BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

SKETCHUP VISUALIZATION OF STATIONARY ROBOTS

VIZUALIZACE STACIONÁRNÍHO ROBOTU VE SKETCHUP

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR — **Egor Iutkin**

AUTOR PRÁCE

SUPERVISOR — **Ing. František Burian, Ph.D.**

VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Bachelor's Thesis

Bachelor's study program **Automation and Measurement**

Department of Control and Instrumentation

Student: Egor Iutkin ***ID:*** 203568

Year of study: 3 ***Academic year:*** 2019/20

TITLE OF THESIS:

Sketchup visualization of statrionery robots

INSTRUCTION:

The aim of this thesis is to build simple visualisation system for robot manipulator in software Sketchup.

1. Learn possibilities of software Sketchup Make 2017 and Ruby scripting inside Sketchup.
2. Create plugin in Ruby, that enables visualisation of selected stationery robot (Kuka KRC6 sixx).
3. Create software in any language, that will parametrize the Sketchup model through TCP connection.
4. Create software in any language that bridges Sketchup (or software from point 3) and robot. This software will transfer positions of robot joints into Sketchup model.

RECOMMENDED LITERATURE:

SPONG, Mark W., Seth HUTCHINSON a M. VIDYASAGAR. Robot modeling and control. Hoboken, NJ: John Wiley, c2006. ISBN 978-0471649908.

Date of project specification: 3.2.2020 ***Deadline for submission:*** 8.6.2020

Supervisor: Ing. František Burian, Ph.D.

doc. Ing. Václav Jirsík, CSc.
Chair of study program board

Abstract

This thesis focuses on building a simple visualization system for a robot manipulator in the software SketchUp. The three-dimensional interface of the software is employed to visualize the movement of a virtual robot under the control of an external application (initially under the control of a real robot), which uses a suite of TCP/IP protocols to track the robot's position.

Communication with the project is done via Ruby code using the SketchUp API, and the values of the angles of the individual robot's joints are interpreted in form of a parametric movement of the virtual robot. The client-server-client application is created in C++, C# and Ruby, and the network sockets are used to establish a TCP connection in the local network.

In the application, the Ruby plugin, which provides the communication interface, acts as the first client. The C# custom application acts as the second client, called the "control client", which provides a user interface to configure the server connection and set individual joint angles of the virtual robot. Communication between the two clients is facilitated by the C++ TCP server (console application), which retrieves data from the control client and forwards it to the Ruby side.

Keywords

SketchUp, Ruby, API, robot, KUKA, visualization, parametric motion, TCP, IP, socket, network interface, server, client.

Rozšířený abstrakt

Pokud jde o parametrickou vizualizaci pohybů robota, první, co přichází na mysl, je demonstrace schopností robota, jeho pracovního rozsahu, přesnosti, maximálního dosahu a rychlostí; za druhé, uživatel může zvážit vizualizaci technologického procesu nebo testování zdrojových kódu.

V každém případě je rozsah aplikací pro tuto technologii široký, ale programy, které umožňují správu virtuálních robotů, obvykle přicházejí za vysokou cenu a jsou obvykle dostupné pouze velkým společnostem. V praxi vizualizace parametrických robotických pohybů může být nezbytná pro menší společnosti nebo organizace, jako jsou vysoké školy.

Prvním cílem této práce je prostudovat mechanismus interakce mezi uživatelem a virtuálním robotem v rámci bezplatné verze aplikace SketchUp Make 2017, která má velké množství nástrojů pro vytváření 3D objektů a především má své vlastní API, které umožňuje rozšířit funkčnost programu prostřednictvím kódu Ruby. Dalším rokem je otestovat možnost vytvoření pluginu, který může komunikovat s objekty ve scéně SketchUp v reálném čase a vyměňovat data s externími aplikacemi prostřednictvím standardního komunikačního modelu (v našem případě prostřednictvím protokolů TCP / IP). Realizace tohoto úkolu může vést k podrobnějšímu zkoumání možnosti využití rozhraní SketchUp k vizualizaci průmyslových procesů.

Návrh řešení je rozdělen na 3 části: implementaci parametrického pohybu robota ve scéně SketchUp pomocí Ruby pluginu; implementace serveru, ze kterého bude Ruby plugin přijímat data s novými úhly pro změnu polohy robota; implementaci testovacího klient. Výsledkem této práce bude program klient-server-klient který dohromady tvoří simulační nástroj pro vizualizaci stacionárního robotu.

3D model robota ve SketchUp je řízen pluginem, který zaprvé vykonává funkci skriptu, který převádí přijata hodnoty uhlů jednotlivých kloubů na parametrický pohyb robota. Za druhé, obsahuje rozhraní soketu a vykonává funkci TCP klienta (dále Ruby klient). Klient neposílá data o své poloze na server, ale ze serveru pouze přijímá data s novými úhly. Pouze během prvního připojení odesílá zprávu na server, aby jej server mohl mezi ostatními identifikovat. Klient Ruby má také uživatelské rozhraní pro připojení k serveru a odpojení od něj a změnu parametrů připojení.

Testovací TCP klient – je C# klient. Aplikace má rozhraní pro připojení k serveru, okno pro zobrazení informací, pole pro zadaní uživatelských hodnot a výstupní pole s aktuální pozicí robota. Klientská aplikace obdrží ze serveru aktuální polohu, pokud byla poloha robota změněna jiným klientem. Uživatel může nastavit úhly pro každý spoj zvlášť. Informace z polí jsou zapsány v jednom bajtkódu a odeslány na server stisknutím tlačítka „Set“. „Continuous Mode“ – úhly jsou rozděleny do menších kroků pro každý kloub a posílány postupně na server. Tento

režim interakce mezi klientem a serverem vytváří situaci, ve které jsou data posílána na server jako datový tok v reálném čase, čímž vydává spojení se skutečným robotem.

TCP server – funguje jako přesměrovače a nezpracovává data přijatá od klientů. K serveru lze připojit několik klientů, každý klient je připojen k samostatnému portu a je zpracován v samostatném vlákně. Připojený klienti jsou zahrnuti do seznamu navázaných spojení – do pole soketů. Když server přijme data od klienta C#, je tato zpráva přeposlána všem ostatním klientům, z nichž jedním je klient Ruby. V tomto případě obdrží další klienti ze serveru informace o změnách polohy robota jedním z ostatních aktivních klientů, současně Ruby klient interpretuje přijatá data do úhlů a mění polohu robota. Server je konzolová aplikace, která uživatele informuje o klientech, kteří jsou k němu aktuálně připojeni, a o přenosu dat. První zprávou serverové aplikace je vždy IP adresa a port, na kterém pracuje. Pokud se klient úspěšně připojí, server zapíše parametry připojeného zařízení a přiřadí mu ID. Když server obdrží zprávu od klienta, vypíše do konzole jeho ID a přijatý úhly pro robota v pořadí kloub 1 - kloub 6. Pokud je klient odpojen, konzole zobrazí příslušnou zprávu.

Implementovaný simulační nástroj má řadu výhod a nevýhod, které jsou uvedeny níže.

První část práce je zaměřena na implementaci parametrického pohybu robota v prostředí SketchUp. Toho bylo dosaženo pomocí hierarchického uspořádaní prvků robotu, kde spodní kloub v hierarchii je spojen s horním a vázán k jeho otočnému bodu. Tato realizace je optimálním a pravděpodobně jediným řešením úlohy v prostředí Free SketchUp.

Plugin Ruby nepoužívá podprocesové funkce, které by mohly být použity k implementaci zpracování toku dat ze serveru v samostatném vláknu. Bohužel, podprocesové funkce uvnitř SketchUp nefungovaly i když samotný Ruby má jejích podporu. V důsledku toho bylo přijímání dat ze serveru slabě implementováno. Funkce přijímání dat byla implementována pomocí časovače, což způsobuje zpoždění ve zpracování dat a komplikuje program. Toto řešení není optimální.

Implementace serveru je nejúspěšnější. Použitím podprocesových funkcí a pole soketů bylo dosaženo dobré podpory několika klientů najednou. Nevýhody na straně serveru: program nemůže určit IP adresu počítače v síti a nemůže si vybrat volný port. Ruční zadání těchto parametrů snižuje rychlost a jednoduchost interakce s aplikací.

Výhodou aplikace C# klientu je jednoduché rozhraní pro připojení k řízení serveru a robota; implementace informačního okna; příjem a zpracování dat ze serveru ve vláknové funkci.

Nevýhodou je slabá implementace „Continuous mode“; použití neplatných hodnot v polích může způsobit výjimku (dvojité tečky nebo dvojité čárky). Což není velký problém, ale stále není opraveno.

Obecnou nevýhodou klientů a serveru je to, že neexistuje žádné dynamické přidělení paměti pro přijatí rámců, když server přijímá zprávy od klientů nebo naopak. Jediným řešením bylo poslat 2 rámců, jeden s velikostí zprávy, druhý se samotnou zprávou, což snižuje přenosovou rychlost, a proto nebylo implementováno.

Stručně řečeno, nastroj implementuje počáteční úlohu – vytvoření jednoduchého simulačního nástroje pro vizualizaci stacionárního robota v aplikaci SketchUp. Níže jsou uvedeny možné nápady, jak tento nástroj vylepšit.

Vytvořit univerzální řešení pro import různých robotů; upravit plugin Ruby tak, aby ovládal více robotů ve scéně současně; vytvořit klient, který rozumí programovacím jazykům (jako například KRL) a interpretuje kód pro plugin Ruby; přidat rozhraní pro připojení k programovatelnému logickému automatu. Nakonec tato technologie může vést k vývoji vlastní platformy pro vytváření digitálních továren založených na Free SketchUp.

Klíčová slova

SketchUp, Ruby, API, robot, KUKA, vizualizace, parametrický pohyb, TCP, IP, socket, síťové rozhraní, server, klient.

Bibliographic citation:

IUTKIN, Egor. *Sketchup visualization of stationary robots*. Brno, 2020. Also available from: https://www.vutbr.cz/studenti/zav-prace/detail/127093. Bachelor's thesis. Brno University of Technology, The Faculty of Electrical Engineering and Communication, The Department of Control and Instrumentation. Supervisor: František Burian.

Bibliografická citace:

IUTKIN, Egor. *Vizualizace stacionárního robotu ve Sketchup*. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/127093. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce František Burian.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma *Sketchup visualization of stationary robots* jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 7. června 2020 ……

podpis autora

ACKNOWLEDGEMENT

I would like to thank my supervisor of the bachelor thesis Ing. František Burian, Ph. D., for his help, his professional view, quick answers to my questions, good recommendations and good theoretical support of my ideas and suggestions.

In Brno: 7. June 2020 .....

author's signature

Content

Abbreviations

Abbreviation:

API	…	Application programming interface
ARP	…	Address Resolution Protocol
BUT	…	Brno University of Technology
DCHP	…	Dynamic Host Configuration Protocol
DNS	…	Domain Name System
FEEC	…	The Faculty of Electrical Engineering and Communication
FTP	…	File Transfer Protocol
HTTP	…	Hyper Text Transfer Protocol
IANA	…	Internet Assigned Numbers Authority
ICPM	…	Internet Control Message Protocol
IEEE	…	The Institute of Electrical and Electronics Engineers
IMEI	…	International Mobile Equipment Identity
IP	…	Internet Protocol address
IPv4	…	Internet Protocol version 4
IPv6	…	Internet Protocol version 6
KRL	…	KUKA Robot Language
LLC	…	Logical Link Control
MAC	…	Media Access Control
MPLS	…	Multiprotocol label switching
NAT	…	Network Address Translation
TCP	…	Transmission Control Protocol
SMTP	…	Simple Mail Transfer Protocol
UDP	…	User Datagram Protocol
UI	…	User Interface

List of Figures

List of Tables

1.INTRODUCTION

When it comes to the parametric visualization of robot motion, the first thing that comes to mind is a visual demonstration of the robot's capabilities, its working range, accuracy, maximum reach and speed; secondly, a user may consider visualizing the technological process in which the robot is involved; it may also involve training and testing software code.

In any case, the range of applications for this technology is wide, but programs that enable virtual robot management usually come at a high price and are usually only available to large companies. In practice, the visualization of parametric robot movements may be necessary for smaller companies or organizations such as universities and special schools.

The first objective of this thesis is to study the mechanism of interaction between the user and the virtual robot within the free version of SketchUp Make 2017, which has a large number of tools for creating 3D objects, and above all, it has its own API that allows extension the functionality of the program through the use of Ruby code. The second goal is to test the possibility of creating a plugin that can interact with objects within the SketchUp scene in real time and exchange data with external applications through the standard communication model (in our case, through the TCP/IP protocols). The implementation of this task may lead to a more detailed investigation of the possibility of using the SketchUp interface to visualize industrial processes, to realize digital twins or digital factories based on it.

2.SKETCHUP

SketchUp is a tool for creating 3D models of varying complexity and tasks, from simple interior elements to building structures. It has a very intuitive interface that can be managed for several hours. It also supports different languages and includes courses and tips for beginners, which are built directly into the program interface. Another feature of SketchUp is a parametric modeling program that allows you to create fairly accurate models. It also supports a large library of ready-made scripts and plugins that allow users to save time and design more complex objects. The user extends the functionality of the program by using these plugins in parallel with an increased level of understanding. The program also supports API - writing custom plugins and scripts in the Ruby programming language. The pricing policy of the company is very user-friendly; there are paid and free versions. Thanks to this feature set, SketchUp is becoming one of the leaders in its segment.

2.1 Pricing policy

In 2012 Google sold SketchUp to Trimble Navigation. Now Trimble supports 4 versions of SketchUp [1]: Free, Shop, Pro, Studio

- The "Make" version will no longer be supported after 2017. In any case, it has the same range of functions as a "Free" version. One difference between the "Make" version and the "Free" version is that the "Free" version works directly in the web browser and does not work without the Internet. "Make" normal installation application.
- The "Shop" version ($119/year) differs from the "Free" version mainly by more options for data import/export and the possibility of commercial use.
- The "Pro" version ($299/year) has more tools than the previous version.
- The "Studio" version ($1199/year) includes a special design tool that allows you to perform energy, ventilation and heating analyses in the building.

In order to compare the paid version and the free version, only a few important aspects were selected.

Table 2-1 SketchUp «Pro» and «Make» version comparing [1]

	Make	Pro
Pricing	free	299$ / year
Commercial use	No	Yes
Drawing	No	Yes
Dynamic Components	No	Yes
Extensions	No	Yes
Import/export	15 formats	33 formats

"*Dynamic Components*" - a powerful presentation tool that allows you to add the dynamic attributes of the object, such as opening/closing doors, windows, lifting shutters, etc. With this tool it is possible to implement the movement of the robot. If the thesis had been done in the "Pro" version, this tool would most likely have been used and the implementation would have been different from the current version.

"*Extensions*" - a large library of plugins and scripts that allows to extend the SketchUp options. Also, as in the case of "Dynamic Components", there was no access to the library.

2.2 SketchUp API

SketchUp API (Application Programming Interface) - is a set of Ruby modules and classes that allow interaction with the SketchUp project at the Ruby programming code level. This tool extends SketchUp's capabilities, allowing you to access the project, create and edit models, work with geometry and SketchUp instruments in general by creating tasks in Ruby code.

There are currently more than eighty classes in the SketchUp API. See Literature [2] for a complete list.

2.2.1 Scripting tools in SketchUp

Writing your own code in the Ruby programming language is possible with "Ruby Console". This console is the part of the SketchUp application. To open it, go to the SketchUp top menu - "Window" - "Ruby Console". In any case, it is inconvenient to use it for writing scripts.

For ease of working with Ruby code within SketchUp, is recommended to install "Ruby Code Editor" [3]. This is an extension that allows you to write full-format code, install libraries, use snippets, etc.

2.2.2 Ruby – SketchUp interaction

Ruby is an object-oriented language. All data are objects, so everything the user works with has a class. Every function is a method.

Every SketchUp script begins by accessing three basic data structures: *Sketchup*, *Model*, and *Entities*. When a new script is created, it has default code with these structures. It is important to understand this code so that you can write your own scripts.

```
1  # Default code, use or delete...
2  mod = Sketchup.active_model # Open model
3  ent = mod.entities # All entities in model
4  sel = mod.selection # Current selection
```

Figure 2-1 SketchUp API default code

Module “Sketchup”

The methods in the *Sketchup* module provide access to the entire SketchUp application. The most important method in the Sketchup module is "active_model". This method returns the class *Model* that corresponds to a currently open project. If the *Sketchup* module itself represents a SketchUp program, *Model* represents a single SketchUp file (*.skp) that contains all the information about the objects it contains. The methods in the *Model* provide information about the current design and various ways to interact with it. See Literature [4] for more details.

