23C1 - C4S1)ox + (S4C23S1 + C4C1)oy + S4S23oz) \end{aligned} \tag{4.12}$$

As an end effector the sixth axis has a gripper which is used for manipulating objects.

4.5 Manipulating and controlling the robotic arm

Controlling and manipulating the robotic arm is done by a program written in the C# programming language. The program uses inverse kinematics equations in order to calculate the joint configuration of the robotic arm.

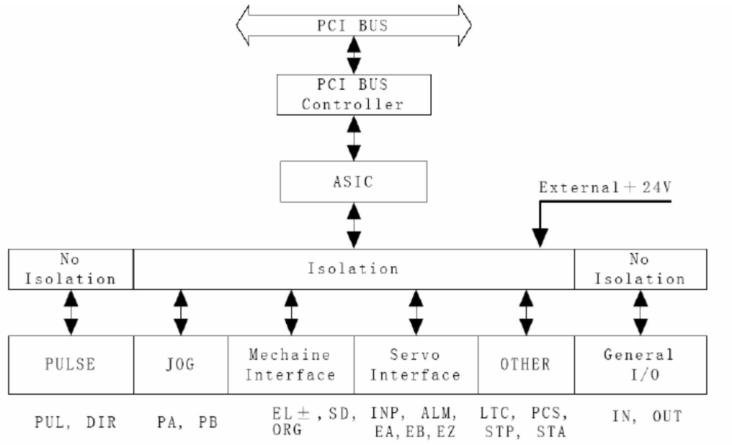


Figure 4.3: Function block diagram of the DMC5400

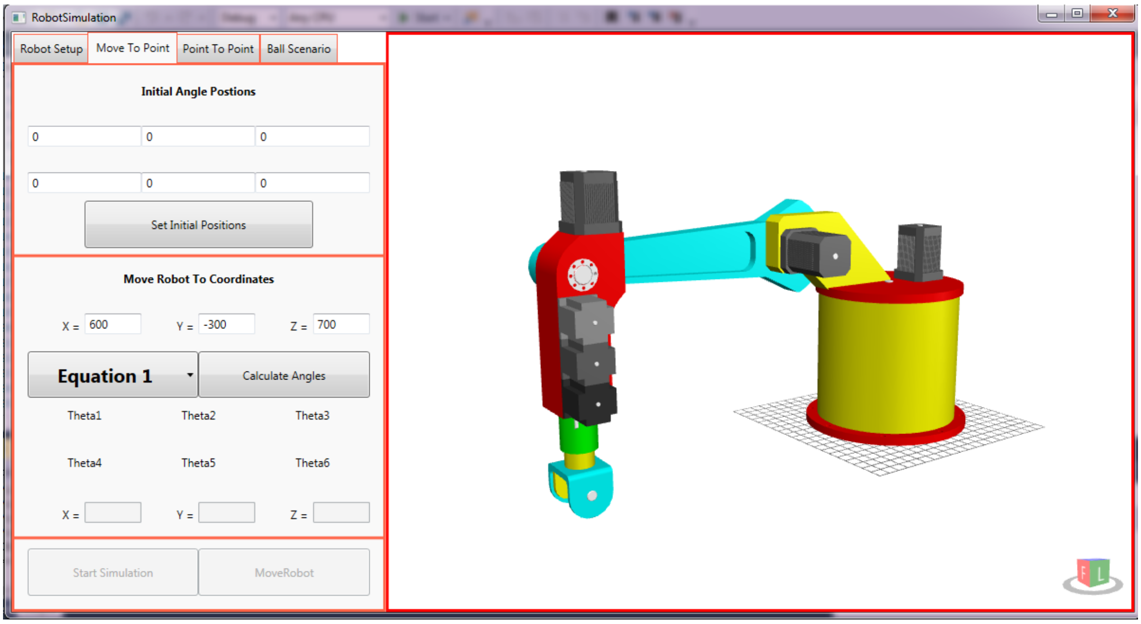
The configuration consists of the angles each axis needs to make in order to get to the desired position. This information is then transmitted to the motion control card which in turn gives orders to the stepper motors.

4.5.1 Motion control card

Manipulating and controlling the robotic arm is done using two 32-bit PCI cards DMC5400 produced by the Chinese company Leadshine. Each card is able to control up to four axes with PCI interface. The card can generate pulse control signal to control stepping and digital servo systems. Figure 4.3 shows the function block diagram of the DMC5400 card.

4.5.2 Setting initial position of the robotic arm

When the program is started, the first thing that needs to be done is to set the initial position of the robot. The user can do this through the ”Robot Setup” tab of the program. When the robot, or the program, is started the robot does not know in which state it is. More precisely, the stepper motors do not have a memory at which step they are, that is at which state the robot was turned off in its last usage.



Initial Angle Postions
0 0 0
0 0 0

X = 600 Y = -300 Z = 700

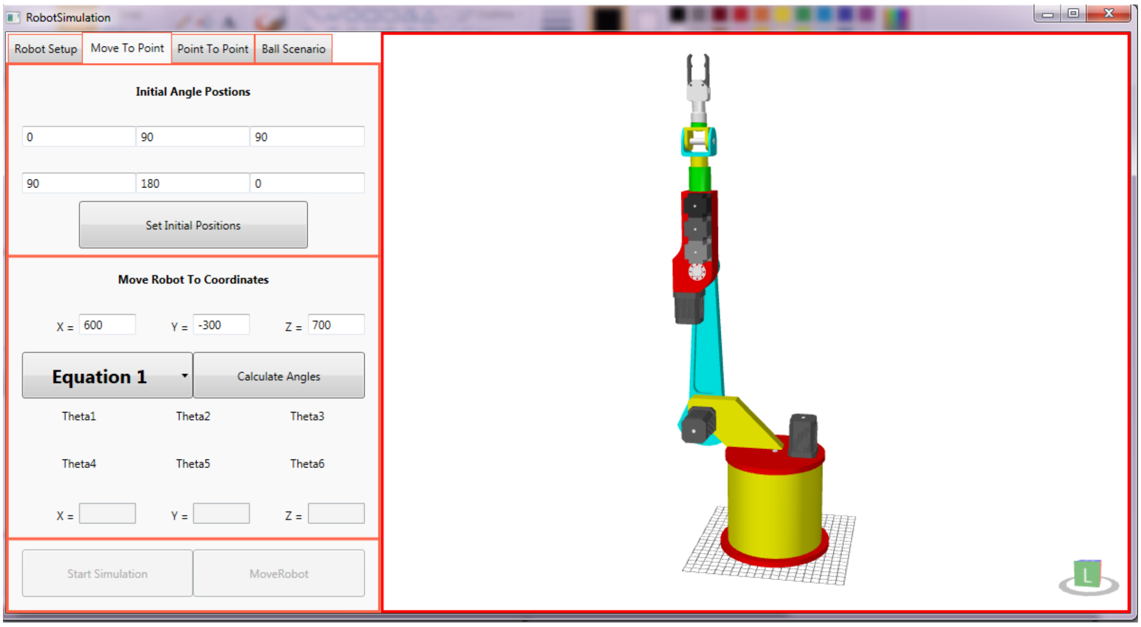
X = Y = Z =

Figure 4.4: Reference position of the robotic arm and the configuration of the axes.

The idea is that the user brings the robotic arm in a certain state at which it is straight forward to calculate the number of steps that the stepper motors have achieved, and then forward these calculations to the stepper motors through the motion control card. After this initialization, the robotic arm knows at which position and which state it is and no more interfering from the user is needed.

The kinematic equations calculate how many degrees each axis should move from the reference position that is shown in Figure 4.4. As we can see from the figure, this position is not achievable in reality and as such can not be used as an initial position. Another thing that is noticeable here is that the angle for all of the axes is zero degrees. This is because the kinematic equations use this position of the robot as the reference position and all the kinematic equations are done in regard to this position.

After evaluating the possible configuration of the axes, I decided to use the state of the robot in which all of the axes are straight up. The angles



Initial Angle Postions

0 | 90 | 90
90 | 180 | 0

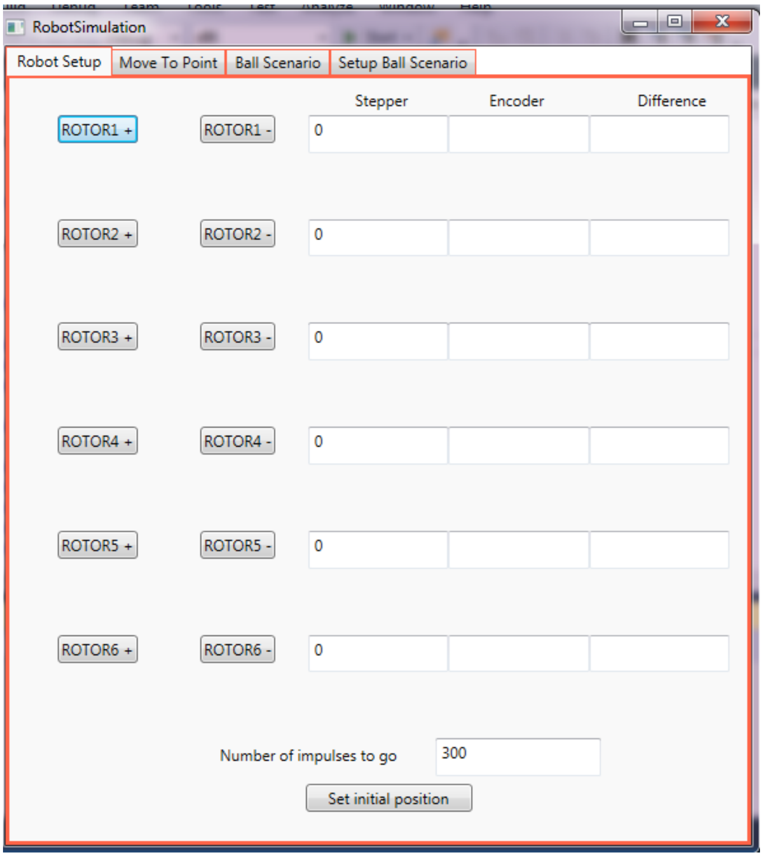
Move Robot To Coordinates

X = 600 Y = -300 Z = 700

Figure 4.5: Initial position of the robotic arm and the configuration of the axes.

of each of the axes (axis1: 0 degrees, axis2: 90 degrees, axis3: 90 degrees, axis4: 90 degrees, axis5: 180 degrees, axis6: 100 degrees) and the robot are shown in Figure 4.5. Using the equations (4.1), (4.3), (4.5), (4.7), (4.9), and (4.11), we set the stepper motors with the following steps: 0, -10125, -7500, 0, 41733, -54420.

Setting the initial position is done through the ”Robot Setup” tab of the program. The user interface for this tab is shown in Figure 4.6. There are two buttons for each axis. One is for moving the axis in the positive direction and the other for moving the axis in the negative direction. By holding each button, the corresponding axis moves in the corresponding direction for the number of steps that is defined in the text button below all the buttons. There are also three text boxes for each axis. They show the state of the stepper motor, the actual position of the stepper motor read by the encoders and the third text box is used for showing the difference between the state of the stepper motor and the actual position of the stepper motor. After



		Stepper	Encoder	Difference
ROTOR1 +	ROTOR1 -	0		
ROTOR2 +	ROTOR2 -	0		
ROTOR3 +	ROTOR3 -	0		
ROTOR4 +	ROTOR4 -	0		
ROTOR5 +	ROTOR5 -	0		
ROTOR6 +	ROTOR6 -	0		

Number of impulses to go 300

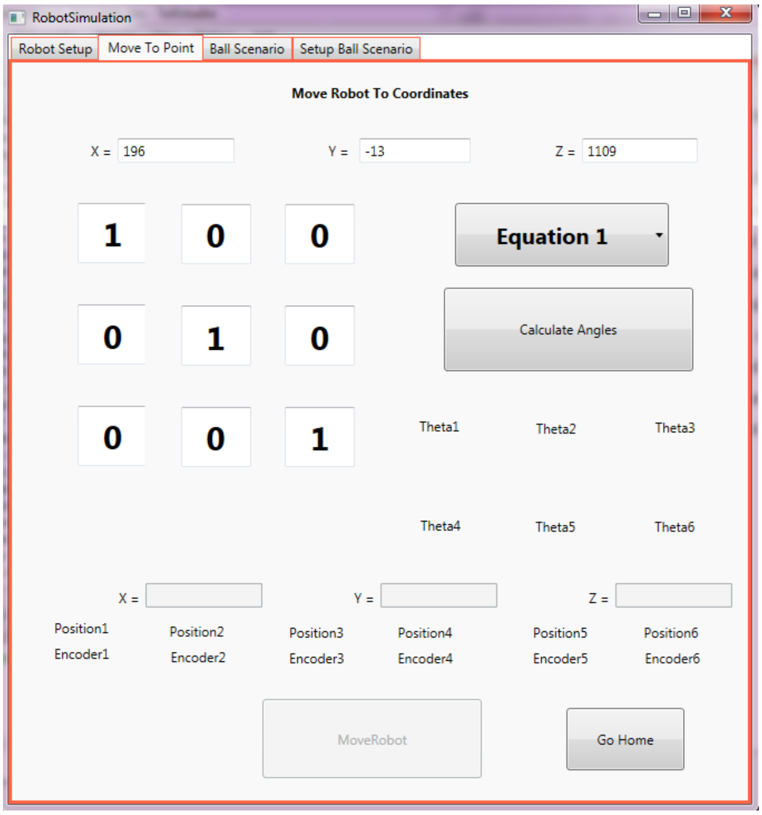
Figure 4.6: User interface of the tab for setting the initial position of the robotic arm.

we are satisfied with the position of the robotic arm, we can set it as initial through the button "Set Initial Position". This indicates setting the values of the stepper motors to 0, -10125, -7500, 0, 41733, -54420 accordingly.

4.5.3 Moving the robotic arm to a point in space

Moving the robotic arm to a certain point in space and specifying the orientation of the end effector (the gripper) is done through the "Move To Point" tab in the program.

The tab shown in Figure 4.7 consists of multiple user interface elements. The three text boxes in the top are used in order to specify the position where



X = 196
Y = -13
Z = 1109

1	0	0
0	1	0
0	0	1

Figure 4.7: User interface of the tab for moving the robotic arm to a certain position with a certain orientation.

the robotic arm should move to. As it was said in the previous chapter, in order for the robotic arm to move to the correct position, the initial position of the robot has to be correctly set.

Below the text boxes for the position of the robot, there are three rows of three text boxes for setting up the orientation of the end effector of the robotic arm. Each set of text boxes is used for setting up a vector. Depending on how we set these vectors, the end effector will be oriented differently. These vectors also affect the possible solutions to getting to the position we indicated previously. For example, if the user sets the vectors as following: (1, 0, 0), (0, 1, 0), (0, 0, 1) the end effector (gripper) will be oriented upwards as shown on figure Figure 4.8 and if the user sets the vectors as : (1, 0, 0),

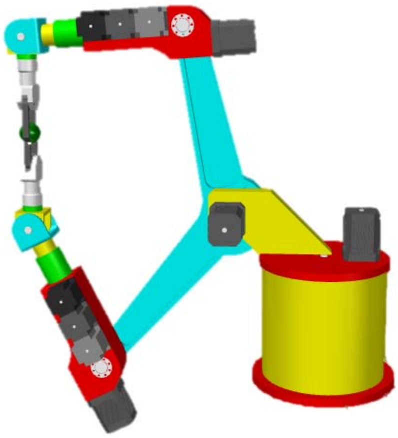


Figure 4.8: The robotic arm grasping a ball with the gripper in upwards and backwards position.

(0, 1, 0), (0, 0, -1) the end effector (gripper) will be oriented downwards as shown in Figure 4.8. This means that the robotic arm will get to the same position, but with different orientation. Changing these vectors will result in different orientations of the gripper.

Next, using the combo box the user can pick a different solution to the kinematic equations. The kinematic equations (4.2), (4.4), (4.6), (4.8), (4.10), and (4.12) have more than one possible solution, to be exact they have 16 different solutions. For a given position and orientation, some of these solutions may be valid and some might not. This makes sense because the robotic arm can usually get to a point in more than one way. Some of the solutions could be possible in theory, but due to the physical limitations of the robotic arm they may not be achievable in practice. To be more specific:

- Axis 1 can move from -135 to 135 degrees

- Axis 2 can move from 0 to 160 degrees

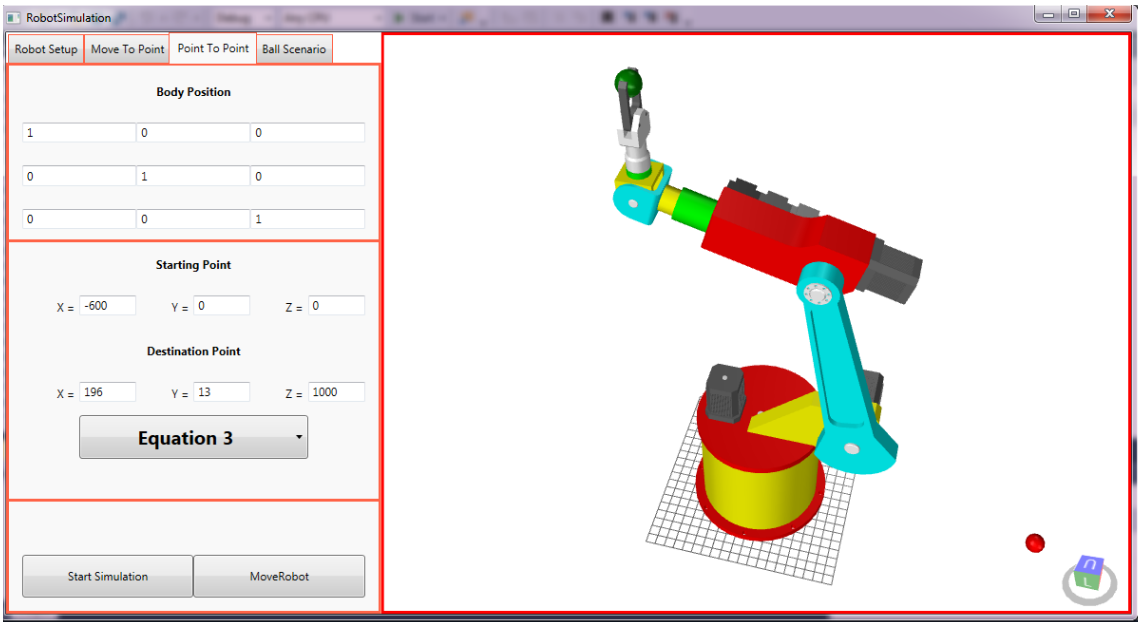
- Axis 3 can move from -55 to 135 degrees
- Axis 4 can move from 0 to 360 degrees
- Axis 5 can move from 90 to 270 degrees
- Axis 6 can move from 0 to 360 degrees

After selecting the appropriate solution (by default, solution 1 is selected), the user can click on the "Calculate Angles" button. By clicking on this button, the program calculates the configuration of the joints, that is the angles of each of the axes using the previously selected solution.