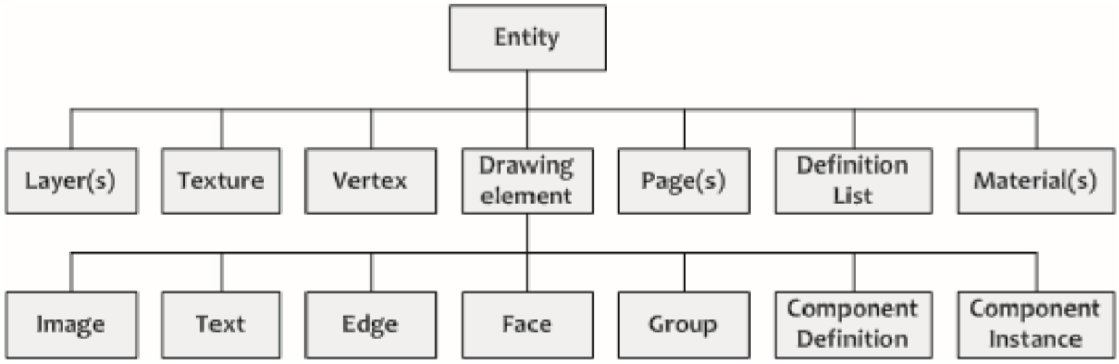


Figure 2-2 Entities of the “Model” class [4]

For example, the entity "materials" controls the materials used in a current project, the entity "layers" controls the visibility and behavior of the layer.

The "Entities" class is also the entity of the "Model" class (in Figure 2-2 it is called "Drawing element"). It represents all the geometric objects in a SketchUp project, i.e. "lines", "faces", "images", "text", "groups" and "components".

Module "Geom"

To change an existing geometry (its position, size, inclination), you need to use the "Transformation" class located in the "Geom" module. The "Transformation" class contains methods such as "Rotation", "Translation", "Scaling", which are used to interact with an object.

Module "UI"

It contains a number of methods for creating simple user interfaces for data entry. The user interface in SketchUp does not support the user entering or changing data in real time. Calling the UI always stops the currently running program, which actually limits the user's ability to interact with the program.

3. TCP/IP MODEL AND PROTOCOL STACK

TCP/IP model - a model of network organization that specifies the layers of a network and the rules that must be followed to achieve proper data transfer between computers, correct merging and separating of networks, and building large composite networks.

The TCP / IP model is a de facto standard; no one has specified a standard for this model. The model includes 4 base layers [5]:

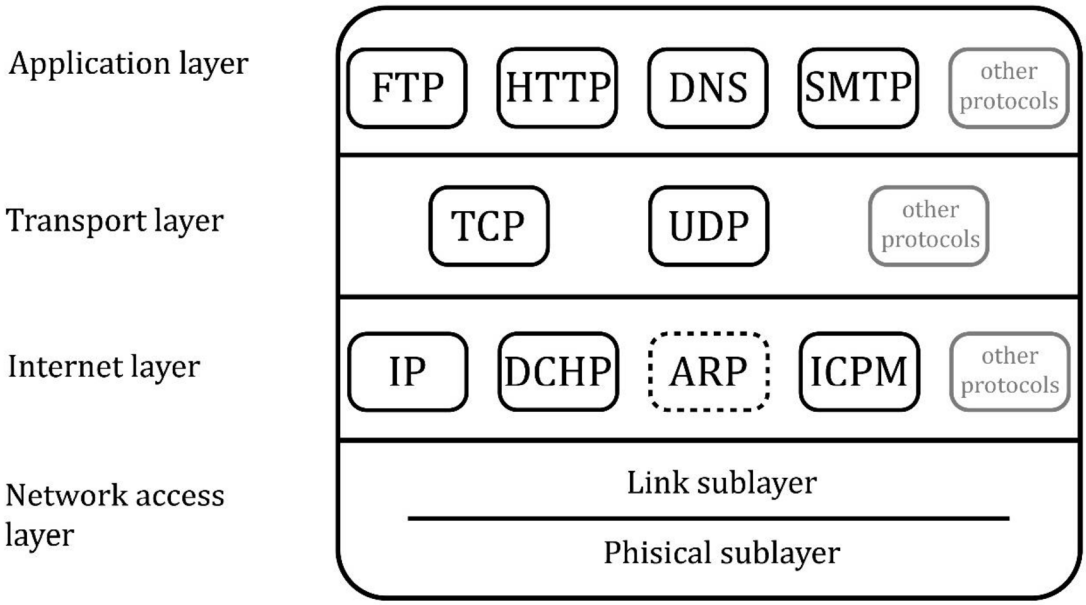


Figure 3-1 Layers of TCP/IP model

Each layer contains more protocols than shown in Figure 3-1. For example, the application layer contains more than 150 protocols [6], and it is not the goal to learn all of them. For this paper had selected some of the most popular protocols that form a basis of the TCP/IP model.

Short description of the layers:

1) ***Network Access layer.*** The layer is designed to interact with network technologies that are not formally part of the TCP / IP protocol stack.
2) ***Internet layer.*** This layer enables addressing in the global network using the IP protocol and additional protocols that ensure the data transmission.
3) ***Transport layer.*** It contains the TCP protocol, which ensures data transmission with a delivery guarantee, and UDP, which enables fast data transmission, but without a delivery guarantee.
4) ***Application layer.*** Contains protocols that serve to work with the received data and represent it to the user.

TCP/IP protocol stack - is not one protocol, but several protocols, of which TCP and IP are the two most important. TSP / / IP is named after the two most popular protocols from the stack

Each of the layers and the most common protocols will be discussed in more detail in future chapters.

3.1 Network Access Layer

In fact, a layer consists of two layers that were taken from the OSI model.

3.1.1 Physical layer

The main task of the *Physical layer* is to represent bits of information in the form of signals that are transmitted via the medium. How exactly the data transmission takes place is not of interest. It is important to know only the parameters of the data channel:

1) Data transfer environment
    - Coaxial cable
    - Twisted pair
    - Optical cable
    - Wireless technology
2) Bandwidth (bit / s)
3) Delay – time of message passing from the sender to the recipient
4) The number of errors
5) Type of communication channel
    - Simplex
    - Duplex
    - Half Duplex

The physical layer does not analyze the information it transfers.

3.1.2 Link layer

Once the problem of transmitting bits over the communication channel has been solved on the *Physical layer*, the question of how to extract a message from the bit stream arises on the *Link layer*.

The main method of detecting a message in a bitstream is to insert a special sequence of bytes or bits at the beginning and end of the frame. In Ethernet, for example, the latter takes 56 bits and is an alternating sequence of 0 and 1.

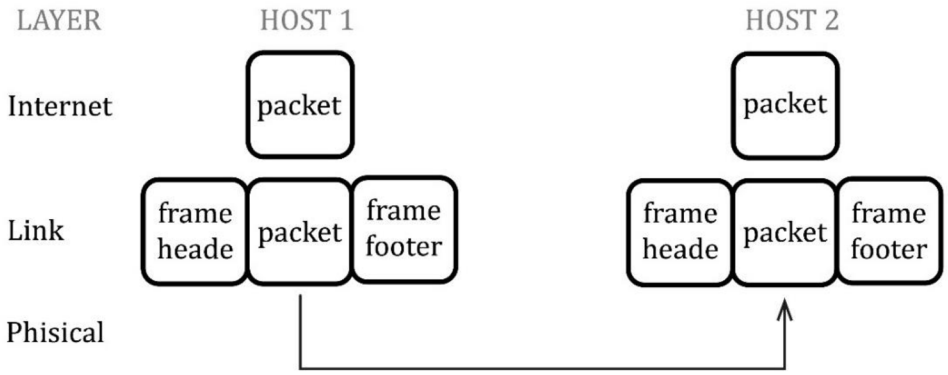


Figure 3-2 Frame transfer structure

The task of the *Link layer* is also to address and consistently access the channel - if there are several devices on the communication channel, it must determine for which device this message is intended and which of the hosts occupies the channel in case of a simultaneous request for data transmission.

3.1.2.1 Logical Link Control (LLC)

This sublevel of the *Link layer* is responsible for data transmission – frame generation, error handling. The level is common to the various technologies. LLC frame is called Protocol Data Unit, PDU and described in the IEEE 802.2 standard.

3.1.2.2 Media Access Control (MAC)

The MAC address is used to identify physical network interfaces of network devices (routers). It is used to define the physical interface for which the data is intended. MAC Addresses are used in common Ethernet and WI-FI *Link layer* technologies.

MAC addresses are regulated by the IEEE 802 standard. The address length is 6 bytes - 48 bits. The recording format consists of six hexadecimal numbers separated by a colon or dash:

$$30 - 5A - 3A - AB - FF - 32$$

$$30{:}\ 5A{:}\ 3A{:}\ AB{:}\ FF{:}\ 32$$

In a network segment the MAC addresses must be unique. If there are two computers with the same MAC address, one of the two computers will not work.

Assign MAC addresses:

- An address is assigned by the hardware manufacturer. The assignment rules are described by the IEEE 802 regulations.
- An address is assigned by the network administrator.

The second bit of the high byte in the MAC address indicates that an address is assigned by the manufacturer - 0 or an address is assigned locally - 1.

To find out the MAC address of your computer in Windows, use the command "ipconfig / all" in the command line.

3.2 Internet layer

The main task of the *Internet layer* is to coordinate differences in technology on the *Network Access Layer*. The layer provides the ability to build a large composite network based on individual local networks. Even if the individual networks use different technologies *of* the *Link layers* such as *Ethernet, wi-fi, 5G/4G/3G, MPLS* etc. In other words, it enables data transfer from an Ethernet network to a WI-FI network.

To negotiate addresses on the *Internet layer,* the concept is used when the address is divided into global and local.

Local address – the *Link layer* addresses. It is associated with a specific data transfer technology. For example, MAC addresses in Ethernet or IMEI addresses in 4G. These addresses cannot be used to build a composite network that uses different technologies.

Global address – an address that is not associated with the *Link layer* technology and allow to build composite networks. In TCP / IP protocol stacks, these are known as IP addresses.

The tasks of the *Internet layer*:

1) *Internetworking* - theory of combining small local networks into global networks.
2) *Address aggregation* - working with addresses blocks, not with individual addresses (address block - network).
3) *Routing* –in large networks there are always several active paths for data transmission from the sender to the receiver. In this situation, the question of choosing the optimal data transmission path arises.
4) Protects the composite network from overload.
5) Detects and prohibits the forwarding of "garbage" packets in the network.

3.2.1 IP protocol

The purpose of the IP protocol is to combine networks that use different Link layers technologies and uniquely identify the device on the network by means of an IP address. The IP protocol allows data transmission without any guarantee of delivery and correct message sequence. IP protocol - the protocol uses data transmission without establishing a connection. If the packet does not arrive for any reason, no attempt is made to notify the sender and no attempt is made to request the packet again. The error must be corrected by higher layer protocols.

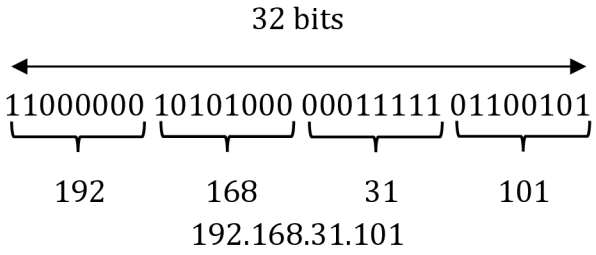
IP address - used to uniquely identify computers in global network [7].

IP version:

- IPv4 - address length 4 bytes
- IPv6 - address length 16 bytes

How IPv4 addresses work:

The length of 4 bytes is 32 bits. To facilitate work with IP addresses, they are represented by 4 parts, each part has 8 bits. This part is called an octet. Each octet is written in decimal format. The entry form consists of 4 octets separated by dots.



Maximum number of IPv4 addresses is $2^{32} = 4\,294\,967\,296$. This number of IP addresses was not sufficient for the whole world. Now almost all IP addresses of version 4 are distributed. Solution - using IPv6 or NAT technology.

Subnet - a set of computers that have the highest part of the IP address the same.

192.168.31.**101**
192.168.31.**102**
192.168.31.**103**

Highest bits - the number of the subnet.

Low bits - the host number - number of individual computers on the network.

To find out where the subnet address and where the host address is, use the subnet mask. The mask has the same length as the IP address and contains 1 where the network number is set and 0 where the host number is set.

To calculate the subnet address from IP and mask, use the AND operation:

IP decimal:	192.168.31.101
IP binary:	11000000.10101000.00011111.01100101
	AND
Mask:	11111111.11111111.11111111.00000000
Subnet binary:	11000000.10101000.00011111.00000000
Subnet decimal:	192.168.31.0

In practice, you write the mask with the IP address using a slash: 192.168.31.101/24, which means that the last 24 bits are part of the subnet and the first 8 are the host number.

Reserved IP addresses [8]:

- 0.0.0.0 – current host. Used when the computer has not yet received its IP address

- 255.255.255.255 – limited broadcast address (all hosts in the current subnet)
- 127.0.0.0/8 – “Loopback” network for testing. Data is not sent to the network but comes back to the computer. Often used 127.0.0.1 – localhost
- 198.18.0.0/15– used for benchmark testing of inter-network communications between two separate subnets
- 169.254.0.0 – link-local address is a network address that is valid only for communications within the network segment or the broadcast domain that the host is connected to
- 10.0.0.0/8; 172.16.0.0/12; 192.168.0.0/16 - local communications within a private network. Not routed on the Internet, used internally without contacting IANA. It is possible to connect a network built based on private addresses to the Internet by NAT technology

3.2.2 DCHP protocol

When you connect your computer to a new network (for example, when you connect to wi-fi in a restaurant), it must be given an IP address to work on the network. The address can be assigned manually or automatically using the DHCP protocol.

DCHP client-server technology is used to assign IP addresses.

- DCHP client - computer that receives an IP address
- DHCP server - assigns IP addresses to computers and maintains a table of dedicated addresses to avoid duplication
-

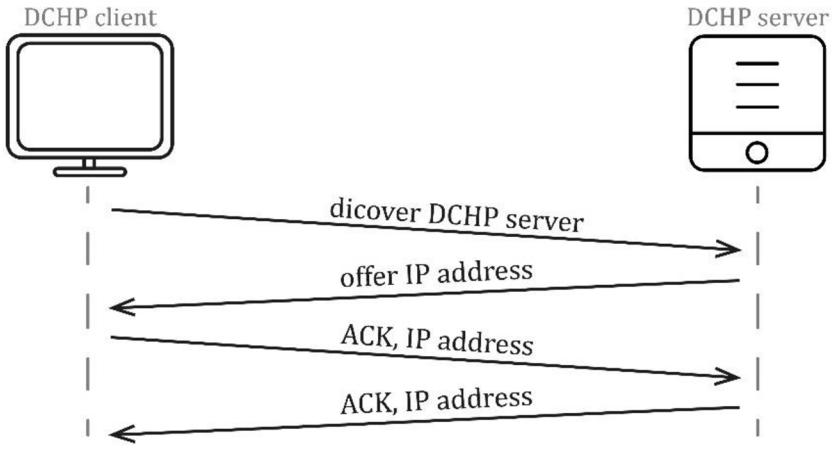


Figure 3-3 The process of obtaining an address in a network

When a client connects to a new network that it has no information about, the client sends a request to the broadcast address to browse the DCHP server. After receiving the request, the DHCP server sends its IP address to the new client. After

that, a few confirmation messages go out. The IP address is assigned in the network for a limited time. After the time expires, the IP is taken back from the user and can be passed on to another client.

3.2.3 ARP protocol

Allows you to automatically determine the MAC address of a computer based on its IP address. An ARP request with an IP address is sent to the broadcast MAC address (FF:FF:FF:FF:FF:FF). All computers on the subnet receive this request. A computer that finds out its IP address sends an ARP response with its MAC address.

The ARP protocol allows you to find out the MAC addresses of computers that are on the same network. Reason: The broadcast traffic does not go through the router, so computers in another ARP subnet do not receive the request.