If the program found a configuration for the user selected solution then the configuration is displayed in the six labels below the buttons. Each label is for one axis. The validity of the solution is evaluated using forward kinematics, that is after getting the angles of each of the axes, we compute where the robotic arm will end up by using the equations for forward kinematics. If the desired position and the position calculated from the forward kinematics equations are the same, the user knows that the right orientation was calculated.

If the user selected solution is not achievable, the program continues on the next one until it finds a correct solution, or until it runs out of solutions in which case a message box is displayed that informs the user that the robotic arm can not get to that position with that orientation of the end effector. The reason for not finding a solution could be that the point in question is out of reach of the robotic arm and in order to resolve this, the user could define another point or try with different orientation of the robotic arm.

After these calculations are done, the user can click on the button "Move Robot" which will actually move the robot to the desired position and with the desired orientation.



Body Position

1	0	0
0	1	0
0	0	1

Starting Point

X = -600 Y = 0 Z = 0

Destination Point

X = 196 Y = 13 Z = 1000

Equation 3

Figure 4.9: User interface of the "Point to Point" tab.

4.5.4 Moving the robotic arm from one point to another

Moving the robotic arm from one point to another can be done through the "Point to Point" tab of the application shown in Figure 4.9. From the user interface of the tab, we can notice that the user can:

- Specify the orientation of the end effector (gripper) of the robotic arm as explained in the previous subsection.
- Define the starting point of the robotic arm and the destination point of the robotic arm. This can be done through the text boxes for starting point and destination point.
- Define the solution with which the user wants the action to be executed. If a solution is not valid, the program will go on to the next one until it finds a valid solution.

The simulation starts by clicking the ”Start Simulation” button while the robotic arm actually moves by clicking the ”Move Robot” button. After clicking one of these buttons, first the robot moves from its actual position to the starting point. In order to get to the starting point, the application goes through 16 inverse kinematic solutions to find the right configuration of the joints in order to get to the starting point. Then, by using forward kinematics, it calculates where the robotic arm will be if it uses the corresponding solution. The solution that is closest to the starting point is selected. After the robotic arm gets to the starting point, it uses the user selected solution in order to get to the destination point.

Chapter 5

Planning and explaining the robot actions

The main task of this master thesis was to design a way by which the robot would make a plan and while executing the plan to explain the decisions it made. In this chapter, I will first give a theoretical overview of planning in artificial intelligence and after that the color ball scenario and block scenario will be thoroughly explained.

5.1 Planning

5.1.1 Representing actions

According to I.Bratko [13] in planning, available actions are represented in a way that enables explicit reasoning about their effects and their preconditions. This can be done by stating, for each action, its precondition, its add-list (relationships the action establishes) and delete-list (relationships the action destroys). Such representations are referred to as STRIPS-like representations.

The explanations of means-end planning mechanisms presented in the sections bellow are abbreviated descriptions from the book Prolog Programming for Artificial Intelligence (4th edition) by Ivan Bratko.

5.1.2 Deriving plans by means-ends analysis

Means-ends derivation of plans is based on finding actions that achieve given goals and enabling the preconditions for such actions. This analysis considers the means as available actions while the ends present the goals to be achieved. Because of the search among alternative actions, combinatorial complexity is quite common in planning. The means-ends planning has some advantages over the simple state-space search. Namely, a means-ends planner will only consider those actions that refer to the achievement of current goals, and then the actions that set up the preconditions of such actions. Compared to this, state space search would produce all possible actions in the initial state and in all states to come. Consequently, the search space of the means-ends planner may be substantially smaller than that of the state-space search. However, it might happen that in the original STRIPS planning the means-ends mechanism of planning does not suggest all relevant actions to the planning process and the reason for this lies in its locality. This means that the planner only takes into account those actions that belong to the current goal and neglects other goals up until the achievement of the current goal is done. Thus, it does not generate plans in which actions that relate to different goals are interleaved, unless they happen by accident. The literature explains this limited planning mechanism through linear planning in which the goals are completed one by one in a linear order. By enabling interaction between different goals, it is achieved that optimal plans are within the search scope. This is done with the use of goal regression.

One downside of the simple planner is that sometimes it involves actions in the plan which terminate already completed goals. In order to prevent this, additional mechanism known as goal protection, is build in to the planner. The role of the mechanism is to keep track of the goals that are already achieved and to keep away actions that damage these 'protected' goals. However, the use of goal protection is restricted since it is not always possible to develop a plan without temporarily impairing a protected goal and re-achieving it afterwards.

5.1.3 Goal regression

Goal regression is a process that determines which goals have to be true before an action, to ensure that given goals are true after the action. Better efficiency can be reached by observing that some combinations of goals are incompatible and thus can never be achieved. Planning with goal regression typically involves backward chaining of actions.

5.1.4 Combining regression planning with best-first heuristic

Means-ends planning involves search through the space of relevant actions. The usual search techniques hence can be used in planning as well: depth-first, breadth-first, iterative deepening and best-first search.

To decrease search complexity, domain-specific knowledge can be used at several stages of means-ends planning, such as: which goal in the given goal-list should be attempted next; which action among the alternative actions should be tried first; heuristically estimating the difficulty of a goal-list in the best-first search.

5.1.5 Uninstantiated actions and goals

By implementing better representations and corresponding data structures, the efficiency can be substantially improved. Also, by allowing uninstantiated variables in goals and actions, the planners can be greatly strengthened. Namely, the key to the improvement in efficiency is that uninstantiated moves and goals stand for sets of alternative moves and goals. Their instantiation is postponed until later, when it is known what their values should be. However, besides the advantage of having more efficient planning while allowing uninstantiated variables in goals and actions, this significantly complicates the planner.

5.1.6 GRAPHPLAN

The GRAPHPLAN planning method generates partially ordered plans represented as sequences of levels of actions, called 'levelled plans'. Each level of actions consists of actions that can be executed in any order, including parallel execution.

The GRAPHPLAN method uses the so-called planning graphs from which a levelled plans of actions can be drawn out at some point in time. A planning graph is a structure that represents in a relatively compact manner all possible sequences of states and actions starting from some start state. This means that the graph roughly represents what can potentially happen when sequences of actions are applied to the initial state of a planning problem. Nodes are organized into levels in a planning graph. Furthermore, a planning graph consists of interchanging levels of actions and state literals, meaning the first level nodes correspond to state literals, the second level nodes correspond to actions, the third level corresponds to state literals, the next level again to action, and so on. An action level may have actions as defined by the planning domain definition. Moreover, it can also contain the so-called persistence actions which represent virtual actions that preserve literals from the previous state level. This being said, if a literal is true in a state level, and no action in the next action level impacts this literal, then the literal will be true in the next state level. However, two literals can also be inconsistent if they cannot both be true at the same time, that is at the same level, indicating that actions have inconsistent effects. If two actions or two literals that belong to the same level cannot occur (or be true) at the same time, they are said to be mutually exclusive, or mutex for short. Namely, only one of them may actually happen, or none of them.

A literal that is present at a state level may be true at that level; however, this is not guaranteed. Correspondingly, an action that is present at an action level may potentially be part of a plan. However, if a literal does not appear at a state level then it definitely cannot be true at that level. This also holds for actions. Namely, if an action does not appear at an action level, then this action definitely cannot be part of a plan at that level. In conclusion, literals that appear at a state level are only potentially true, and actions that appear at an action level can only potentially appear in a plan at that level. There are constrains between actions and state literals that indicate the mutex relations, and possibly other relations between actions and literals.

In general, two actions are mutex if:

- Their preconditions are inconsistent, or
- Their effects are inconsistent, or
- An effect of one action is inconsistent with a precondition of the other action, or vice versa. This case of mutex actions is called interference.

A planning graph can be constructed relatively efficiently, in time polynomial in the length of the plan. Although a planning graph is not yet a solution, a plan can be extracted from it. One way of doing so is by satisfying the set of constraints between literals and actions. Next, despite the complexity of this task, efficient satisfaction algorithms that work efficiently on average are available. An alternative use of a planning graph is as a source of heuristic information for other planning methods. Planners that use planning graphs in one way or another are generally among the most competitive.

5.2 Colored ball example scenario

In this scenario the user can define the number of balls and their colors (red, green, blue). The balls can be positioned at four centers: one at which the balls are initially positioned, and three color centers where the balls are moved to afterwards. The idea is that the robot repositions each ball according to its color to the appropriate color center, following the path that was defined by the user. Through the application, the user can define the path to the centers as he/she wishes, that is the user can specify the points that the robotic arm should visit and the orientation of the end effector. This is useful because depending on the situation and on the obstacles the robot will have, the user can define different path, that is a different path for a different scenario.

5.2.1 Defining the path to a color center

The path to a specific center can be defined through the tab "Setup Ball Scenario" shown in Figure 5.1. Through this tab the user can define:

- Scenario name - different paths can be defined for different situations considering the environment of the robot, the obstacles it may encounter and other conditions. This means that the user can define multiple paths through the same center. Later, the user can pick the appropriate path through the name of the scenario.
- Phase - the path consists of multiple points. The phase number indicates the order by which the robot will go through the points. A bigger phase number means that the robot will go to that position later in the execution.
- Path to Center - there are four possible options (Red, Green, Blue, BackToCenter) and by selecting one of them, the user indicates for which center the path refers to.
- Solution Number - the user can specify by which solution the robotic arm should move to the specified point. There are 16 possible solutions. The user can also select "Automatic" which means that the program will go through all the solutions and select the first that achieves the goal. This option is selected by default.
- Orientation Vectors - the user can specify the orientation vectors and by that specify the orientation that the robotic arm should achieve when reaching the desired destination. By default, the arm is oriented upwards.

- Location - the user must specify the location that the robotic arm should achieve.

By clicking the ”SavePoint” button all the options that the user specified are saved in a SQLite database. I decided to use SQLite database and not a more traditional database, like Oracle, Microsoft Sql Server or MySql, because SQLite does not require a server, it does not require any installation on the user side and because it is very simple to use. The database consists of one table with 16 fields and it is shown in Figure 4.9.

5.2.2 Literals and actions

The following are the possible literals that can be encountered in the plan (in the explanations, an object can be a ball or a ball center):

- CLEAR(object) - indicates that the object does not have a ball on top of it.

- ON(ball, object) - the literal indicates which colored ball is on which object, where the object can be another colored ball or a ball center.

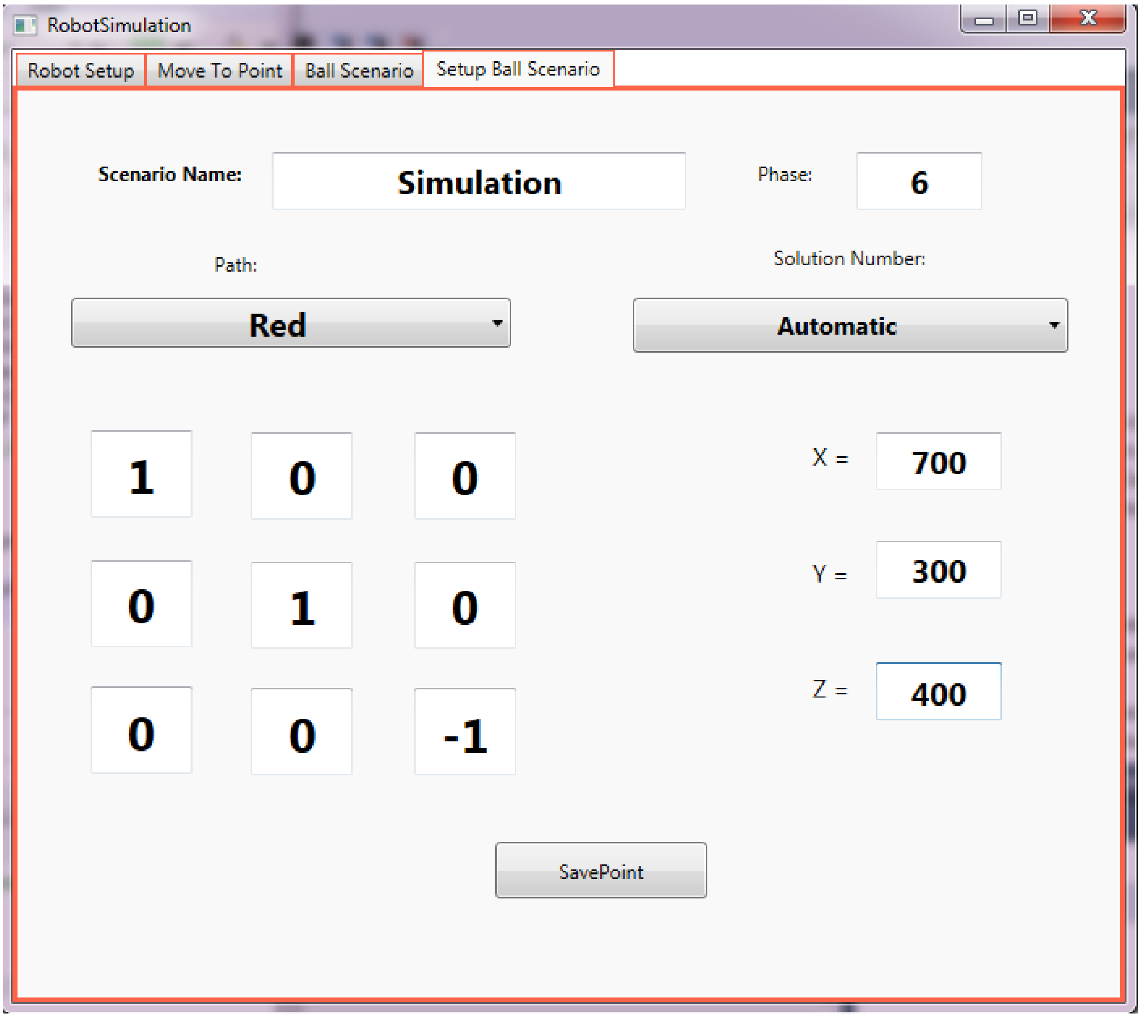
- GRIPPED(ball) - indicates which ball is currently taken by the gripper.

- NOT_GRIPPED(ball) - indicates that the ball is not taken by the gripper at the moment.

- HOVERS_AT(center) - this literal indicates at which center the robotic arm currently hovers at.

- GRIPPER_OFF - this literal indicates that the gripper of the robotic arm does not hold any ball at the moment.

- AT(ball, center) - indicates that the colored ball is positioned at the corresponding and correct color center.



Scenario Name: Simulation

Phase: 6

Path: Red

Solution Number: Automatic

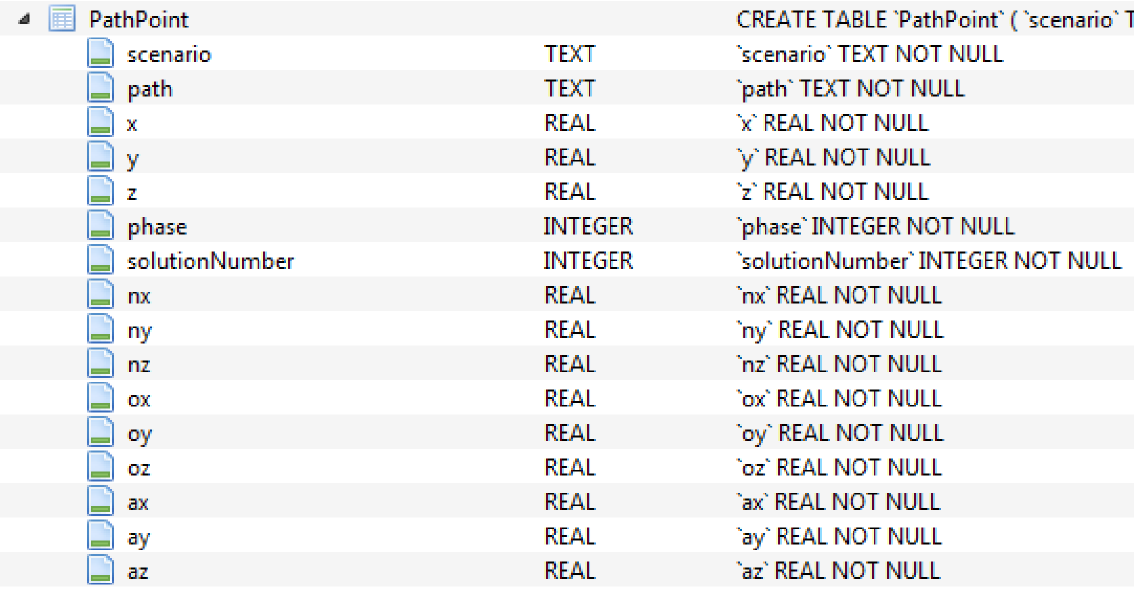
1	0	0
0	1	0
0	0	-1

X = 700

Y = 300

Z = 400

Figure 5.1: User interface of the "Setup Ball Scenario" tab.



PathPoint		CREATE TABLE `PathPoint` (`scenario` T
scenario	TEXT	`scenario` TEXT NOT NULL
path	TEXT	`path` TEXT NOT NULL
x	REAL	`x` REAL NOT NULL
y	REAL	`y` REAL NOT NULL
z	REAL	`z` REAL NOT NULL
phase	INTEGER	`phase` INTEGER NOT NULL
solutionNumber	INTEGER	`solutionNumber` INTEGER NOT NULL
nx	REAL	`nx` REAL NOT NULL
ny	REAL	`ny` REAL NOT NULL
nz	REAL	`nz` REAL NOT NULL
ox	REAL	`ox` REAL NOT NULL
oy	REAL	`oy` REAL NOT NULL
oz	REAL	`oz` REAL NOT NULL
ax	REAL	`ax` REAL NOT NULL
ay	REAL	`ay` REAL NOT NULL
az	REAL	`az` REAL NOT NULL

Figure 5.2: The table for keeping the points of a path to a center with its fields.