ARP table - a table that stores the correspondence of MAC -IP addresses and their type (static - set manually, dynamic - received by an ARP request).

"*arp -a*" - Windows command to display the ARP table on the computer.

3.2.4 ICPM protocol

IP protocol used only for data transmission over a network without delivery guarantee. Sometimes it is necessary to inform sides about errors that occurred during data transmission, or to test or diagnose a low-layer network without using higher layer protocols. The ICPM protocol is used for this purpose.

"*ping*" - Windows command that informs if a computer is available on the network.

"*tracert*" - Windows command that allows to determine the route from the sender to the receiver. Returns a list of all routers the packet passes through.

3.2.5 Routing

Hardware that works on the Internet layer is called a router. Routers are used to connect local networks to a global network. Each device has several connection interfaces through which computers connect to it. Each of these interfaces has its own local and global address.

Routers are used to deliver data from one network to another. It also analyzes the topology of the network in which it operates. The structure of the global network may change - new routers may appear; old routers may fail. The next important task is load balancing in the network - the efficient use of network bandwidth. The path search for each data packet is performed independently.

Figure 3-4 explains how data is transferred from one network to another.

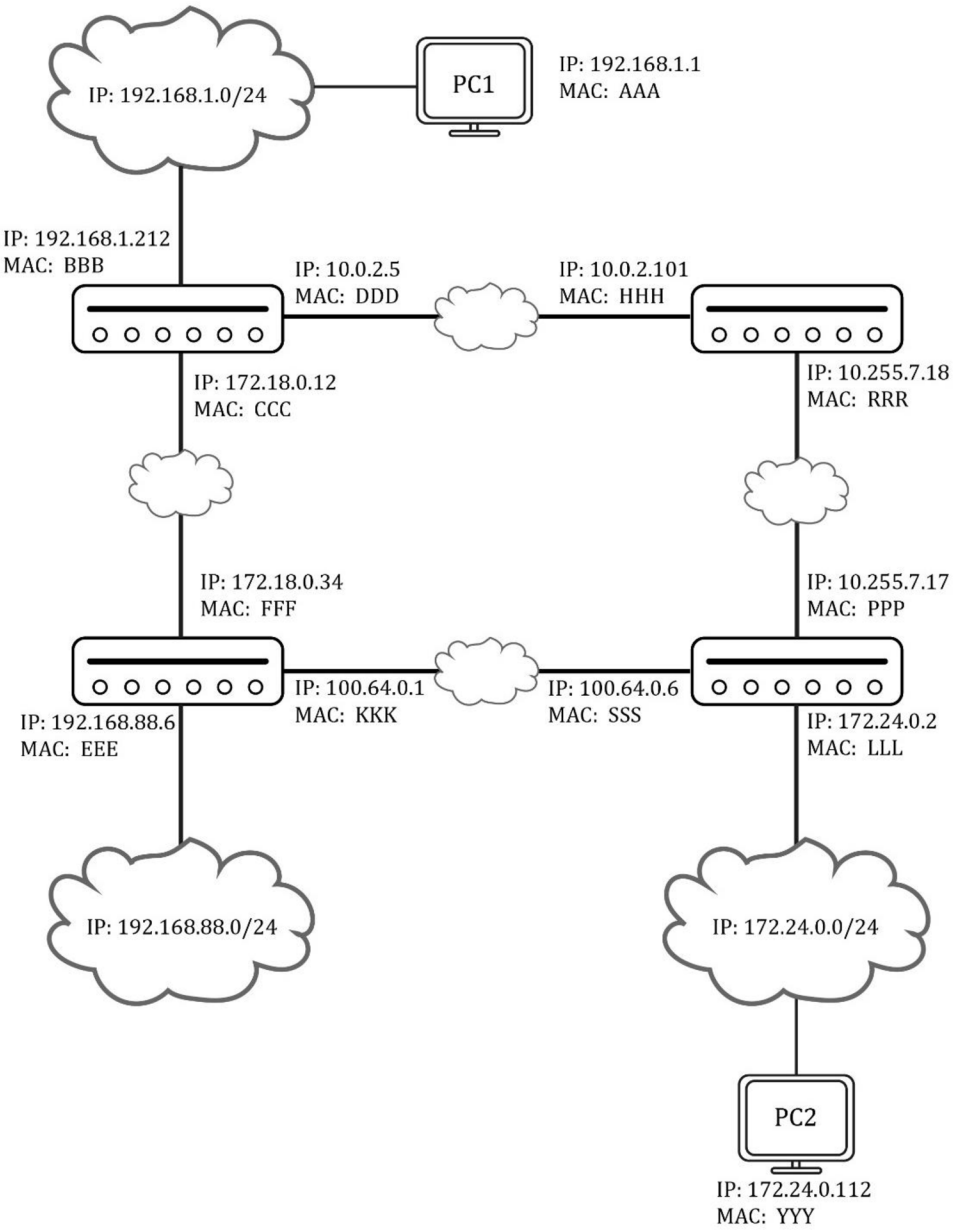


Figure 3-4 Routing

Data transfer principle [9]:

To transfer data from PC1 to PC2, it must know an IP address of PC2. First PC1 packs the packet into headers with its own IP address, the IP address of the destination computer (PC2), its own MAC address and MAC address of the next router.

Abstract scheme:

Source IP address	Destination IP address
MAC address of current package owner	MAC address of next router

First step – packet transfer from computer to router:

192.168.1.1	172.24.0.112
AAA	BBB

Now data on the first router. It has two ways to deliver data: via subnet 172.18.0.0/24 or 10.0.0.0.0/24. To select a path, use the values of the metric field in the routing table. Metric is a number that describes the distance and number of routers from one network to another. For example, it was decided that subnet 172.18.0.0/24 is better.

Second step – packet transfer from one router interface to another:

192.168.1.1	172.24.0.112
BBB	CCC

Third step – packet transfer from one router to another:

192.168.1.1	172.24.0.112
CCC	FFF

It continues until the last router receives a packet and finds the same subnet address as the destination address in its own *router table* and sends the packet from that network directly to the computer.

The *router table* contains IP and MAC addresses of networks and other routers connected to the interfaces of this router.

Last step:

192.168.1.1	172.24.0.112
LLL	YYY

3.2.6 Fragmentation

Splitting a single package into several segments. If the packet size exceeds the maximum packet of the interface it is currently passing through, it is split into smaller segments. Fragmentation at the *Internet layer* is hidden from the sender and receiver. You do not need to know which networks the data must pass through and what size of data can be sent over these networks.

3.3 Transport layer

The *Transport layer* has the task of transferring data between processes on hosts. When the data packet is received on the computer, it must understand for which application or process it is intended. "*Port*" is used to address specific processes on a single computer. A port is a number from 1 to 65535 that represents the "address" of an application or process on the computer. Port numbers on a single host must not be repeated. To connect to the service on another computer, the original computer must know the IP address of the computer and the port on which the service is running.

A port is written after the address with a colon:

192.168.31.101: 8081

IP address port

The *Transport layer* provides a "pass-through connection" between two hosts. There can be multiple network devices between two interacting hosts, but they do not interfere with the *Transport layer*. The *Transport layer* allows you to hide details of network interaction from developers.

3.3.1 UDP protocol

The UDP protocol specifies the port of the sender and the port of the receiver. It does not provide a delivery guarantee, and there is no guarantee that the sequence of packets will be followed.

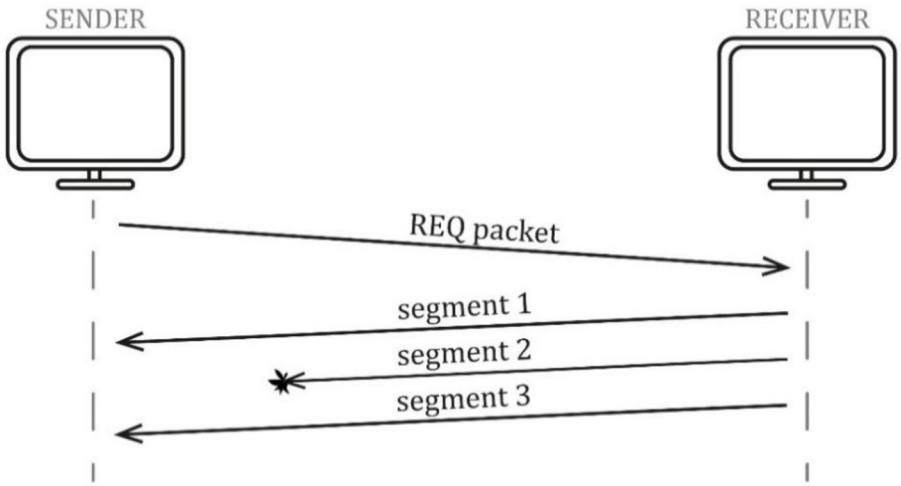


Figure 3-5 UDP protocol

Table 3-1 UDP header

source port	destination port
len	checksum

Data

- *len* - total length of the UDP header
- *checksum* - is used to check whether the data is supplied correctly. If the checksum calculated by the receiver does not match the checksum in the tcp header, this segment is removed.
- *data* is followed after UDP header

3.3.2 TCP protocol

Both the TCP protocol and the UDP protocol are used to specify the port of the sender and the port of the receiver. The TCP protocol takes care of the data transmission and guarantees the correct order of the packets.

Before data can be sent via the TCP protocol, a connection must be established. The connection is established by setting a SYN flag in a TCP header. The connection fulfills the following functions:

- Agreement on the numbering of the segments
- Agreement on the maximum segment sizes

Proper connection termination consists of setting a FIN flag and a two-way confirmation on the hosts. An RST flag is used for one-way (emergency) connection termination.

Table 3-2 TCP header

source port			destination port
sequence number			
acknowledgement number			
len	reserved	flags	window
checksum			urgent pointer
[options]			

data

- *sequence number* – first byte number in the segment. Specifies the location of the segment in the byte stream
- *acknowledgement number* – number of the next expected byte
- *len* – total length of the TCP header
- *flags*:
    - URG – flag indicates that packet contains urgent data that needs to send to application first. This flag is used with the urgent pointer
    - ACK – acknowledgement
    - PSH – indicate that data must be passed to the application without writing to the buffer
    - SYN – establish a connection
    - RST, FIN – break a connection
- *window* – in this field, the recipient specifies how much data it can accept
- *checksum* – is used to check whether the data is supplied correctly. If the checksum calculated by the receiver does not match the checksum in the tcp header, this segment is removed
- *common options*:
    - maximum segment size
    - selective confirmation

The TCP protocol receives a stream of bytes from an application. This stream is divided into separate segments and sent separately to a receiver. A receiver accepts these segments, collects them into one large stream of bytes and sends this stream of bytes to an application. After a segment is sent, a timer is started on the transmitter side. If no acknowledgement response is received after the timer expires, the segment is retransmitted. In practice, an acknowledgement response from the receiver is not sent after each segment, but only after several segments have been received to save time (cumulative acknowledgement). There is also a selective acknowledgement, which is used to acknowledge a single segment from the stream.

Confirmations and resending are not sufficient for reliable data transmission. This method only guarantees the delivery of segments, not their sequence. Therefore, each segment is numbered to avoid duplication and to maintain their order in the byte stream when a segment is reached faster than the previous one. The numbering is by the first byte of the segment from the byte stream.

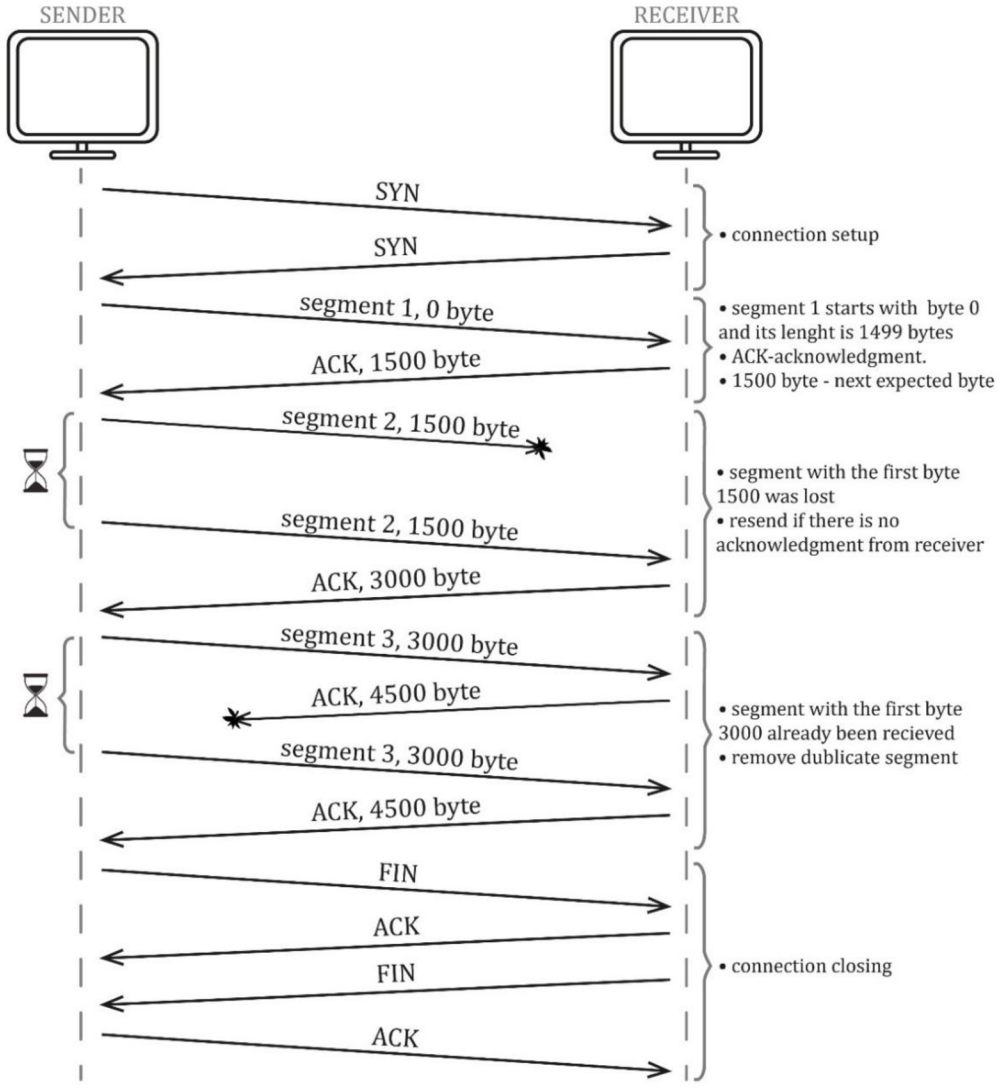


Figure 3-6 TCP protocol

Interaction with the transport layer

The *Transport layer* is the first level with which the programmer can interact with. The interface of the *Transport layer*, which allows you to write programs for the network, is called the "***socket interface***". The *socket interface* is discussed in the next chapter.

3.4 Application layer

The application layer is used for the interaction between network applications and data that represent for users. Examples of such applications can be a Web browser that uses the HTTP protocol to transfer http files and demonstrate Web pages, or mail services that use the SMTP protocol to transfer mail. In this paper some information about two protocols is given to get an idea of what kind of protocol works at the *Application layers*.

3.4.1 FTP protocol

The protocol uses the client-server model. The server gives the client access to its file system - the catalog structure in which the files are located. The Client must authenticate, enter a user name and password, and then the Client is given access to the structure where it can change directories, upload, modify and download folders and files. The URL is used to address the files. URL is a path to the file written as: ftp://lockalhost.com/

An example of a program that uses the FTP protocol is FileZilla. Example of an FTP server at a university BUT Figure 3-7. The student gets access to FTP servers after user identification.