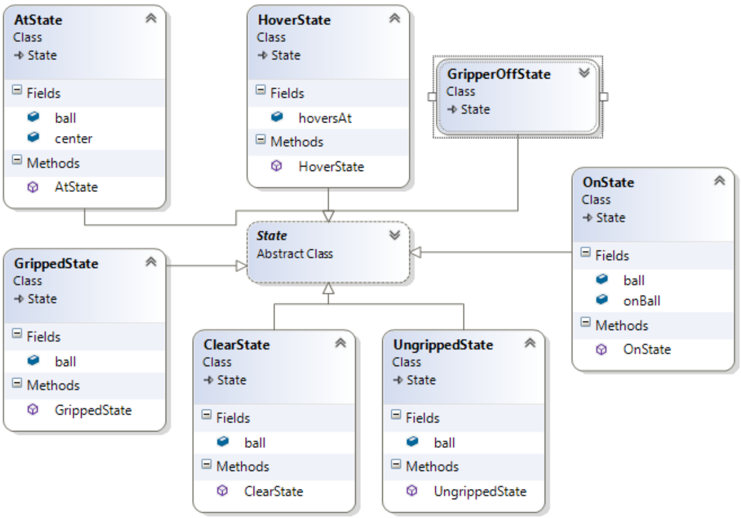


Figure 5.3: The class diagram for the state of the planner with the corresponding fields.

Each of the above mentioned literals is implemented as a class. All of them extend from the class State. The class hierarchy can be seen in Figure 5.3.

The possible actions that can be taken in order to resolve the scenario are the following:

- Grip(ball) - instructs the robotic arm to grip a certain ball.
  In order to execute this action, certain preconditions have to be fulfilled. First of all, the robotic arm has to hover at the initial center (HOVER(center) literal), then the corresponding ball must not have other ball on top of it (CLEAR(ball) literal) and the gripper must not hold other ball at the moment (GRIPPER_OFF literal).
  Executing this action adds two new literals: GRIPPED(ball) literal which indicates that the ball in question is now gripped, and CLEAR(object) which says that the object on which the gripped ball was on, is now clear.
  This action also removes two literals from the current plan: ON(ball, object) - this means that the gripped ball is not on the same object anymore, and GRIPPER_OFF which means that the gripper is not free anymore and that it holds a ball.

- LetGo(ball) - instructs the robotic arm to let go the currently gripped ball and by that to position the ball at the corresponding color center.
  In order to execute this action, two preconditions have to be fulfilled. First of all, the robotic arm has to hover at the corresponding color center (HOVER(center) literal) and the gripper must hold the ball at the moment (GRIPPED(ball) literal).
  Executing this action adds two new literals: AT(ball, center) literal which indicates that the ball is now positioned at a certain color center, and GRIPPER_OFF literal which means that the gripper does not hold a ball anymore.
  This action also removes one literal from the current plan: GRIPPED(ball) - this means that this ball is not held by the robotic arm.

- MoveToAllCenter(from) - instructs the robotic arm to move to the center where all the balls are initially positioned.
  In order to execute this action there is only one precondition, which is the robotic arm should hover at some color center (HOVER(from) literal).
  Executing this action also adds one literal, that is the robotic arm now hovers at the initial center (HOVER(to) literal).
  This action removes one literal as well: HOVERS(from) which means that the robotic arm no longer hovers at the previous color center.

- MoveToColorCenter(ball, to) - instructs the robotic arm to move the gripped color ball to the corresponding color center.
  In order to execute this action, the following preconditions have to be fulfilled: first, the robotic arm has to be positioned upon the initial ball center (HOVER(center) literal) and second, the ball in question has to be gripped by the robotic arm (GRIPPED(ball) literal).
  Executing this action also adds only one literal, which is that the robotic arm now hovers at the corresponding color center (HOVER(to) literal).
  This action removes one literal: HOVERS(center) which means that the robotic arm no longer hovers at the initial ball center.

Each of the actions is represented as a separate class which is a subclass of the class Action. Another important feature of the actions is that each action has a method called execute. In this method, the programming logic for actually performing the action is implemented and the actual explanation happens. The class hierarchy can be seen on Figure 5.4.

5.3 Block manipulation example scenario

In this scenario, the user can define the number of blocks and their relative position in regard to other blocks and centers. There are four centers. The

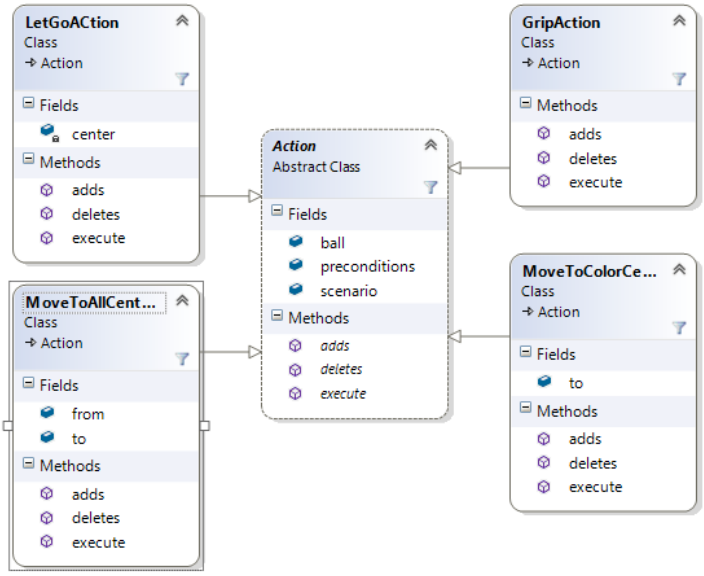


Figure 5.4: The class diagram for the actions of the planner with the corresponding fields.

user can define the initial state of the blocks and the goals that need to be accomplished after the manipulation of the blocks. The initial state is defined by specifying where each block is positioned. Each block can be positioned on top of a center or on top of another block which means that it is possible that the block can have one or more blocks on top of it. The planner should find the optimal (shortest) sequence of block manipulations in order to achieve the goals defined by the user.

5.3.1 Literals and actions

The following are the possible literals that can be encountered in the plan (in the explanations, an object can be a block or a center):

- CLEAR(object) - indicates that the object does not have a block on top of it.
- ON(block, object) - the literal indicates which block is on which object,

where the object can be another block or a block center.

Each of the above mentioned literals is implemented as a class. All of them extend from the class State.

In this scenario only one type of action can be taken, that is the action MOVE(block, object1, object2) which instructs the robotic arm to move the block from "object1" to "object2" where "object1" and "object2" can be a center or block.
In order to execute this action, certain preconditions have to be fulfilled. First of all, the block must not have another block on top of it (CLEAR(block) literal), also the block that is being moved has to be on top of "object1" (ON(block,object1)). Moreover, the object where the block is being moved to must not have a block on top of it(CLEAR(object2)).
Executing this action adds two new literals: CLEAR(object1) literal which indicates that the object on which the block was positioned does not have a block on itself anymore. The other literal that is added is (ON(block,object2)) which means that now the block is positioned on top of object2.
This action also removes two literals from the current plan: ON(block, object) - this means that the block in question is not on object1 anymore, and CLEAR(ojbect2) which means that object2 now has a block on top of it.

5.4 Goal regression planner

When evaluating what kind of planner to use for creating plans for the above mentioned scenarios, I decided to use the goal regression planner because it produces optimal plans.

For implementing the goal regression planner, I used the algorithm described in the book Prolog Programming in Artificial Intelligence (4th edition) by Ivan Bratko.

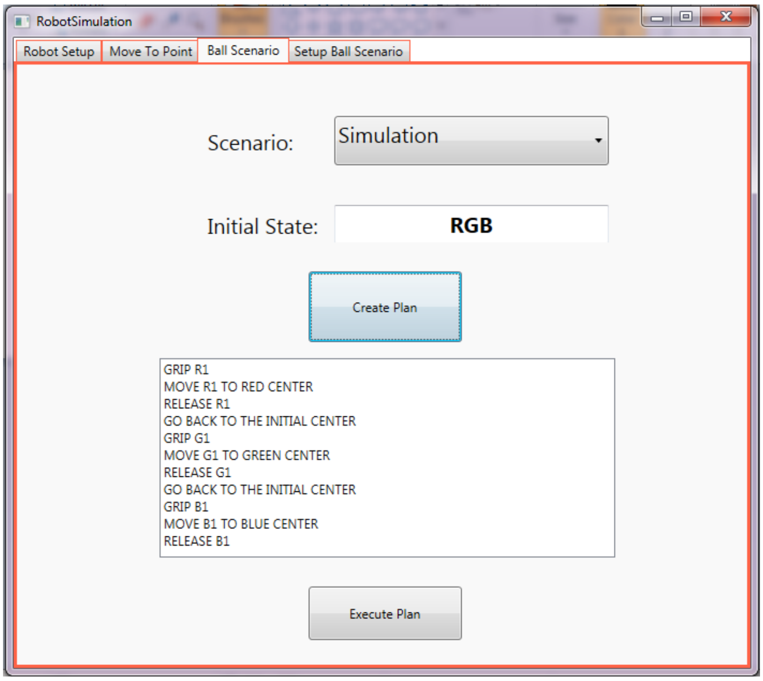
- I implemented the algorithm with iterative deepening, with starting depth of 1 and increasing the depth until a solution is found.

- The algorithm first checks if all the main goals are satisfied, if this is the case, then an empty plan is returned.
- If all the goals are not satisfied then the depth is checked, if the depth is zero that means that no solution was found.
- If the depth is not zero the algorithm selects a goal, and then selects an action that achieves the current goal and preservers the other goals. After that, the regressed goals are computed, which now become the new goals.
- The algorithm is run recursively on the new goals with the depth decreased by one. If a solution is found for the new goals the current action is added to the solution for the new goals. If there is no solution for the new goals a new action is tried.

The algorithm was optimized to check for impossible goals. Also memoization was added so that if a solution for specific goals and a certain depth was already found, not to recompute it again.

5.4.1 Using the goal regression planner in the colored ball scenario

The setting up of the initial state of the plan can be done through the "Ball Scenario" tab of the application shown in Figure 5.5. First, in the combo box the user can choose a previously defined path of movement for the robotic arm. After that, in the text box the user can set how many balls there will be and their colors. The order of the balls depends on how they are written in the text box. 'R' is for red ball, 'G' is for green ball and 'B' is for blue ball. After the balls are set, the user can click on the "Create Plan" button and the plan will be created and the order of execution will be shown in the list box. The user can actually start the execution by clicking the "Execute Plan" button. One thing to be noticed is that here the goals are automatically created by the program and the user does not need to define them himself.



Scenario: Simulation

Initial State: RGB

```
GRIP R1
MOVE R1 TO RED CENTER
RELEASE R1
GO BACK TO THE INITIAL CENTER
GRIP G1
MOVE G1 TO GREEN CENTER
RELEASE G1
GO BACK TO THE INITIAL CENTER
GRIP B1
MOVE B1 TO BLUE CENTER
RELEASE B1
```

Figure 5.5: The user interface of the ”Ball Scenario” tab used for creating and executing a plan.

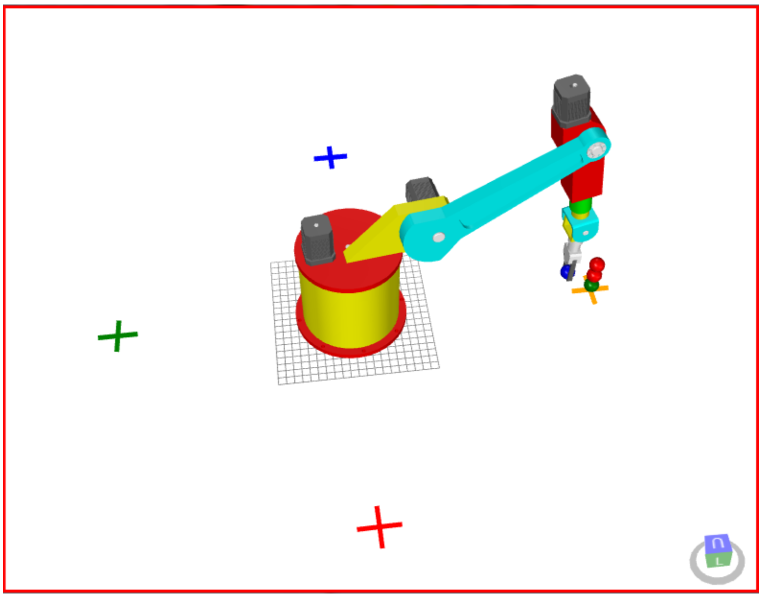


Figure 5.6: The position of the balls and the robotic arm before executing the plan.

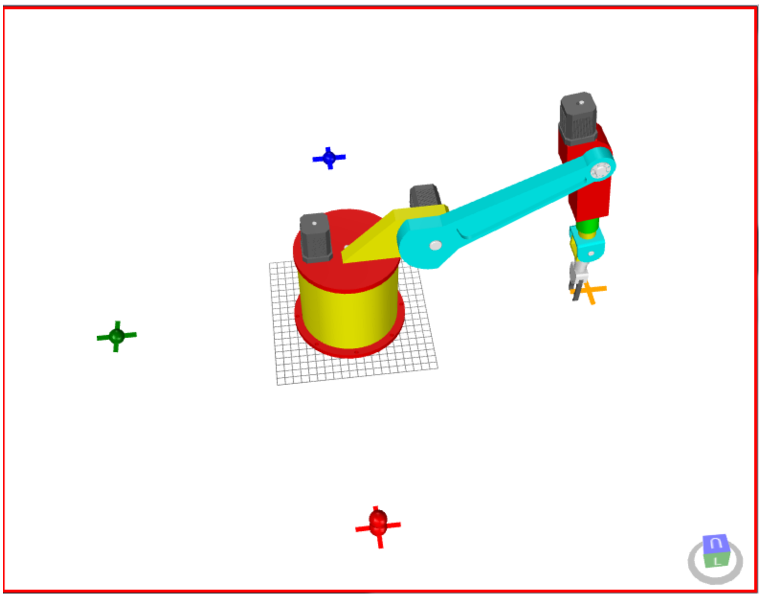


Figure 5.7: The position of the balls and the robotic arm after executing the plan.

Figure 5.6 shows the state of the balls before the execution of the already created plan and in Figure 5.7 we can see the balls and their positions after the plan has been executed. There are two red, one green and one blue ball.

5.4.2 Using the goal regression planner in the block scenario

The setting up of the initial state of the plan for the block scenario can be done through the "block Scenario" tab of the application shown in Figure 5.8. The user can define the number of blocks using the text box on top of the screen. After setting the number of blocks, the user can define literals for the initial state and goal literals.

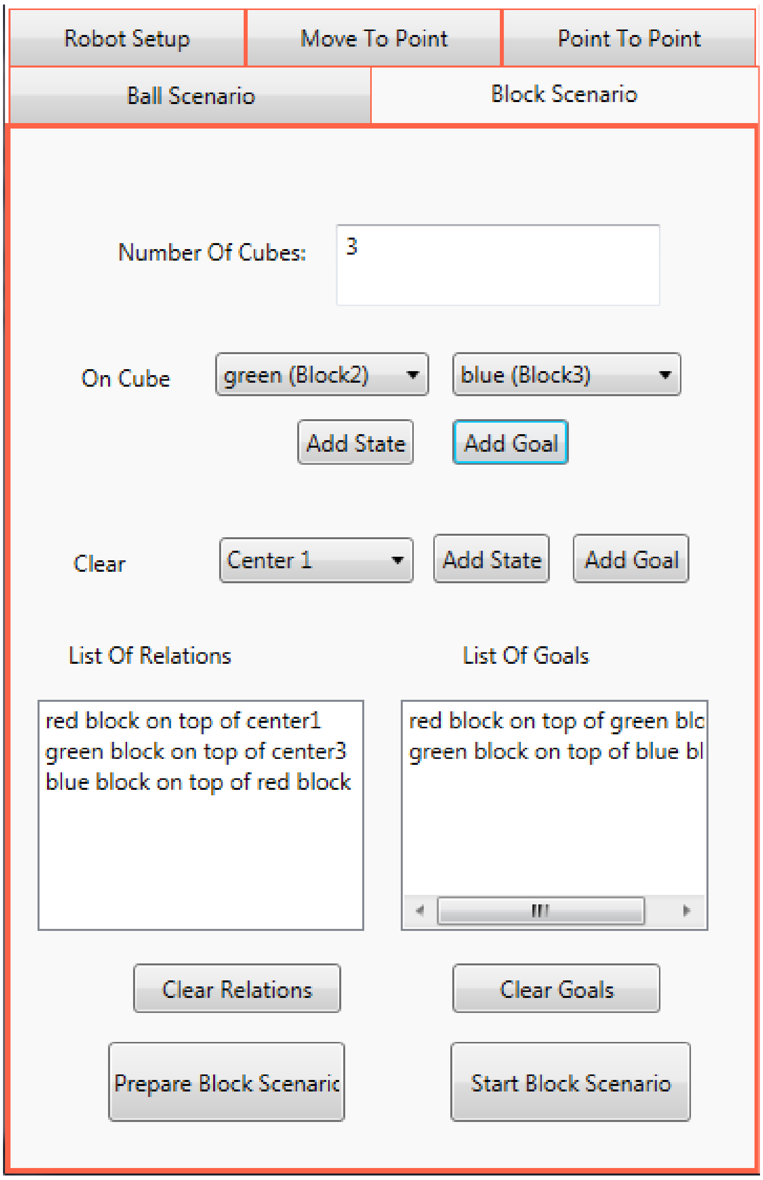
Setting up a On literal can be done through the two combo boxes where the user can pick a block and an object on which the block is. By clicking the "Add state" button the user adds this literal to the initial state, if the user clicks the "Add final" button the literal will be added to the list of goals. The planner will try to create a plan where the goals are achieved.

Setting up a Clear literal can be done through a combo box where the user can pick an object that does not have a block on top of it. By clicking the "Add state" button the user adds this literal to the initial state and if the user clicks the "Add final" button the literal will be added to the list of goals. Clear literals are also automatically added to the initial state if not defined otherwise.

The initial state is shown in the list "list of relationships" and the list of goals are shown in the list "list of goals". The user can reset the lists by clicking on the corresponding reset button.

By clicking the "Prepare Block Scenario" the initial state is shown in the simulation and by clicking "Start Block Scenario" the plan is created and executed.

For example, if the initial state is defined as the following: (on(red block, center1), on(green block, center3), on(blue block, red block), clear(blue block), clear(green block), clear(center2), clear(center4)) where the clear



Number Of Cubes: 3

On Cube: green (Block2), blue (Block3)

Clear: Center 1

List Of Relations

red block on top of center1
green block on top of center3
blue block on top of red block

List Of Goals

red block on top of green blc
green block on top of blue bl

Figure 5.8: The block scenario tab used for defining the initial state of the block scenario and the goals that need to be achieved.