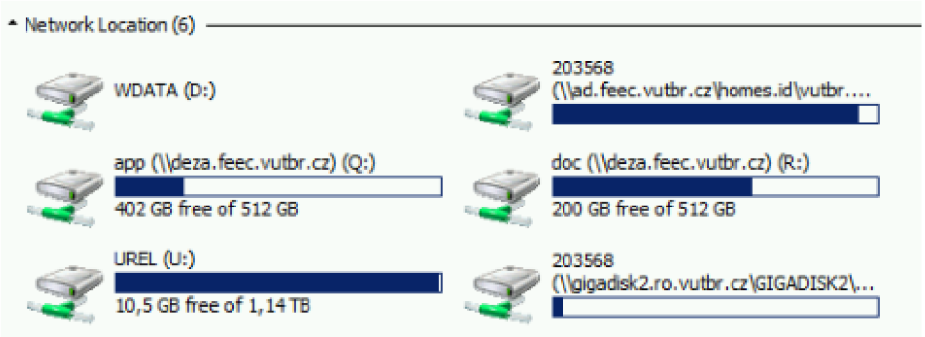


Figure 3-7 FTP servers at the BUT

3.4.2 DNS protocol

For addressing on the Internet, it is difficult for users to use a numerical representation of addresses. It is easier to remember a meaningful letter name. This

is done by the DNS protocol, which allows the use of letter names that correspond to the IP addresses of servers and computers.

"*nslookup*" - Windows command that allows to find out the address of a computer or server by its domain name. If you set "nslookup www.fekt.vut.cz" at the command line, it will say that for the domain name "www.fekt.vut.cz", an address will correspond to "147.229.71.28". Use this address in the URL field of a browser to open our faculty website.

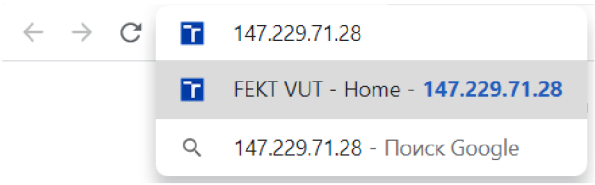


Figure 3-8 Domain name and IP address

3.5 Conclusion

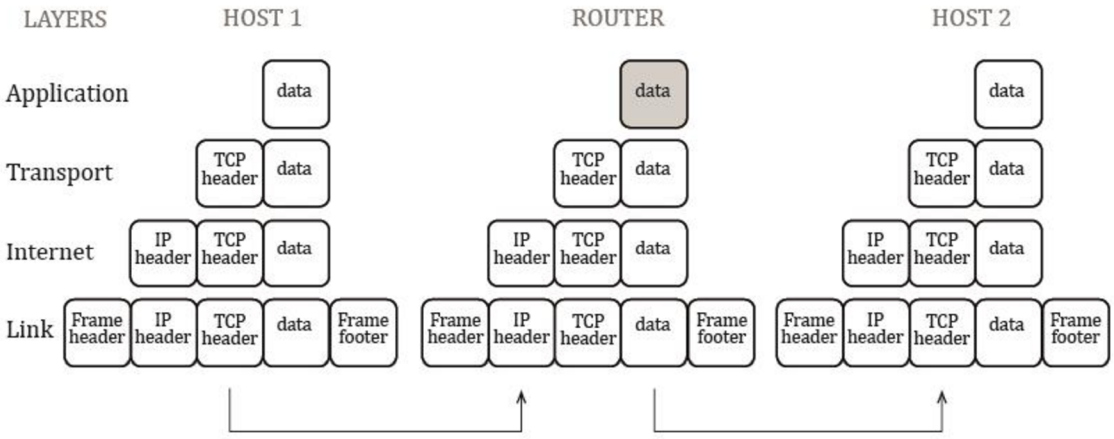


Figure 3-9 TCP / IP protocol stack communication model

The data is transferred between two hosts (computers) via network routers. There may be several routers on the path (in our case one). In most cases, a router is a device that operates on the first three layers of the TCP/IP model. On the Link, Internet and Transport layers and does not process received data packets, but only redirects them. However, there are also routers that work on the Application layer. These could be content filters, for example - devices that analyze data traffic. Content filters can restrict access to certain resources.

TCP headers - the first layer containing the address of the specific application on the recipient host for which the data is intended (port). The next layer, the IP header, contains the global address of the host to which the data is to be delivered. The last layer is the frame header/footer, which adds local network device addresses (MAC addresses), which send packets from one router to another until a device with the specified global address is reached. The MAC addresses along the path change after each router, and the global IP address does not change.

4.SOCKET INTERFACE

Socket is an interface (standard) for the interaction between programs and the TCP / IP *Transport layer* [10]. Sometimes a socket is called the "endpoint" of network communication. The communication model that uses the socket interface is the server-client communication model.

Server - a program on a computer with a known IP address and a port that waits for a connection request in passive mode.

Client - a program that connects to the server.

Socket operations are divided into several stages:

The first stage — sockets creation:

1) *Socket* – creating a new socket
2) *Bind* – binding an IP address and port to a server socket
3) *Listen* – declaring that the socket is ready to connection

The second stage — establish connection:

4) *Accept* – accepting a connection request from the client
5) *Connect* – request to establish a connection with server

The third stage — data transmission:

6) *Send* – send data over the network
7) *Receive* – get data over the network

The fourth stage — closing a connection

8) *Close* – closing a connection

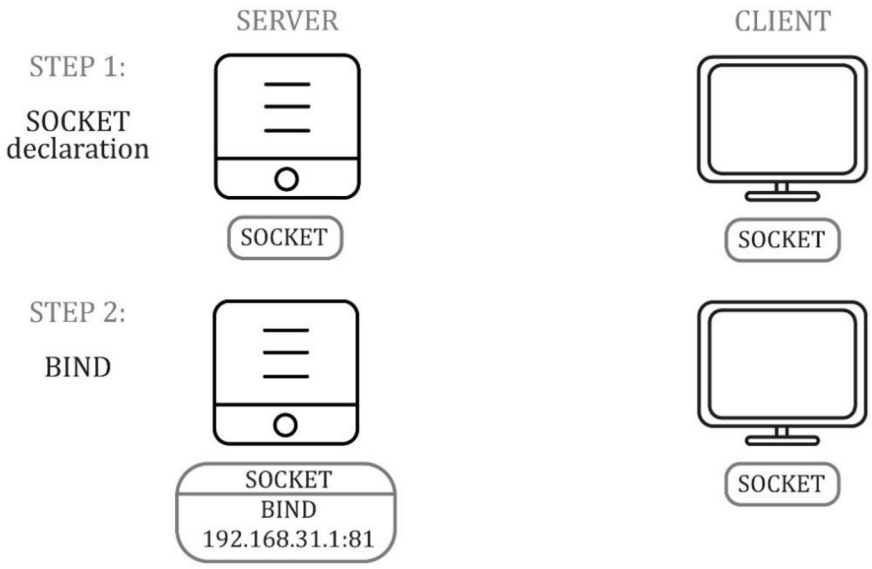


Figure 4-1 Socket point to point communication principle (part 1)

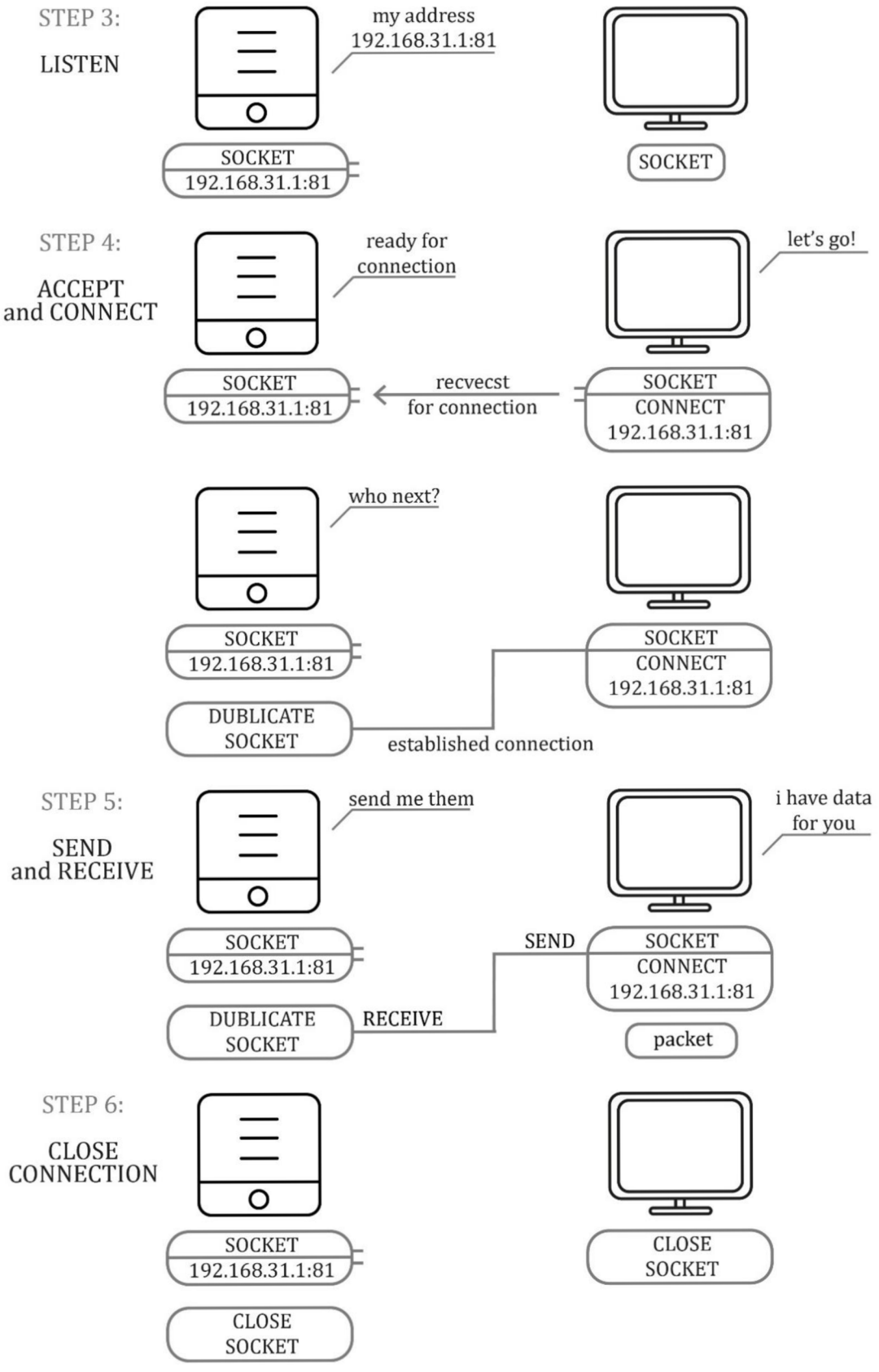


Figure 4-2 Socket point to point communication principle (part 2)

5. ROBOT KUKA KR6

5.1 Real robot specification

This abstract gives the reader an idea of the real robot behavior. Its industrial purpose, technical parameters. Its navigation, communication and programming tools.

KUKA KRC6 R900 is an industrial robot that belongs to a group of small robots with low payload, with high speed, repeatability and accuracy. The most common purpose of this robot is painting, gluing, welding, packaging, sorting and measuring.

Table 5-1 Technical data [11]

Maximum reach	901.5 mm
Maximum payload	6 kg
Pose repeatability	± 0.03 mm
Number of axes	6
Mounting position	Floor, Inverted, Angle
Robot mass	52 kg
Footprint	320 mm x 320 mm
Protection rating	IP54
Ambient temperature during operation	5 °C to 45 °C

Axis range - range of motion in degrees or millimeters that defines the maximum and minimum rotation angle for each axis.

Table 5-2 Axis range [11]

Axis 1	± 170°
Axis 2	+ 45° / -190°
Axis 3	+ 156° / -120°
Axis 4	± 185°
Axis 5	± 120°
Axis 6	± 350°

Workspace - define area within which a robot may move and how far it can reach. Workspace based on the axis range.

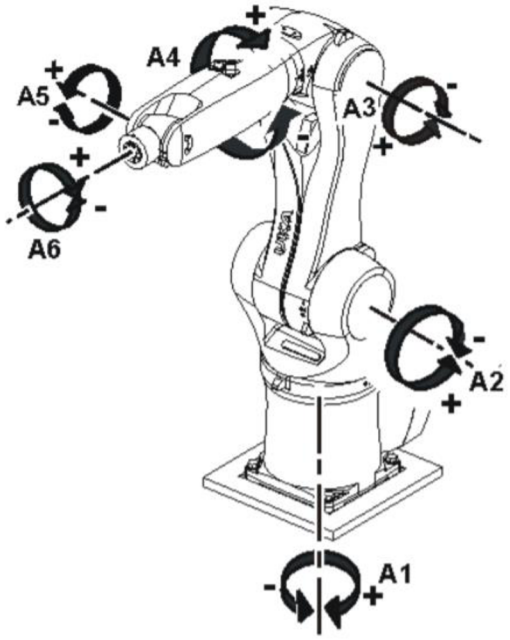


Figure 5-1 Rotation direction of robot axes [11]

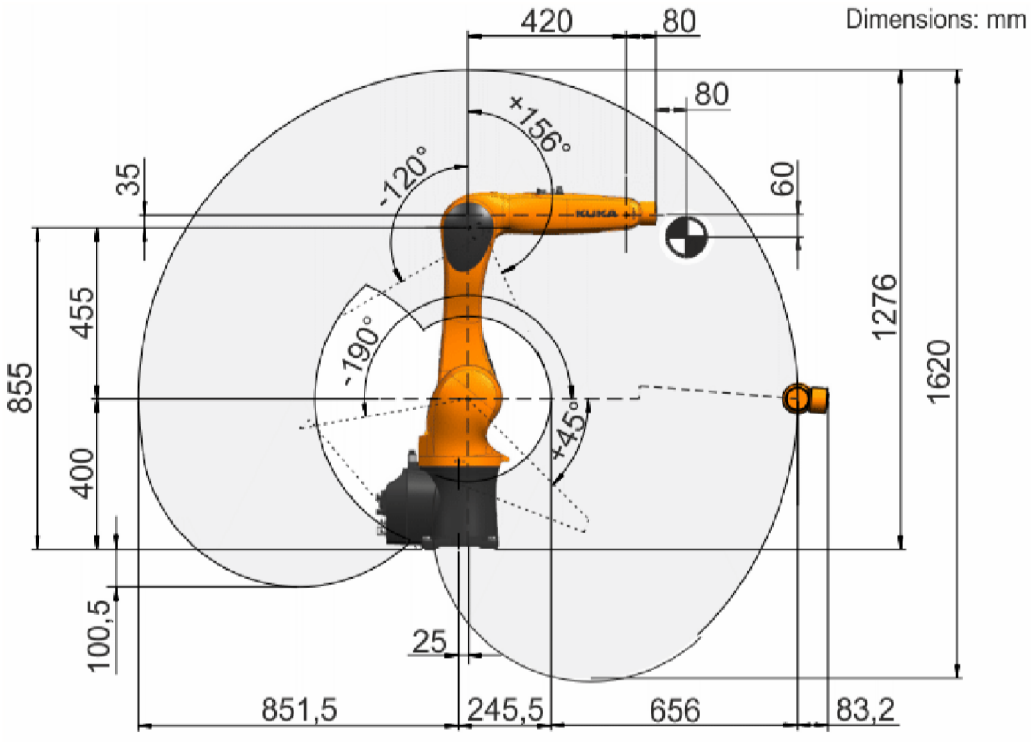


Figure 5-2 Workspace, side view [11]

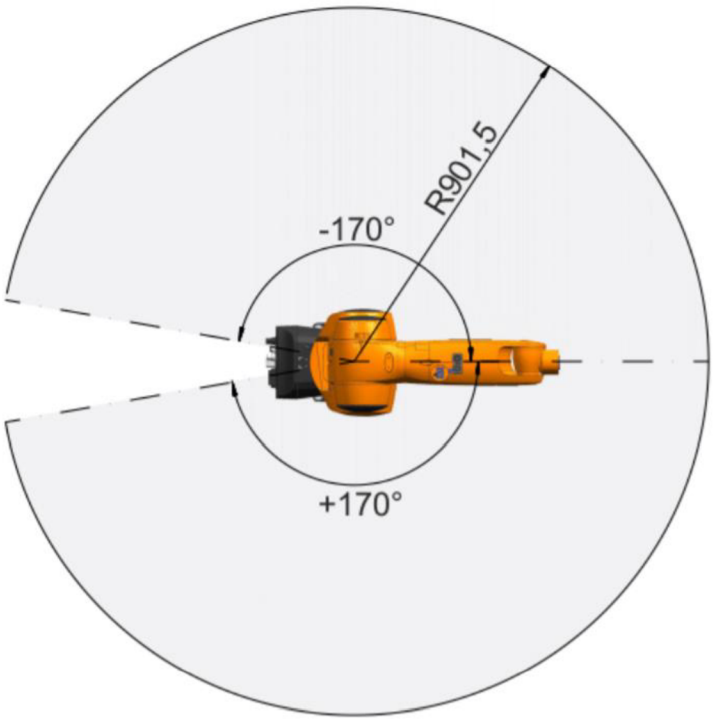


Figure 5-3 Workspace, top view [11]

The workspace and range of motion of each joint tells you how far it can reach. This information results in correct robot positioning in SketchUp dimensions. It can be used to control the collision of robot parts within the motion.