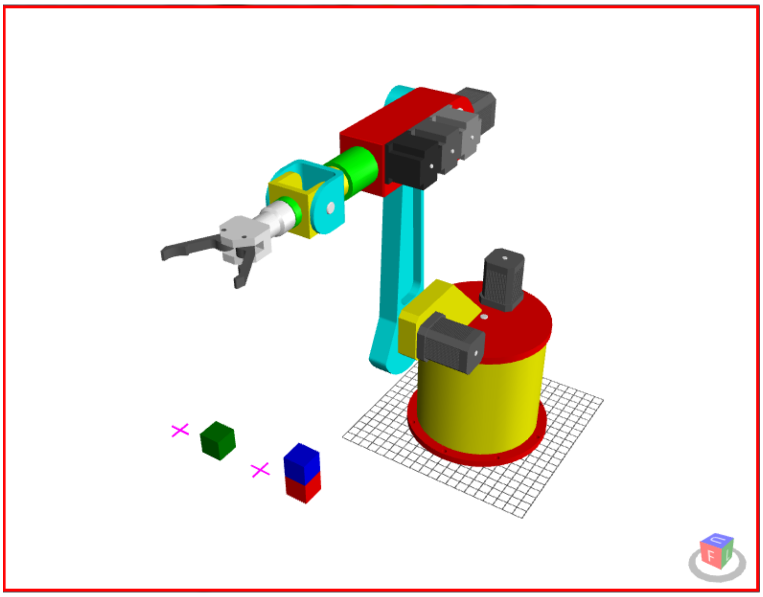


Figure 5.9: The state of the block scenario before executing the plan.

states the user does not have to define and the list of goals as: (on(red block, green block), on(green block, blue block)) the scene will look like in Figure 5.9.

After clicking the start block scenario the plan is created and executed. The final state of the scenario is shown on Figure 5.10.

The list of actions that the plan created are the following:(move(blue block, red block, center2), move(green block, center3, blue block), move(red block, center1, green block). The final state is: (on(blue block, center1), on(green block, blue block), on(red block, green block), clear(red block), clear(center1), clear(center3), clear(center4)). From this we can conclude that the planner successfully created and executed a plan that achieves the goals.

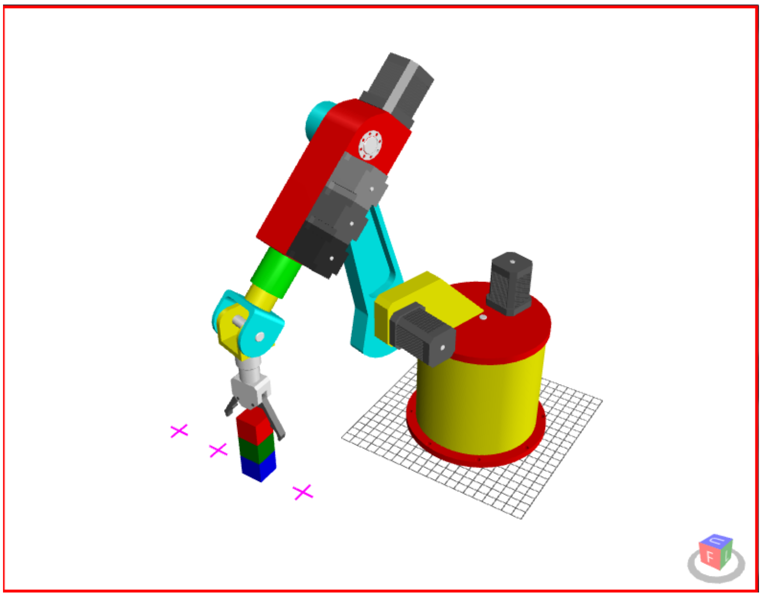


Figure 5.10: The state of the block scenario after executing the plan.

5.5 Explaining the actions that the robot takes

The motivation for explaining the actions is that we often see robots taking particular actions that we do not truly understand. This also makes it easier for engineers, users and for the audience to understand what the robot is actually trying to achieve.

I resolve the explanation by basic description of each action and also by describing each action in regard to the immediate regressed goal that it is trying to achieve and also in regard to the main, user defined goal it tries to achieve.

The basic explanation of each action is defined when searching for the optimal plan by using the direct regressed goal it accomplishes. The action can actually fulfill more than one regressed goal but only the regressed goal for which this action was specifically picked is used for the basic explanation. After the plan is created, if the regressed goal is not a goal that the user defined, the explanation of the action will be further enriched. The enrichment of the explanation is done using an algorithm that works as follows:

The algorithm first checks if the regressed goal is also a user defined (main) goal, if this is the case no enrichment of the explanation is done. If this is not the case, then first the algorithm checks if some of the literals that this action adds is a main (user defined) goal, then the following explanation is added "This also achieves the main goal" plus the explanation of the main goal. If the action does not directly affect a main goal then the algorithm searches for the main goals that it indirectly affects. If only one or two main goal are indirectly affected then the explanations for these main goals are added to the explanation of the action. If more than two main goals are affected by this action then only the numbers of the main goals are added to the explanation in order not to over complicate the explanations. Also if the action affects all the main goals then the following explanation is added "This action affects all the main goals".

The main goals that are indirectly affected by the action are found in the following way:

- First all of the actions that happen after the current action are checked to see if some of the preconditions for these actions are added by the current action.
- For each action that is found the algorithm checks if the literals added by these actions contain a main goal. If this is the case than the main goals added by these actions are returned.
- If none of the literals that these actions add are main goals then for each action the algorithm is called recursively to find the affected goals by these actions.

5.5.1 Explaining the actions of the planner ball scenario

If the initial state is defined as follows: (clear(redball1), on(redball1, greenball1), on(greenball1, AllBallCenter), hover(AllBallCenter), gripperOff) and the list of goals as: (atState(redball1, RedBallCenter), atState(greenball1, GreenBallCenter)) the generated explanations for the actions are the following:

- Action grip(redball1). The basic explanation for this action is the following:"Gripping ball redball1 so that redball1 is gripped.". This explanation is further enriched by the sentence "This brings the robot closer to achieving the main goal ball redball1 at center RedBallCenter". The enrichment is done because the action indirectly affects the main goal and directly affects the regressed goal gripped(redball1).
- Action MoveToColorCenter(redball1, RedBallCenter). The basic explanation for this action is the following:" Moving ball redball1 to RedBallCenter so that the robot hovers at RedBallCenter". This explanation is further enriched by the sentence "This brings the robot closer to achieving the main goal ball redball1 at center RedBallCenter". The enrichment is done because the action indirectly affects the main goal and directly affects the regressed goal hover(RedBallCenter).
- Action LetGo(redball1, RedBallCenter). The basic explanation for this action is the following:"Letting go ball redball1 so that gripper is off". This explanation is further enriched by the sentence "This also achives the main goal ball redball1 at center RedBallCenter".
- Action MoveToAllCenter(RedBallCenter). The basic explanation for this action is the following:"Moving to the initial center from RedBallCenter so that the robot hovers at AllBallCenter". This explanation is further enriched by the sentence "This brings the robot closer to achieving the main goal ball greenball1 at center GreenBallCenter". The enrichment is done because the action indirectly affects the main goal and directly affects the regressed goal hover(allBallsCenter).
- Action grip(greenball1). The basic explanation for this action is the following:"Gripping ball greenball1 so that greenball1 is gripped". This

explanation is further enriched by the sentence "This brings the robot closer to achieving the main goal ball greenball1 at center GreenBallCenter". The enrichment is done because the action indirectly affects the main goal and directly affects the regressed goal gripped(greenball1).

- Action MoveToColorCenter(greenball1, GreenBallCenter). The basic explanation for this action is the following:"Moving ball greenball1 to GreenBallCenter so that the robot hovers at GreenBallCenter". This explanation is further enriched by the sentence "This brings the robot closer to achieving the main goal ball greenball1 at center GreenBallCenter". The enrichment is done because the action indirectly affects the main goal and directly affects the regressed goal hover(GreenBallCenter).
- Action LetGo(greenball1, GreenBallCenter). The basic explanation for this action is the following:" Letting go ball greenball1 so that greenball1 at center GreenBallCenter.".

5.5.2 Explaining the actions of the planner in the blocks scenario

Example with 3 blocks and 3 actions

If the initial state is defined as follows: (on(red block, center1), on(green block, center3), on(blue block, red block), clear(blue block), clear(green block), clear(center2), clear(center4)) shown in Figure 5.9 and the list of goals as: (on(red block, green block), on(green block, blue block), on(blue block, center2) shown in Figure 5.10, the generated explanations for the actions are the following:

- Action move(blue block, red block, center2). The basic explanation for this action is the following:"Moving blue block from red block to center 2 so that red block is clear." This explanation is further enriched by the sentence "This also achieves the main goal blue block on top center

2”. The enrichment is done because the action also indirectly achieves one of the main goals.

- Action move(green block, center3, blue block). The basic explanation for this action is the following:”Moving green block from center 3 to blue block so that green block on top of blue block.” No further enrichment is needed because this action directly achieves the main goal on(green block, blue block).
- Action move(red block, center1, green block). The basic explanation for this action is the following:”Moving red block from center 1 to green block so that red block on top of green block.” No further enrichment is needed because this action directly achieves the main goal on(red block, green block).

Example with 5 blocks and 8 actions

If the initial state is defined as the following: (on(red block, center1), on(green block, center2), on(blue block, green block), on(violet block, center4), on(orange block, violet block), clear(blue block), clear(orange block), clear(red block), clear(center3)) shown in Figure 5.11 and the list of goals as: (on(violet block, center1), on(orange block, violet block), on(red block, center1), on(green block, red block), on(blue block, green block)) shown in Figure 5.12, the generated explanations for the actions are the following:

- Action move(blue block, green block, orange block). The basic explanation for this action is the following:”Moving blue block from green block to orange block so that green block is clear.”. This explanation is further enriched by the sentence ”This brings the robot closer to completing the main goals: green block on top of red block and blue block on top of green block.”. The enrichment is done because the action also indirectly contributes to achieving two of the main goals.
- Action move(green block, center2, blue block). The basic explanation for this action is the following:”Moving green block from center 2 to blue

block so that center 2 is clear.". The explanation is further enriched by the sentence "This brings the robot closer to completing the main goals: red block on top of center 2 and green block on top of red block.". The enrichment is done because the action also indirectly contributes to achieving two of the main goals.

- Action move(red block, center1, center2). The basic explanation for this action is the following "Moving red block from center 1 to center 2 so that center 1 is clear.". The explanation is further enriched by the sentence "This also achieves the main goal red block on top of center 2.". This is done because the action as a side effect also achieves one of the main goals.
- Action move(green block, blue block, red block). The basic explanation for this action is the following:"Moving green block from blue block to red block so that blue block is clear.". This explanation is further enriched by the sentence "This also achieves the main goal green block on top of red block.". The enrichment is done because the action also achieves one of the main goals.
- Action move(blue block, orange block, green block). The basic explanation for this action is the following:"Moving blue block from orange block to green block so that orange block is clear.". This explanation is further enriched by the sentence "This also achieves the main goal blue block on top of green block.". The enrichment is done because the action also achieves one of the main goals.
- Action move(orange block, violet block, blue block). The basic explanation for this action is the following:"Moving orange block from violet block to blue block so that violet block is clear.". This explanation is further enriched by the sentence "This brings the robot closer to completing the main goals: violet block on top of center 1 and orange block on top of violet block.". The enrichment is done because the action contributes to achieving two main goals.

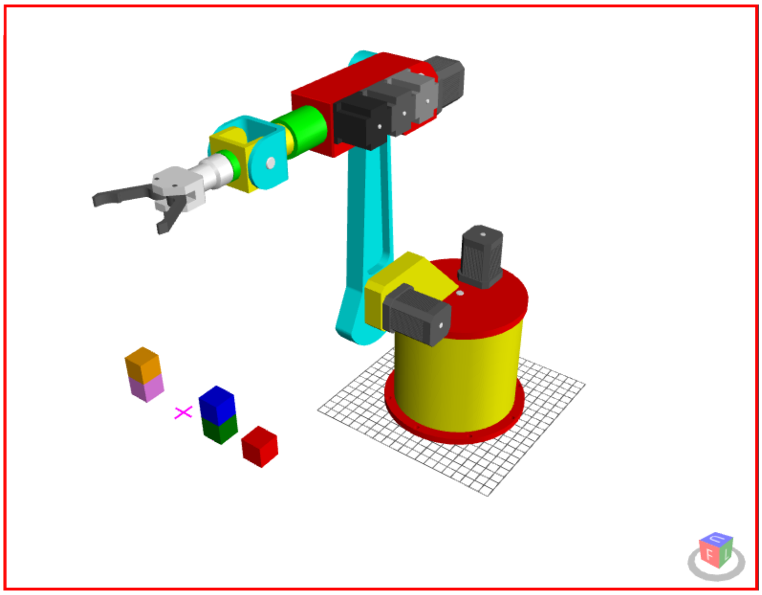


Figure 5.11: The state of the scenario with 5 blocks before executing the plan.

- Action move(violet block, center4, center1). The basic explanation for this action is the following:”Moving violet block from center 4 to center 1 so that violet block on top of center 1.”. No further enrichment is needed because this action directly achieves a main goal.
- Action move(orange block, blue block, violet block). The basic explanation for this action is the following:”Moving orange block from blue block to violet block so that orange block on top of violet block.”. No further enrichment is needed because this action directly achieves a main goal.

5.5.3 Assessment of the algorithm for generating explanations

In the examples given in the previous subsections the algorithm works as expected and generates appropriate and human understandable explanations.

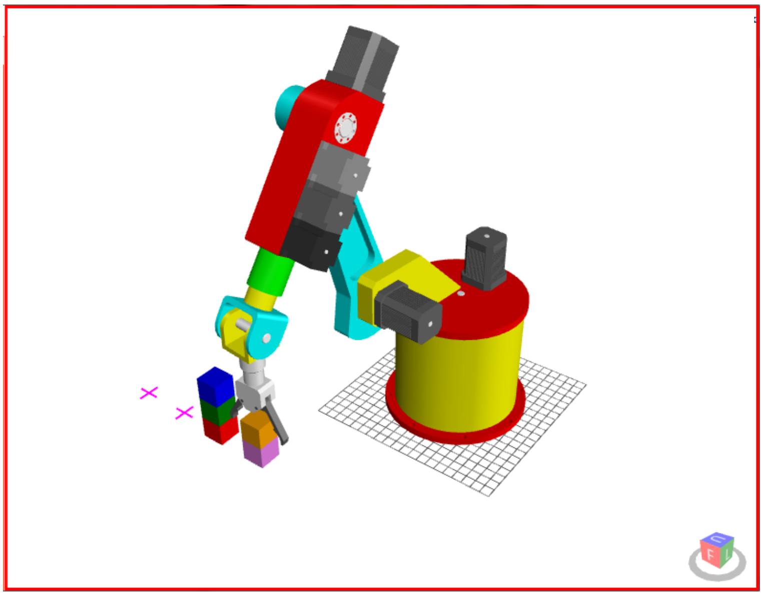


Figure 5.12: The state of the scenario with 5 blocks after executing the plan.

This algorithm could run into a problem if the number of goals is very large, in which case it would be really hard to describe all the goals that are affected by an action because the explanations can become too long. I address this problem by not adding the whole description of the goals, but only their order number. Another solution could be to add conceptual explanations that can be used when the number of goals is too big. The conceptual explanations could be in the following example format: the first 5 actions build a stack of blocks on center1, the last 3 actions build another stack of blocks on center2. This could be done in some future work.

Chapter 6

Conclusion

In this master thesis I developed a program for manipulating a robotic arm with six degrees of freedom. The program also contains a simulation of the robotic arm which mimics its actual movement. Further on, the program can be used for creating motion plans using which the robot can resolve a certain scenario and which enables the robot to solve certain classes of tasks and to explain to the user why it took each particular action.

The explaining of the actions is important because now the engineers as well as the broader audience can reason and better understand how and why the robot is acting. Also, the visual simulation that I developed can be used for educational purposes where the robotic arm does not need to be physically present.

The developed program can be used for manipulating any robotic arm with six degrees of freedom by only changing the inverse kinematic equations and the dimensions of the robotic arm.

Taking in regard the related work that has already been done in the field of explaining actions for automated systems, I could state that by using planning as a basis for the explanations, I managed to develop an alternative and novel approach for explaining the actions that the robot makes. Moreover, this kind of action explanation makes the robot's behavior more understandable for viewers.

As future work, I would develop more scenarios for which the robotic arm could come up with plans and explanations. Also, further research can be done in shortening the explanations for the actions and making them more understandable for the user. Also, the goal regression algorithm can be further optimized to build more complex plans.

Bibliography

[1] J. D. Lee, K. A. See, Trust in automation: Designing for appropriate reliance, Human Factors: The Journal of the Human Factors and Ergonomics Society 46 (1) (2004) 50–80.

[2] P. P. Duez, M. J. Zuliani, G. A. Jamieson, Trust by design: information requirements for appropriate trust in automation, in: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, IBM Corp., 2006, p. 9.

[3] D. R. Billings, K. E. Schaefer, J. Y. Chen, P. A. Hancock, Human-robot interaction: developing trust in robots, in: Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction, ACM, 2012, pp. 109–110.

[4] M. Lomas, R. Chevalier, E. V. Cross II, R. C. Garrett, J. Hoare, M. Kopack, Explaining robot actions, in: Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction, ACM, 2012, pp. 187–188.

[5] L. Takayama, D. Dooley, W. Ju, Expressing thought: improving robot readability with animation principles, in: Proceedings of the 6th international conference on Human-robot interaction, ACM, 2011, pp. 69–76.

[6] D. J. Brooks, A. Shultz, M. Desai, P. Kovac, H. A. Yanco, Towards state summarization for autonomous robots., in: AAAI Fall Symposium: Dialog with Robots, 2010.

[7] O. Lemon, A. Bracy, A. Gruenstein, S. Peters, Information states in a multimodal dialogue system for human-robot conversation, in: 5th Workshop on Formal Semantics and Pragmatics of Dialogue (Bi-Dialog 2001), 2001, pp. 57–67.

[8] M. Cvetkov, Industriski robot so 6 stepeni na sloboda [industrial robot with 6 degrees of freedom], 2014, unpublished thesis.

[9] S. Nakov, D. Dimitrov, H. Germanov, M. Stoynov, K. Valkov, M. Bivas, Y. Yosifov, in: Fundamentals of computer programming with C#, 2013.

[10] Helix toolkit documentation, `https://media.readthedocs.org/pdf/helix-toolkit/latest/helix-toolkit.pdf`, accessed: 2016-12-20.

[11] Blender, `https://www.blender.org`, accessed: 2016-12-20.

[12] Sqlite, `https://sqlite.org`, accessed: 2016-12-20.

[13] I. Bratko, Planning, in: Prolog programming for artificial intelligence (4th ed.), 2012.