5.2 3D model of KUKA KR6

The 3d model of the robot as a part of the thesis was provided by the BUT. The original format of the model is "stl".

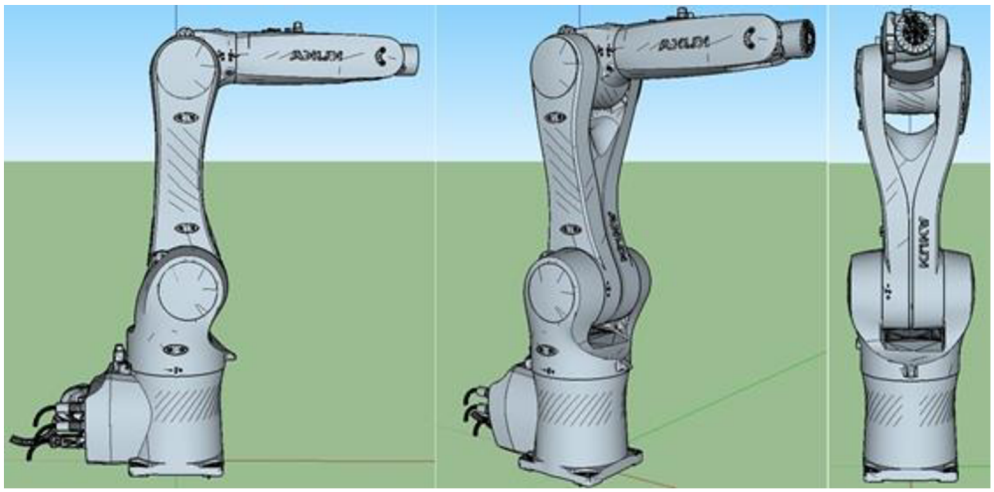


Figure 5-4 3D model of the KUKA KRC6

To implement the robot motion in SketchUp, it was necessary to transfer the control method of a real robot in three dimensions. The design of this "control method" is based on a simple term "changing the position of an industrial robot", which primarily means changing the angles of the individual joints. In order to represent a similar working principle in the virtual model, it was decided to redesign the model and to add rotation points in places where the joints are connected. These points are used as pivot points inside SketchUp.

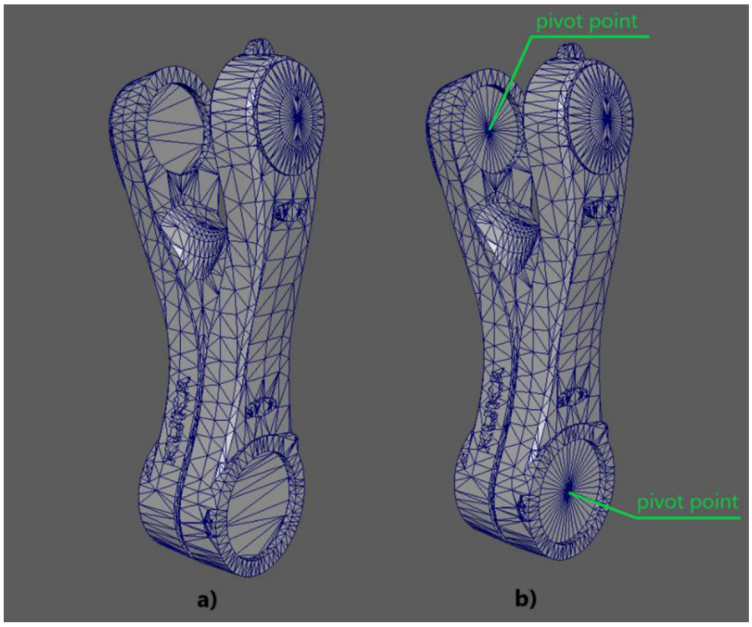


Figure 5-5 Adding central rotation points – pivot points. a) before b) after

Axis of rotation of a component in SketchUp, formed by a pivot point on a model and a vector starting from that point. Further change in the angle of the joint occurs around this vector.

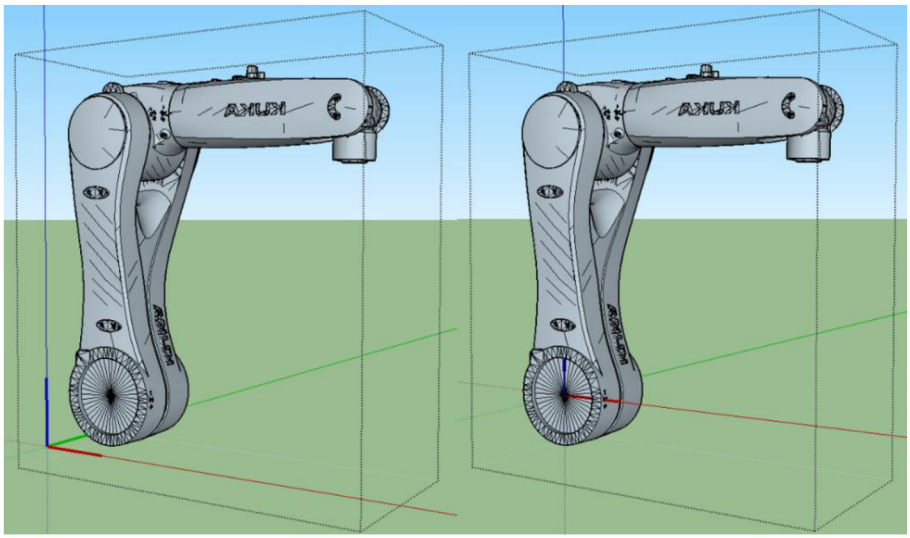


Figure 5-6 Changing the origin of an axis in a SketchUp component

Each joint has its own pivot point and axis of rotation. These axes are firmly connected to components.

Ultimately, the problem arises when a model part has to be held in the correct position relative to other parts during rotation. The solution to this problem is based on the ability to group objects (create components) and create a hierarchy in SketchUp. Component can contain not only model objects, but also other components. By inserting one component into another, a hierarchy tree is created.

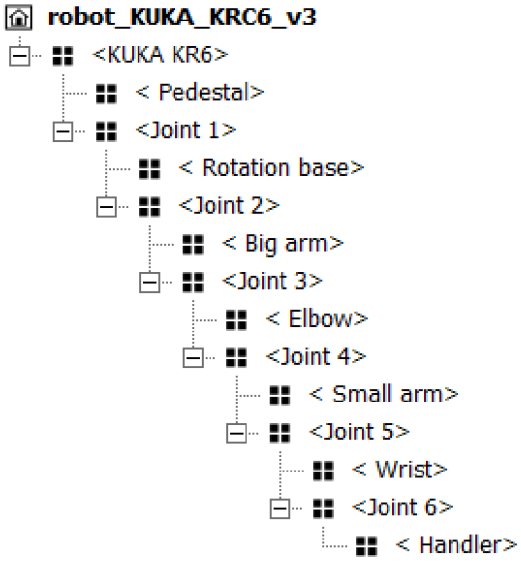


Figure 5-7 Components hierarchy in SketchUp

Now the robot consists of the 7 components - "KUKA KR6", "Joint 1" - "Joint 6". Each component has its own pivot point and its own axis of rotation. Components that are higher up in the hierarchy cover all components that are lower down. For example:

"Joint 2" is the component consisting of the model part "Large Arm" and the next component "Joint 3", at the same time the component "Joint 3" consists of the model part "Elbow" and the next lower component "Joint 4".

Consequently, when the SketchUp target is directed at a component, it selects all model parts below it in the hierarchy and allows all embedded elements to rotate fixedly around the axes of the highest selected component.

This design has two advantages:

- for the motion execution only the rotation vector and angle for each joint must be specified
- thanks to strong component binding, it is not possible to lose model integrity

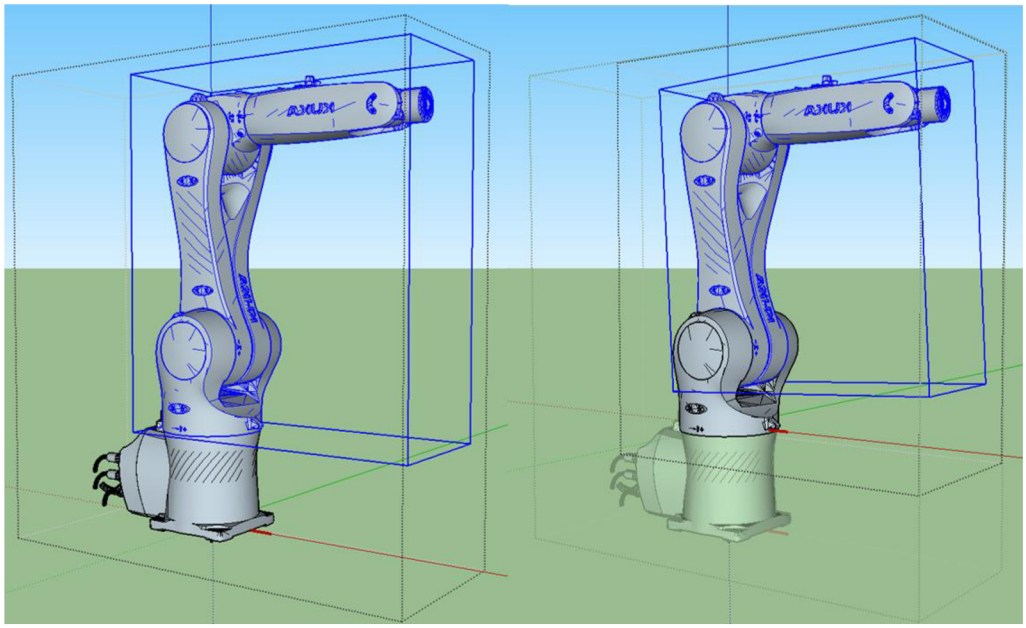


Figure 5-8 Hierarchical structure in SketchUp

6. IMPLEMENTATION

The data is transmitted according to the TCP/IP model and via socket interfaces.

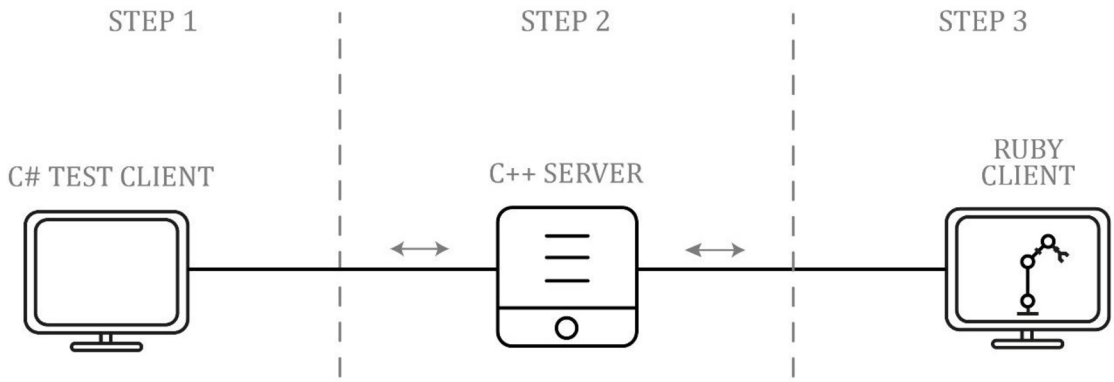


Figure 6-1 Communication scheme

Ruby client - the 3D model of the robot in SketchUp is controlled by a plugin that simultaneously performs the function of a TCP client (it has a socket interface through which it connects to the server) and a script that converts data received from the server into angles for each joint of the robot.

C# test client - is a TCP test client that also uses a socket interface through which it connects to the server. The C# application also has a user interface that allows the user to set the angles for each joint. Data in the form of a string is transmitted to the server via the TCP protocol.

TCP server - acts as a relay and does not process data received from clients. Several clients can be connected to the server. Each client is connected to a separate port and is processed in a separate thread. Connected clients are included in the array of established connections - the array of sockets. When the server receives data from the C# client, this message is forwarded to all other clients, one of which is the Ruby client. In this case, other clients receive information from the server about changes in the robot's position by one of the other active clients. The Ruby client interprets the received data into angles and changes the position of the robot.

6.1 TCP/IP server in C++

To build a TCP / IP server in C++, the Winsock API will be used.

6.1.1 Required headers

#pragma comment (lib, "Ws2_32.lib") – instructs the linker to add the library "Ws2_32.lib" to the list of library dependencies.

<WinSock2.h> – Winsock API. This header contains functions like accept, bind, listn, recv, send etc.

<WS2tcpip.h> – the header file contains definitions introduced in the WinSock 2 Protocol-Specific Annex document for TCP/IP, which contains newer functions and structures used to retrieve IP addresses.

"iostream"

<string.h>

6.1.2 Initialize the wsock32.dll library

```
WSAData data;
WORD version = MAKEWORD(2, 2);
int winsock = WSAStartup(version, &data);
if (winsock != 0) {
    std::cerr << "WSAStartup failed" << std::endl;
    return WSAStartup_failed;
}
```

Figure 6-2 Initialize the wsock32 library

WSAData data – structure contains information about the Windows Sockets implementation.

MAKEWORD(a, b) – macro containing a version of the Winsock interface. Byte a - version, byte b - under. version. Possible versions are 1.0, 1.1, 2.0, 2.2. Later versions have new functions and extension mechanisms.

WSAStartup – function for initializing Winsock. Returns 0 if the initializing was successful.

6.1.3 SOCKET function

If the socket declaration was successful, this function returns a socket descriptor – a non-negative integer number. If an error was detected during the operation, the function returns "-1" (*INVALID_SOCKET*).

```
Connections = socket(AF_INET, SOCK_STREAM, NULL);
if (Connections == INVALID_SOCKET) {
    std::cerr << "Socket Connections. Getting discriptor failed";
    std::cerr << "Error: " << WSAGetLastError() << std::endl;
    WSACleanup();
    return Discriptor_failed;
}
```

Figure 6-3 Socket declaration

Parameters:

1) Address family

- *AF_INET* – The Internet Protocol version 4 (IPv4) address family
- *AF_INET6* – The Internet Protocol version 6 (IPv6) address family.

2) Type – specification for the new socket.
   - *SOCK_STREAM* – with establish connection. Socket type that use TCP protocol.
   - *SOCK_DGRAM* – without establish connection. Socket type that use TCP protocol.
3) Transport protocol. If this argument is set to 0, then the default protocol will be used:
   - *IPPROTO_TCP* for *SOCK_STREAM*
   - *IPPROTO_UDP* for *SOCK_DGRAM*

Two variables of SOCKET type must be declared for the server-client connection. One is used to listen on an open port and the other to accept a connection.

6.1.4 SOCKADDR_IN structure

This structure contains parameters of the socket (server)

```
char server_ip[] = "192.168.0.101";
int server_port = 8082;
SOCKADDR_IN  parameters;
parameters.sin_family = AF_INET;
parameters.sin_port = htons(server_port);
parameters.sin_addr.s_addr = inet_addr(server_ip);
```

Figure 6-4 SOCKADDR_IN structure

Structure methods:

sin_family – defines the address family (protocol suite). For TCP/IP it must be AF_INET or AF_INET6.

sin_port – contains port number

sin_addr – contains address (IP). To represent addresses in numerical form, use the function inet_addr.

```
SOCKADDR_IN client;
int clientsize = sizeof(client);
```

Figure 6-5 SOCKADDR_IN client structure

SOCKADDR_IN client – empty structure that will contain parameters of a new client after connection.

6.1.5 BIND function

Function associates *SOCKADRR_IN* structure, that contains server properties, with a socket.

```
int error_bind = bind(Listen, (sockaddr*)&parameters, sizeof(parameters));
if (error_bind == SOCKET_ERROR) {
    std::cerr << "Socekt Listen. Bind failed.";
    std::cerr << "Error: " << WSAGetLastError() << std::endl;
    closesocket(Listen);
    WSACleanup();
    return Bind_failed;
}
```

Figure 6-6 BIND function

Parameters of “bind” function:

1) Socket descriptor
2) *SOCKADDR_IN* structure
3) *SOCKADDR_IN* structure size

If successful, function returns 0, otherwise - "-1". If the return value is -1, it is necessary to close the declared sockets.

6.1.6 LISTEN function

Function used by the server socket to inform the OS that it is waiting ("listening") for communication requests on the agreed port. Without such a function, any request to communicate with this socket will be rejected.

```
int error_listen = listen(Listen, SOMAXCONN);
if (error_listen == SOCKET_ERROR) {
    std::cerr << "Listening failed.";
    std::cerr << "Error: " << WSAGetLastError() << std::endl;
    closesocket(Listen);
    WSACleanup();
    return Listen_failed;
}
```

Figure 6-7 "listen" function

Parameters:

1) Socket for “listening”.
2) SOMAXCONN – is a positive integer that determines how many communication requests can be received on the socket simultaneously. This number is not related to the number of connections that the server can support. This argument refers only to the number of connection

requests that arrive simultaneously. The number of connections established may exceed this number.

6.1.7 ACCEPT function

The function extracts first connection request from the queue and returns a descriptor to the new socket that has the same properties as the socket specified by the first argument. This new descriptor must be used in subsequent data exchange operations.

```
Connections = accept(Listen, (sockaddr*)&client, &clientsize);
if (Connections == SOCKET_ERROR) {
    std::cerr << "Accept failed.";
    std::cerr<< "Error: " << WSAGetLastError() << std::endl;
    closesocket(Connections);
    WSACleanup();
    return Accept_failed;
}
```

Figure 6-8 “accept” function

Parameters:

1) “listening” socket descriptor.
2) Structure describing the address of the client socket through which he has made his connection request. For TCP/IP networks this is the sockaddr_in structure.
3) The size of this structure.

If the request queue is empty, the program switches to the state of waiting for requests from clients. If acceptance by the client has failed, the function returns a negative value. If the connection was successful, the new connection will place the client into the socket array.

```
Connections_count[client_count] = Connections;
client_count++;
```

Figure 6-9 Storage new client socket in sockets array

6.1.8 CreateThread function

Provide function to work with a new client in the thread.

The *ReciveMessage* thread function implements the continuous reception of messages from the client using the *recv* function and forwarding them to other clients using the send function. These two functions are part of the infinite loop.

The *ReciveMessage* function accepts one parameter - the index belonging to a client in the socket array.

```
HANDLE thread_handle = CreateThread(NULL, NULL,
(LPTHREAD_START_ROUTINE)ReceiveMessage, (LPVOID)(client_count - 1), NULL, NULL);
if (thread_handle == NULL) {
    std::cerr << "Thread hendl failed.";
    std::cerr << "Error: " << WSAGetLastError() << std::endl;
    WSACleanup();
    return Thread_failed;
}
```

Figure 6-10 "CreatThread" function

Parameters:

1) Security descriptor.
2) Initial stack size, in bytes. If this value is zero, the new thread uses the default stack size of the executable program.
3) Function to be executed by the thread. In this case function "ReciveMessage"
4) Variables that belong to thread function. In this case parameter is client's descriptor.
5) Flag
6) Variable that will get an identifier (id) of the thread.

Function returns handle (descriptor) to the new thread or returns NULL if the function failed.

6.1.9RECV function

Receive data from a network communication partner.

```
int recive = recv(Connections_count[client_id], buffer, sizeof(buffer), 0);
if (recive == SOCKET_ERROR || recive == 0) {
    closesocket(Connections_count[client_id]);
    std::cout << std::endl << "Client disconected.";
    std::cout << "Client ID : " << (int)Connections_count[client_id] << std::endl;
    break;
}
```

Figure 6-11 "recv" function

Parameters:

1) Security descriptor through which data is received
2) Pointer that points to a valid area of memory to accommodate received data.
3) The length of this area in bytes.
4) Flags

The function returns the number of bytes declared in the third parameter. If the message contains 0 bytes, this means that the client has completed the

connection. If the client disconnects, the server places the message with the client ID in the console.

6.1.10 SEND function

Send data to a network communication partner.

```
for (int i = 0; i < client_count; i++) {
    send(Connections_count[i], buffer, sizeof(buffer), NULL);
}
```

Figure 6-12 “send” function

Send data to all clients that are in socket array by looping.
Parameters:

1) Security descriptor for which data is sending
2) Pointer that points to a memory area that accommodate sending data
3) The length of this area in bytes
4) Flags

6.1.11 CLOSESOCKET

If the socket descriptor is a positive number, then the socket is not closed.

```
for (int i = 0; i < client_count; i++) {
    if ((int)Connections_count[i] > 0) closesocket(Connections_count[i]);
}
if ((int)Connections > 0) closesocket(Connections);
if ((int)Listen > 0) closesocket(Listen);
WSACleanup();
```

Figure 6-13 “closesocket” and “WSACleanup” functions

Call the *closesocket* function if the descriptor number is greater than 0.
WSACleanup – test function. Do the same as *closesocket* function.

6.2 C# Test Client

This application implements a user interface for testing the server side. Testing the correct client connecting, disconnecting, sending data, real-time process data flow, receiving and interpreting data.
Application contains next functionality:

- IP field for an IP address of server
- PORT filed for a port of the server process
- Connect – button to connect to the server

- Info – information lines. Contains information about success connecting and disconnecting
- Set position – column where the user can enter his own values for each joint
- Current position – column containing information about the current joint position of the robot
- Set – button that sends user data on the server
- Reset – button that sends zeros for each joint on server
- Continuous mode – sends data to the server every *n* millisecond
- Exit – disconnect server. Close application

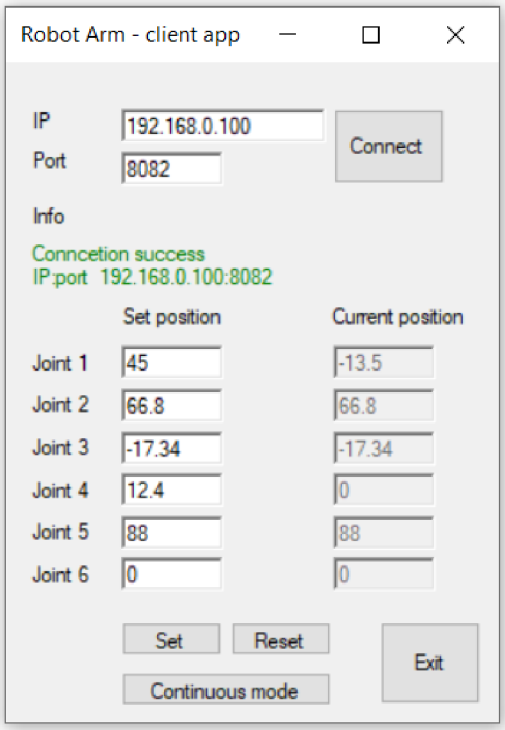


Figure 6-14 C# test application

6.2.1 Headers

```
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Globalization;
using System.Threading.Tasks;
```

Figure 6-15 C# headers

Non-standard libraries:

- *System.Net* and *System.Net.Sockets* are used to declare the client socket and establish connection to the server
- *System.Threading* is used to create stream function, which processes data received from the server in a separate thread
- *System.Globalization* is used to correctly convert string variable to float
- *System.Threading.Tasks* is used to implement delay; the function does not stop the current thread, as in *sleep* method

6.2.2 Socket declaration and connection setup

When the user clicks the *Connect* button, the program reads and parses the IP and port fields. Then a new socket is declared with the following parameters:

- *Addressfamily.InterNetwork* – declare that the connection will be established by using IPv4 address
- *SocketType.Stream* – support two-way byte streams in establish connection mode
- *ProtocolType.Tcp* – specify the network protocol that is used to communicate with server. In this case is used TCP.

```
ip = IPAddress.Parse(ip_box.Text);
port = int.Parse(port_box.Text);

Client = new Socket(AddressFamily.InterNetwork,
             SocketType.Stream, ProtocolType.Tcp);
Client.Connect(ip, port);

thread_ServerRespons = new Thread(delegate() { ServerResponse(); });
thread_ServerRespons.Start();
```

Figure 6-16 C# socket declaration

Thread_ServerRespons - Execute the *ServerRepons()* function in a separate thread. This allows the client application to receive data from the server while processing user data.

6.2.3 Sending data to the server

By clicking on the *Set* button, the data from the fields Joint 1 - Joint 6 are stored in the string variables and passed as parameters to the *SendData* function.

By clicking the *Reset* button, the client sends a string of 6 zeros to the server. This sets the robot to the default position.

SendData function

The function creates a buffer with the same length as the message, then generates the bytecode of our message and writes the code into the buffer. To send a message to the server, use the *Send* method. If the send was not successful, the function will output a message, close the socket, and stop the thread function to receive messages from the server.

```
void SendData(string message)
{
    try
    {
        byte[] buffer = new byte[message.Length];
        buffer = Encoding.UTF8.GetBytes(message);
        Client.Send(buffer);
    }
    catch
    {
        info_box.ForeColor = Color.Red;
        info_box.Text = "Error! Connection lost\nIP:port  " + ip + ":" + port;
        if (thread_ServerRespons != null) thread_ServerRespons.Abort();
        Client.Close();
        LockWindows(false);
    }
}
```

Figure 6-17 C# SendData function

6.2.4 Receiving data from the server

The *ServerResponse* - thread function is used to receive messages from the server.

To receive messages, a 1024-byte buffer has been declared. It is obviously impossible to know how long the bytecode from the server will be. A constant value of 1024 bytes is used, assuming that the server will not be able to go beyond that.

When the byte code is received from the server, it is decoded and stored in a string variable and then displayed on the application screen.

```
byte[] buffer = new byte[1024];
for (int i = 0; i < buffer.Length; i++) { buffer[i] = 0; }
try
{
    for(; ; )
    {
        Client.Receive(buffer);
        string message = Encoding.UTF8.GetString(buffer);
        string[] strLines = message.Split('\n');
        this.Invoke((MethodInvoker)delegate ()
        {
            joint1_c.Clear();
            joint1_c.AppendText(strLines[0]);
            joint2_c.Clear();
            joint2_c.AppendText(strLines[1]);
            joint3_c.Clear();
            joint3_c.AppendText(strLines[2]);
            joint4_c.Clear();
            joint4_c.AppendText(strLines[3]);
            joint5_c.Clear();
            joint5_c.AppendText(strLines[4]);
            joint6_c.Clear();
            joint6_c.AppendText(strLines[5]);
        });
    }
}
catch
{
    if (thread_ServerRespons != null) thread_ServerRespons.Abort();
    Client.Close();
    info_box.ForeColor = Color.Red;
    info_box.Text = "Error! Connection lost\nIP:port  " + ip + ":" + port;
    LockWindows(false);
}
```

Figure 6-18 ServerResponse function

6.2.5 Close client application

By clicking the “Exit” button, the program closes the receive thread function, closes the socket, and closes the application. Before the socket is closed, the program sends an empty message, which is interpreted as disabling the client on the server side.

```
private void exit_button_Click(object sender, EventArgs e)
{
    if (thread_ServerRespons != null) thread_ServerRespons.Abort();
    Client.Close();
    Application.Exit();
}
```

Figure 6-19 Close client application

6.3 Ruby plugin

Ruby plugin performs 4 main functions at once:

1) Providing UI interface, to work with plugin
2) Socket interface
3) Robot movement implementation.
4) Import 3d model.

Before everything it is necessary to puts the «Ruby_robot_plugin» folder on the following path:

C:\ Users\ %username \ AppData\ Roaming\ SketchUp\ SketchUp 2017\ SketchUp\ Plugins

* "AppData" folder is usually hidden, keep that in mind

In SketchUp top menu open "Window" and select "Extention Manager", click on "Install Extension" and go to the « Ruby_robot_plugin » folder, that was placed in the previous step. In this folder find and select the rbz file "su_robot_arm".

If everything is done correctly, in SkecthUp Top menu, in Extension will be new plugin Figure 9-1.

6.3.1 Ruby plugin menu

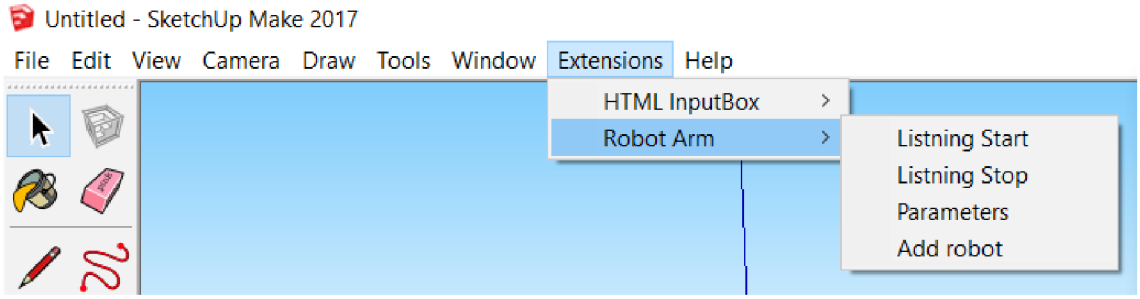


Figure 6-20 Plugin menu in SketchUp

The plugin includes 4 functional buttons:

- *Listening Start* – connect to the server
- *Listening Stop* – disconnect from the server, close socket
- *Parameters* – opens a window where user can change the IP address and port of the server
- *Add robot* – import robot model into a project

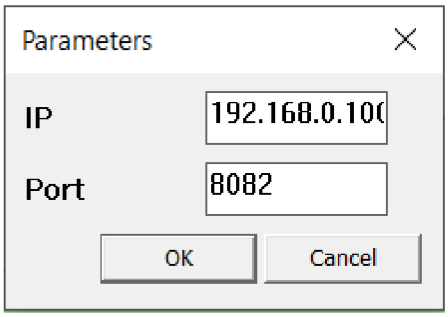


Figure 6-21 SketchUp UI

The UI module is used to create the user interface in SketchUp. The *add_submenu* method adds a submenu to a SketchUp top menu. The next method, *add_item*, adds click items to a submenu. Each item has its own ruby function that starts the process when clicked.

```
unless file_loaded?(__FILE__)
  # Add plugin menu
    menu = UI.menu('Plugins').add_submenu('Robot Arm')

  # Add submenu items
    menu.add_item('Listning Start') { Robot_arm::listning_start() }
    menu.add_item('Listning Stop') { Robot_arm::listning_stop()}
    menu.add_item('Parameters') { Robot_arm::parameters()}
    menu.add_item("Add robot") {Robot_arm::add_robot()}
    file_loaded(__FILE__)
end
```

Figure 6-22 Create submenu in SketchUp

To allow a user to enter his own parameters and change connection settings uses the *inputbox* method, which is also part of the UI module.

```
def  Robot_arm::parameters()
    prompts = ["IP", "Port"]
    defaults = [$ip,$port]
    input = UI.inputbox(prompts, defaults, "Parameters")
    $ip = input[0].to_s
    $port = input[1].to_i
    listning_start()
end
```

Figure 6-23 Create input window in SketchUp

6.3.2 Ruby socket interface

To use the socket interface, it is necessary to include the socket library:
require 'socket'

```
def Robot_arm::listning_start()
    @buffer = ""
    @ruby_socket = TCPSocket.new($ip,$port)
    @ruby_socket.puts 'Sketchup socket connected'
    puts "Ruby socket start"
    @stimer = UI.start_timer(@freq,true) {
      begin
          @ruby_socket.read_nonblock(500, @buffer)
          Robot_arm::do_transform()
      rescue
    end
    }
end
```

Figure 6-24 "listening_start" function

The function *listening_start* uses a socket to connect to the server and receive messages from it. The function works as follows: A new socket is declared using the TCPSocket module and the "*new*" method, where the input parameters are the IP address and the port of the server. *$ip* and *$port* are global variables of type string, which store parameters about the server. The test message is sent to the server using the "*puts*" method and the socket descriptor (@ruby_socket). This message is used to inform the server that SketchUp has connected to it. Using the timer and the "*read_nonblock*" method, the received message is written to the *@buffer* string variable every *@freq* seconds. The first parameter of the *read_nonblock* function is the size of the message, and the second is a variable to write the received message into. After receiving a message from the server, the *do_transform* function is executed, which is responsible for changing the position of the robot.

The *listening_stop* function stops the timer started in the *listening_start* function and closes the ruby socket.

```
def Robot_arm::listning_stop()
    UI.stop_timer(@stimer)
    @ruby_socket.close
    puts "Ruby socket close"
end
```

Figure 6-25 "listening_stop" function

6.3.3 Robot movement implementation

Of an entire rb file, only the *joint_rotation* function interacts with the robot model in SketchUp. The function has 2 input parameters:

- *index* - a number from 1 to 6, the index indicates the joint with which the function now interacts.
- *rotation_axis* - vector around which the rotation takes place.

```
def Robot_arm::joint_rotation(index, rotation_axis)
  # Part 1. Find and select component by name
      mod = Sketchup.active_model
      ent = mod.entities
      mod.selection.add(Sketchup.active_model.definitions["Joint #{index}"].instances)
      joint = mod.selection[0]
      tr = joint.transformation
      point = tr.origin

  # Part 2. Step is a difference between the previous and the next angle received from the client
      step = @buffer.lines[index-1].to_f - $j_last[index-1].to_f
      $j_last[index-1] = @buffer.lines[index-1]

  # Part 3. Do transformation
      joint_transform = Geom::Transformation.rotation point, rotation_axis, step.degrees
      joint.transform! joint_transform

  # Part 4. Clear selection
      mod.selection.clear
end
```

Figure 6-26 "joint_rotation" function

Part 1.

- connect to the project in SketchUp
- select the component by its name using the index
- extract the pivot point of this component

Part 2.

- calculate the angle of rotation for the component. The angle of rotation is defined as:

$$step = new\ angel - previous\ angle$$

- The sign indicates the direction in which the turnaround is going.
- after the calculation, the new position is rewritten to the previous one.

$$previous\ angel = new\ angel$$

Part 3.

- Rotation of the selected component. The "*Rotation*" method is part of the "*Geom*" module and has three parameters: The component's rotation point, the vector around which the rotation is performed, and the angle.

Part 4.

- clear target

The *do_transform* function calls the *joint_rotation* function one after the other for each of the joints.

```
def Robot_arm::do_transform()
    Robot_arm::joint_rotation(1, [0,0,1])
    Robot_arm::joint_rotation(2, [0,1,0])
    Robot_arm::joint_rotation(3, [0,1,0])
    Robot_arm::joint_rotation(4, [1,0,0])
    Robot_arm::joint_rotation(5, [0,1,0])
    Robot_arm::joint_rotation(6, [0,0,1])
end
```

Figure 6-27 “do_transform” function

6.3.4 Robot import

The robot model is located in the *Plugins* folder, which we placed in the SketchUp program folder at the beginning. The path to the Skp file is created and stored in the variable *path_skp*. The model should be imported into the project using the "*load_from_url*" method, but the model is still not available because the location of the model was not specified in the SketchUp layout. To do this, we call the "*add_instance*" method with the following parameters: the model we want to place, the position where the model should be placed.

```
def Robot_arm::add_robot()
  $j_last = [0,0,0,0,0,0]
  path_skp = Sketchup.find_support_file('Plugins') + '/su_robot_arm/robot_KUKA_KRC6_v3.skp'
  mod = Sketchup::active_model
    begin
        load_robot = mod.definitions.load_from_url(path_skp)
    rescue
    end
    point = Geom::Point3d::new( 0, 0, 0 )
    add_robot = mod.active_entities.add_instance(load_robot, Geom::Transformation::new(point))
end
```

Figure 6-28 "Add_robot" function

7. TESTS AND RESULTS

7.1 Server

The server is a console application that informs the user about the clients that are currently connected to it and about the data transfer. The server application's first message is the local IP address and the port on which it is working.

Neither the address nor the port has been selected by the application, its value has been manually set for network use. To work in another network and on another device, the file main.cpp must be accessed to change it manually again. To find out your IPv4 address, use the /ipconfig command in the Window command line.

If the client connects successfully, the server writes out the parameters of the connected device and assigns it an ID that is also the client's socket descriptor.

When a new socket is connected, its parameters are written and then extracted from the SOCKADDR_IN structure. The structure contains the next information:

- Name of the host in the network
- Local IPv4 address
- The port to which the client is connected
- Client ID - the socket handle of the client

The server collects sockets from connected clients in a "sockets array" that allows multiple clients to connect and transfer data through the server at the same time, which is not quite correct from the "one client - one robot" point of view, but the task was to write a standalone server application that could support multiple clients at the same time, accept connection requests and easily disconnect.

The server does not perform any data processing. When a message is received, the server outputs it to the console and sends it to other clients (one of which is the Ruby client).

The server also recognizes and informs the user if the connected client is a Ruby client.

When the server receives a message from the client, it outputs the client ID and the angles for the robot in the order Joint 1 - Joint 6.

If the client is disconnected, the console displays the appropriate message.

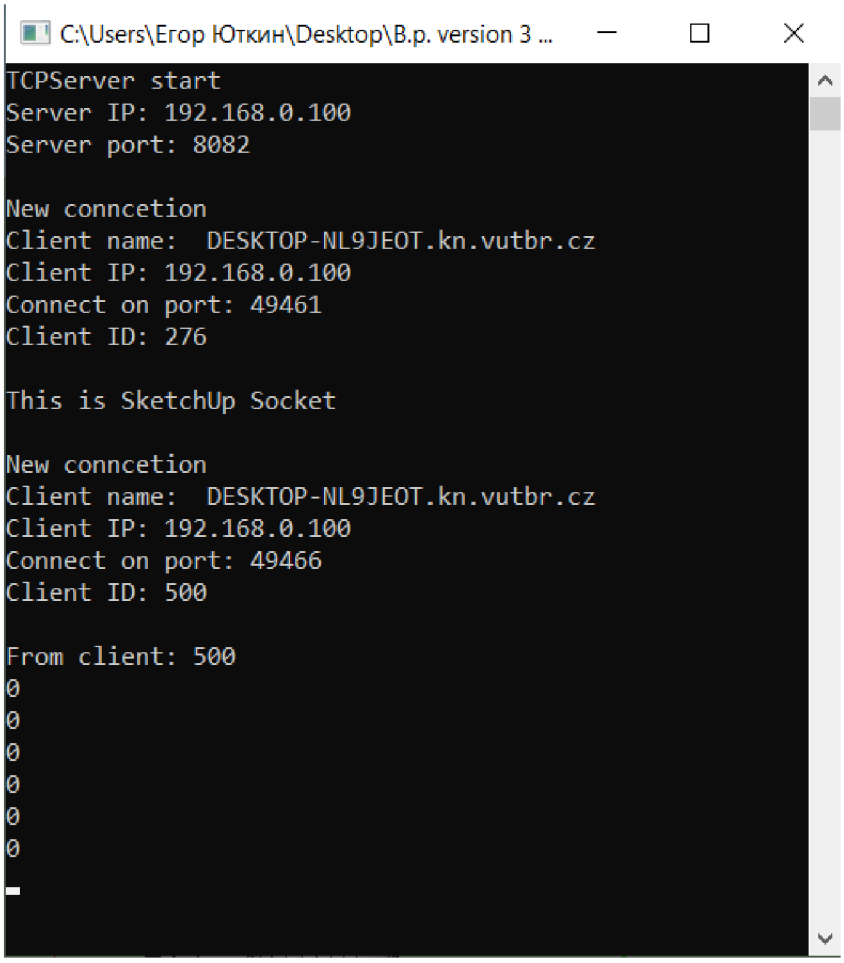


Figure 7-1 Server console application

7.2 C# Test client application

The interaction with the virtual robot was implemented via the user interface (Figure 6-13) of the C# client application, which will serve as an imitation of a real robot.

Due to the current situation it was not possible to work with a real robot.

The client application has an interface to connect to the server, a window to display information, input fields for angles and output fields with the current position of the robot. The client application receives the current position from the server if the position of the robot was changed by another client.

"Manual test mode" - the user can set the angles for each joint separately. Information from the fields Joint 1-Joint 6 is written in a one byte code and sent to the server by pressing the "Set" button.

"Continuous Mode" - implemented to test the ability of the server and Ruby client to process data and change the robot position in real time. The angles in the fields Joint 1 - Joint 6 are divided into 10 smaller steps for each joint and sent sequentially to the server at a frequency of 200 milliseconds. This client-server

interaction mode creates a situation in which data is sent to the server as a real-time data stream, thereby issuing a connection to a real robot. The smoothness of movement of the virtual robot in "Continuous mode" depends primarily on the computer and network on which the client and server are located. We can observe the intermittent movement of the joints, but it gives a clear idea of the sequence of movements of each joint.

The client "Info" window displays the following information:

- Successful connection to the server
- Connection to server failed
- Connection lost
- Error in "Continuous mode".

Figure 7-2 «info» message. C# client

7.3 Ruby client

Ruby client (Ruby plugin) functionality:

First of all it contains a script to perform an action with a robot inside a SketchUp. Two functions are responsible for implementing the movement of the robot: the functions *joint_rotation* and *do_transform*.

Second, the Ruby plugin contains a socket interface and executes the function of a TCP client. The client does not send data about its position to the server, but only receives data with new angles from the server. Only during the first connection does the Ruby client send messages to the server so that the server can identify it as a Ruby client among others. The Ruby client also has a user interface for connecting to and disconnecting from the server and changing connection parameters (Figure 6-18 and Figure 6-19).

The final function of the Ruby client is to import the robot into the SketchUp scene.

8.CONCLUSION

The implemented simulation tool has a number of advantages and disadvantages, which will be discussed in this chapter.

In this work, the robot motion was achieved by using the hierarchical impact model, where the lower joint in the hierarchy was connected to the upper one and bound to its pivot point (for more information see chapter 5-2). This realization of the robot motion implementation is the optimal and probably the only solution to the task in the Free SketchUp environment.

The Ruby plugin does not use the thread functions that could be used to implement the processing of the data flow from the server in a separate thread. Unfortunately, despite the fact that "clean" Ruby supports the thread functions, the thread functions did not work inside SketchUp. As a result, receiving data from the server was implemented rather poorly. The data receiving function was implemented using a timer, which causes delays in data processing and makes the program more complicated. This solution is not optimal.

The implementation of the server side is the most successful. By using thread functions and a socket array a good support of several clients at once was achieved. The disadvantages of the server side: it cannot determine the IP address of the computer in the network and cannot choose the free port. Entering these parameters manually reduces the speed and comfort of interaction with the application.

The good advantage of C# Test Client App is the simple interface for the connection to the server and robot control; the implementation of an information window; the reception and processing of data from the server in the thread function.

The disadvantage is a weak implementation of "continuous mode"; the use of invalid values in fields can cause an exception (double dots or double commas). Which is not a big deal, but still not fixed.

A general disadvantage of clients and server is that there is no dynamic memory allocation of memory for packets when the server receives messages from clients or backwards. The only solution that was considered was to send 2 packets, one with the message size, the other the message itself, which reduces the transfer speed and was therefore not implemented.

In summary, the application implements the initial task - creating a simple simulation tool to visualize a stationary robot in Sketchup. Below are the main possible ideas for improving this tool.

Create a universal solution for importing different robots that is easy to control; modify the Ruby plugin to control multiple robots in a scene simultaneously; create a client that understands the robot's programming languages (such as KRL) and interprets the code for the Ruby plugin; add an interface for connecting to a programmable logic controller. Finally, this technology

can lead to the development of a custom platform for creating digital factories based on Free SketchUp.

Literature

[1] SketchUp Make vs Pro vs Free vs Shop vs Studio [cit. 2019-11-14]. Available at: https://mastersketchup.com/sketchup-make-pro-free-shop-studio/

[2] SketchUp Ruby API Documentation [cit. 2020-04-28]. Available at: https://ruby.sketchup.com/_index.html

[3] Ruby Code Editor [cit. 2019-10-22]. Available at: https://alexschreyer.net/projects/sketchup-ruby-code-editor/

[4] Automatic SketchUp „*Creation 3-D Models in Ruby*" 2010, author: Matthew Scarpino [cit. 2019-11-08]. Available at: https://forums.sketchup.com/t/automatic-sketchup-sketchup-scripting-extension-and-ruby-help/14939

[5] TCP/IP protocol suite [cit. 2020-04-05]. Available at: https://docs.oracle.com/cd/E23823_01/html/816-4554/ipov-6.html

[6] Application layers protocols suite [cit. 2020-04-08]. Available at: https://en.wikipedia.org/wiki/Category:Application_layer_protocols

[7] IPv4 and IPv6 addresses [cit. 2020-03-12]. Available at: https://bezopasnik.info/протоколы-ipv4-и-ipv6-в-чем-разница-и-что-лучше/

[8] Routing [cit. 2020-04-20]. Available at: https://geek-university.com/ccna/ip-routing-explained/

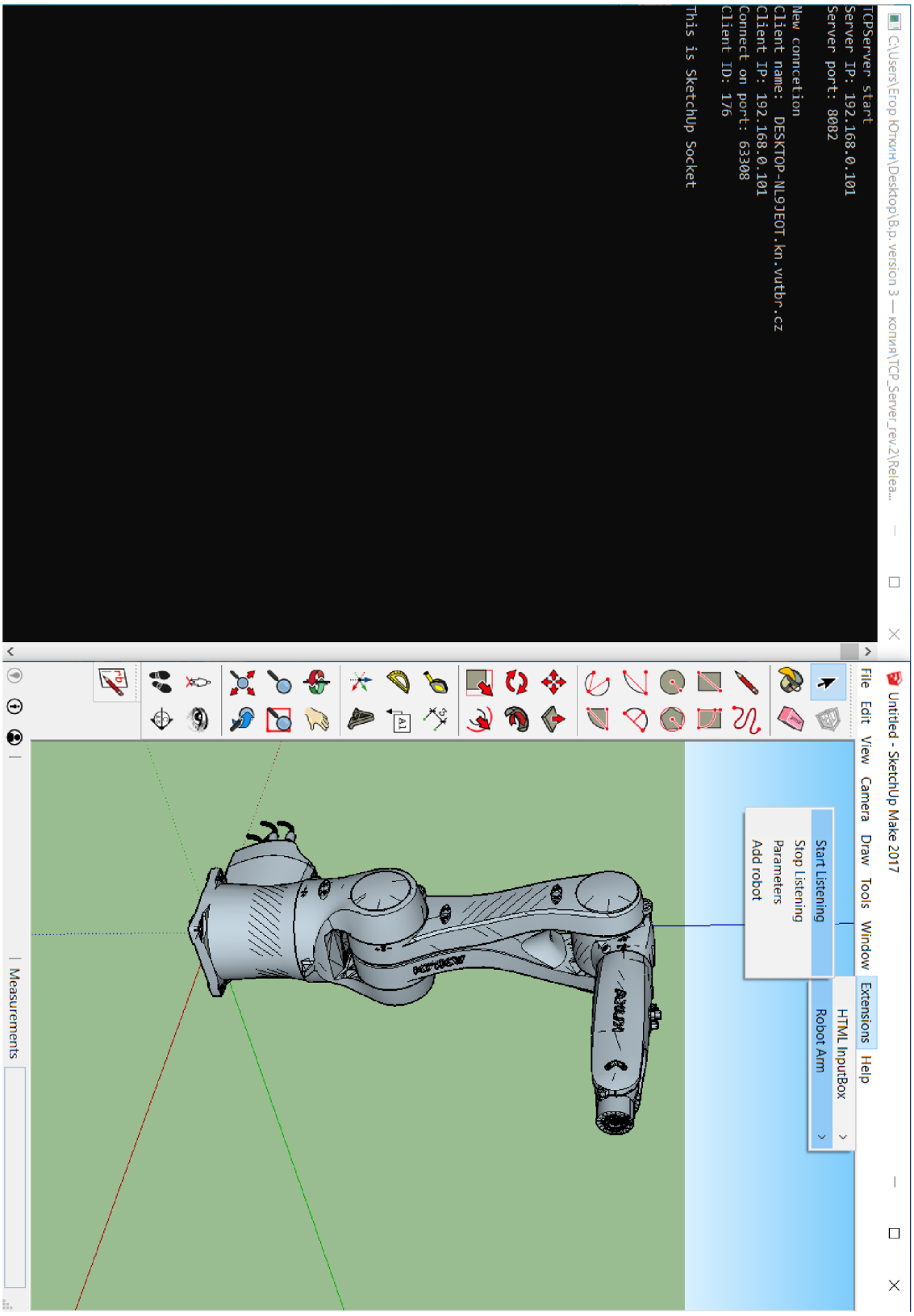
[9] Reserved IP addresses [cit. 2019-11-08]. Available at: https://ru.qwe.wiki/wiki/Reserved_IP_addresses

[10] What is a Network Socket? [cit. 2020-03-12]. Available at: https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

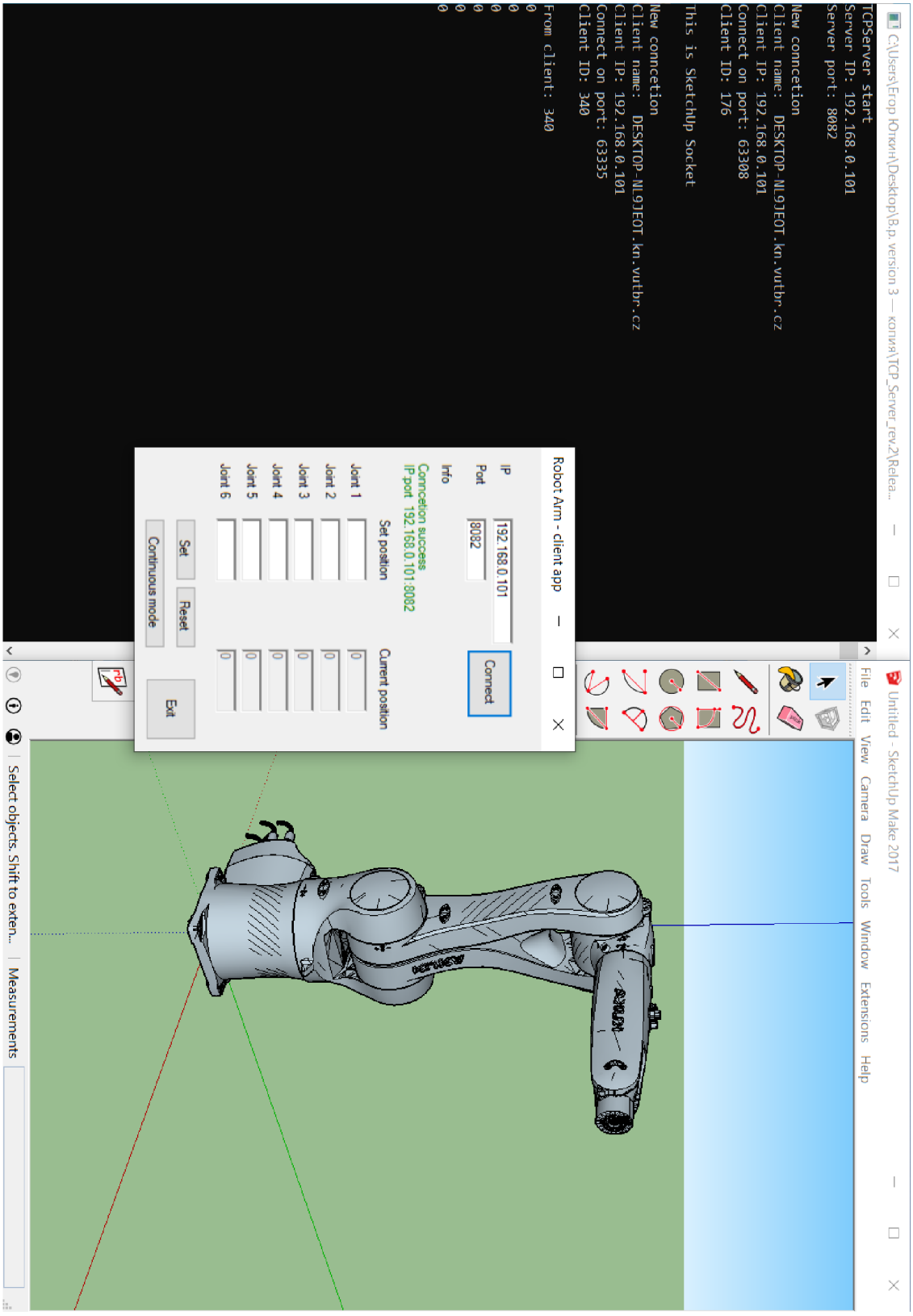
[11] KUKA KR6 manual [cit. 2020-03-24]. Available at: http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20KR%206%2010_AGILUS.pdf

List of appendices

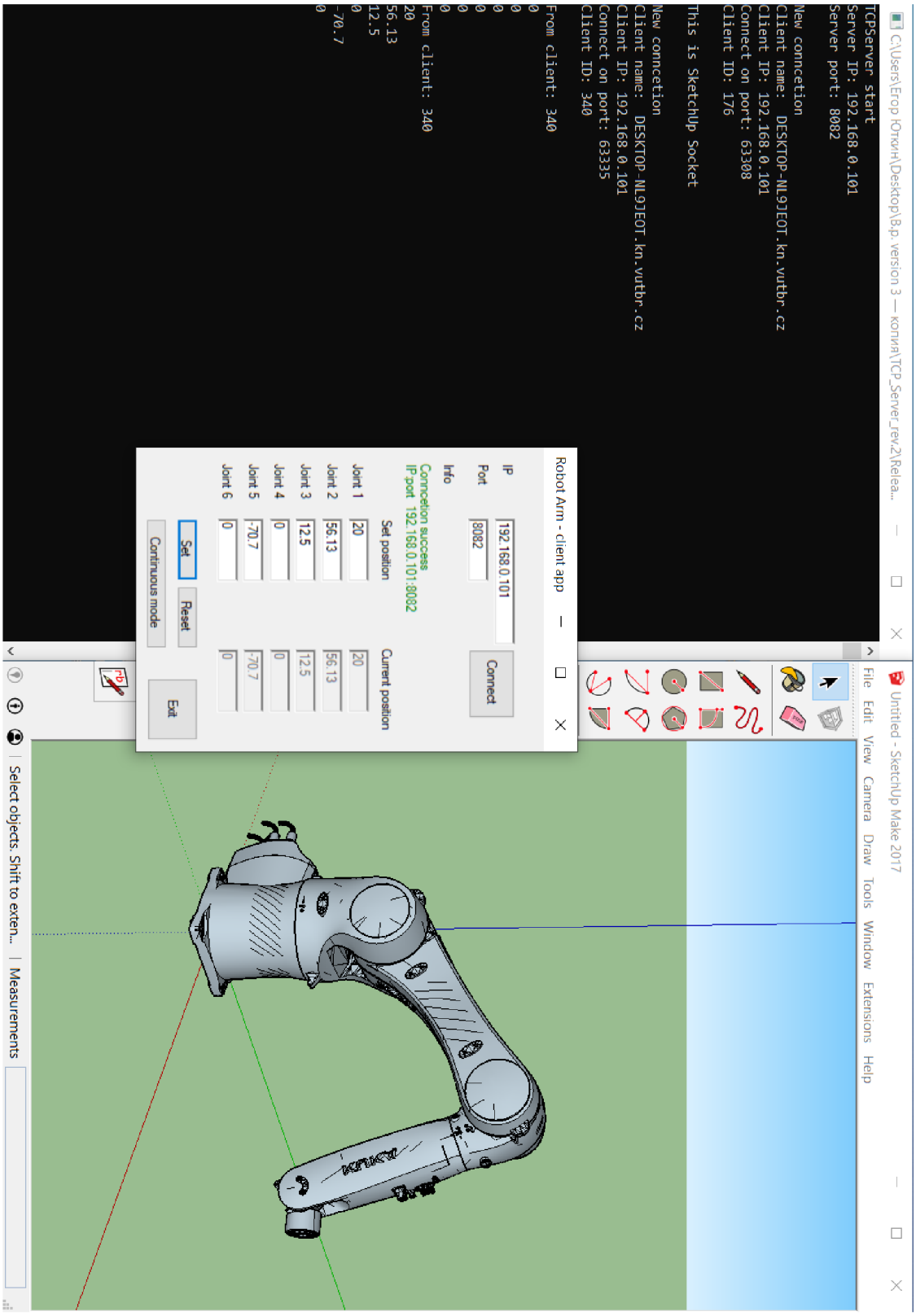
Appendix 1 – The Ruby client connects to the server



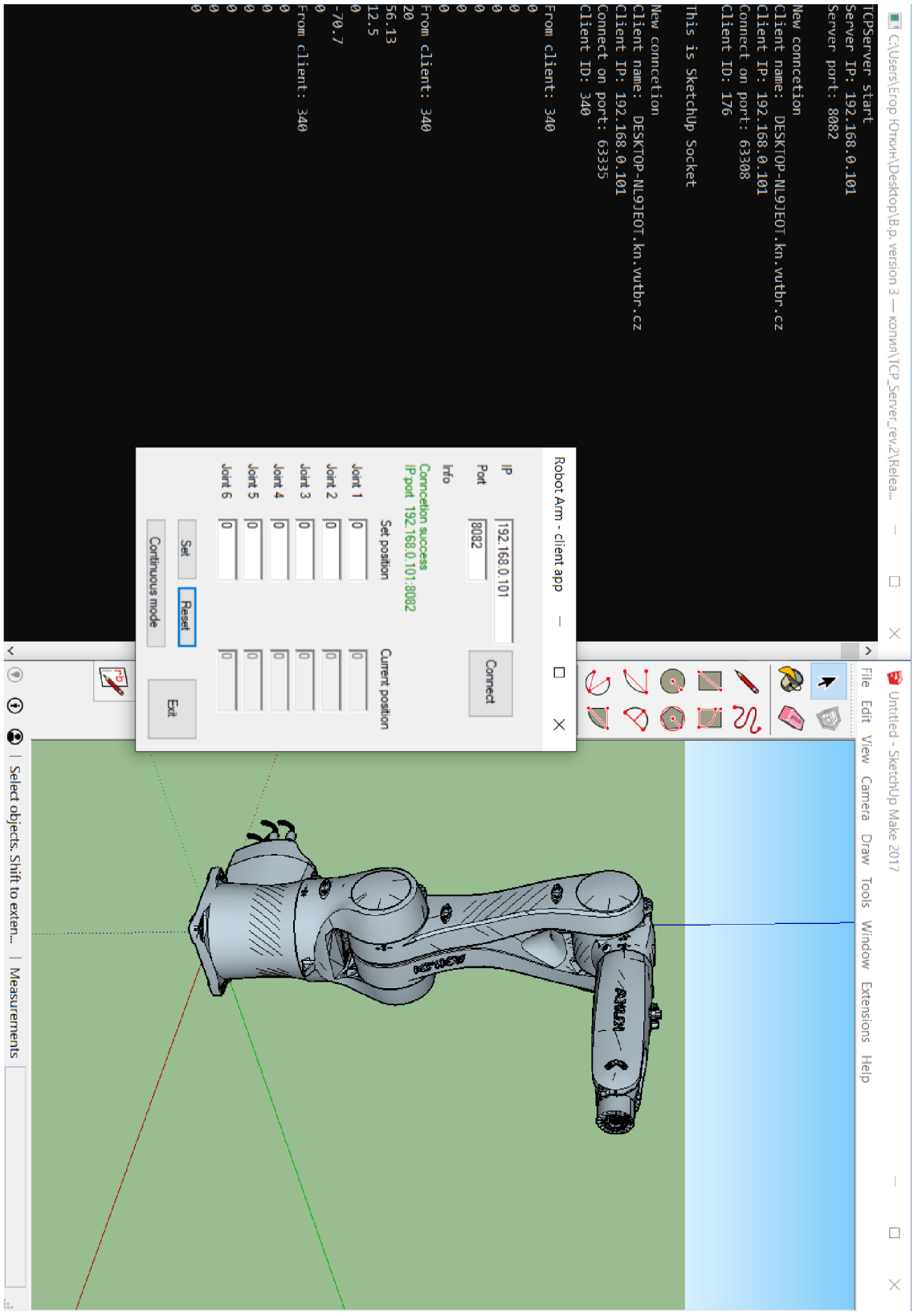
Appendix 2 – The C# client connects to the server



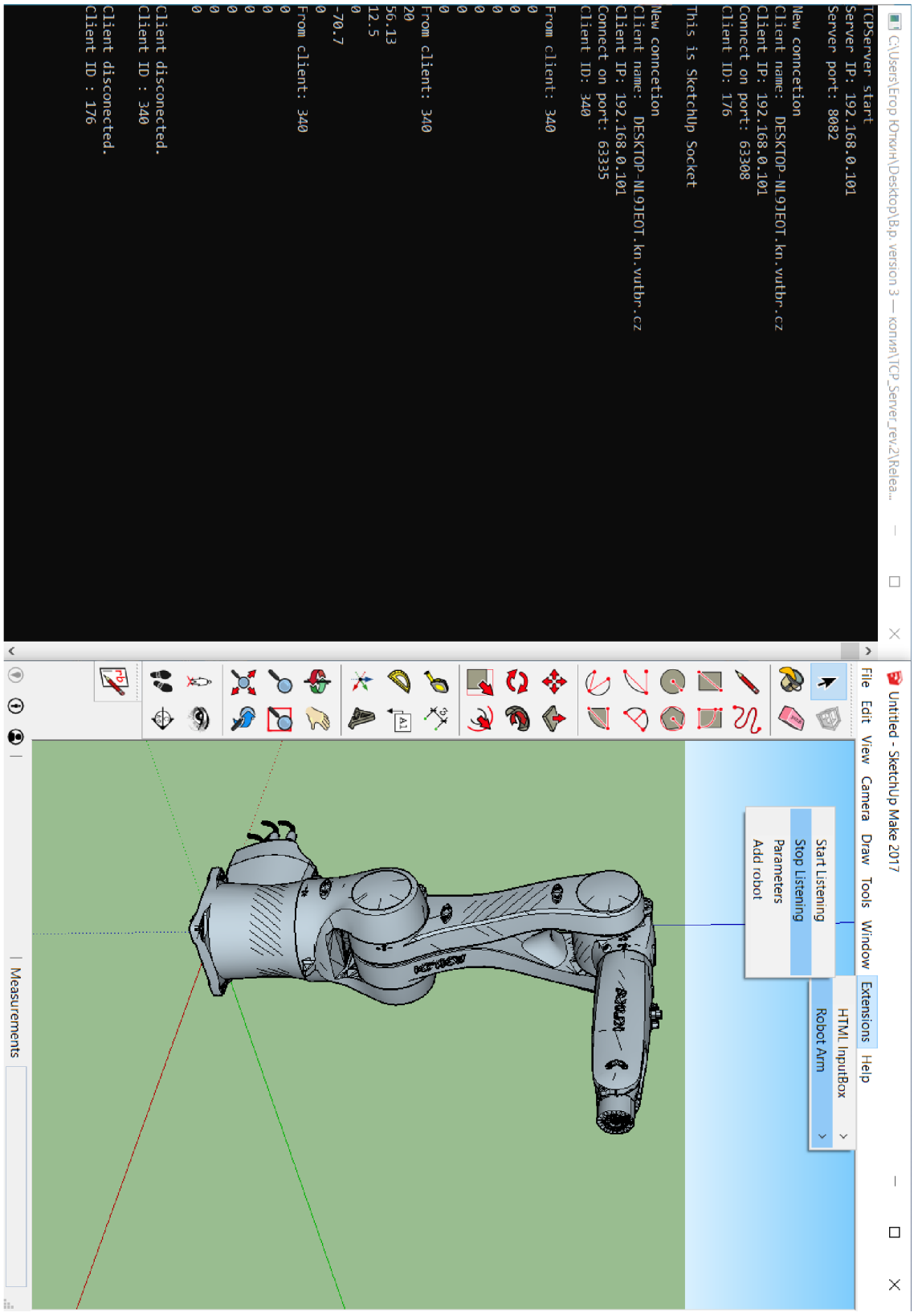
Appendix 3 – Set a new position



Appendix 4 – Reset the position



Appendix 5 – The clients are disconnected



Appendix 6 - CD with simulation tool

The compact disc contains the following data:

- “TCP_Server” - Visual Studio Project folder with C++ server. Do not forget to change an IP address and a port before debugging, otherwise an exception will occur.
- “Test_Client” - Visual Studio Project folder with C# client.
- “Ruby_robot_plugin” - folder with 3D model of the robot and SketchUp plugin. Do not forget to put this folder in the folder with SketchUp Extensions, see chapter 6-3.