PD.TVZ.HR
POLYTECHNIC & DESIGN Vol. 9, No. 4, 2021.
Mrežna inačica: ISSN 2459-6302
Tiskana inačica: ISSN 1849-1995
DOI: 10.19279/TVZ.PD.2021-9-4-19

COMPUTER SIMULATION OF ROBOTIC ARM OPERATION

Petar Mamić, Vesna Alić-Kostešić, Goran Sirovatka, Vlatko Mićković

Zagreb University of Applied Sciences/Mechanical Engineering Department

ABSTRACT

The robotic arm operations can be simulated using a computer. This work is the result of a study at Zagreb University of Sciences, Mechatronics Study program and shows how to simplify complicated functions and the importance of programming rules, in developing one's own robot programming language, as well as the user interface for it. Computer environment (interface and programming language) was developed for a simulation of a robotic arm using Unity Engine and C # for programming. The interface is designed to be simple, so each one with the basic level of knowledge about robotics can use it for educational purposes or just for the sake of experimentation. This paper will explain the prerequisites and development done in reaching the goal. Expiation of methods used, and scripts written are given. The goal achieved is creation of the free tools that can simulate a robotic arm to the point of being useful, intuitive, and educational. We will also state the improvements that we plan to do (multiple selection, rotation and scaling tool).

Keywords: *robotic arm, 3D simulation, C#, Unity, programming*

1. INTRODUCTION

Robotics develops and produces robots that serve to replace jobs that would otherwise be done by humans. This work was based on P.Mamić Bachelor thesis successfully defended in 2020.[1]

Control of the robot itself and how robot behaves in the working environment is an important aspect of robotics and such a human operator is needed to make sure that the robot does exactly what it is supposed to do.[2][3].

Challenges begin when each robot has its own programming language and its own control scheme that are different from other robot equipment. Understanding these controls and programming languages of the robot arm makes the robot more useful and efficient. Such knowledge [4] can and are acquired by practice, but it is difficult to do the same in an educational environment.
Programming a robotic arm in an educational environment must start from entry level and gradually rise. In education understanding to control the robotic arm means to understand and be able to use the knowledge from mathematics, programming, electrical, and mechanical engineering.

The solution for this problem we find is developing an educational tool that can enable students to learn about robots and their limitations. While such programs exist for robotic arms that are in production, they are expecting some prior knowledge before one can use them, which would be a very frightening first experience.

In this work we would like to show how we can simplify a robot arm simulation program with its own simple programming language, and describe how we have simplified complicated functions, implemented the simple programming language for the simulated robotic arm and how the interface for students would look like.

2. DEVELOPMENT OF SIMULATION ENVIRONMENT

2.1. GENERAL

We will describe step by step process of development of our robotic arm simulation program named RoboSim and will present the whole project sharing the github link.

We used Unity[7], C# and Blender as tools , and will included a 3D view of a model robotic arms with some basic physics which is part of RoboSim.

2.2. DEVELOPMENT PLAN

We choose the project planning software Trello to track the progress of project. Steps in our project were defined in following order:

- Camera control mode
- 3D model of the robotic arm and objects to be used.
- Robotic arm movement method
- Programming language for the robotic arm
- Adding functionality to individual objects
- Ability to change the work environment.

These main steps were divided into sub-steps.

We started with Unity project with the development our RoboSim work environment.

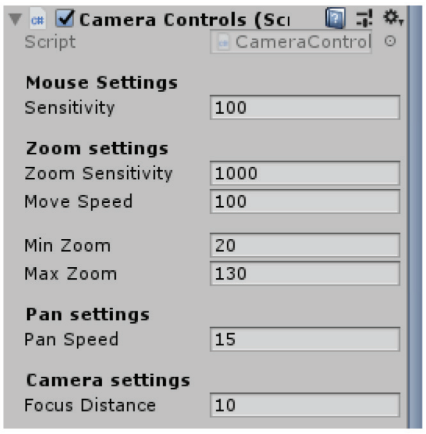
2.3. CAMERA CONTROL MODE

The first step is to develop control the orientation camera in the work environment. Student must have an intuitive way of controlling the camera for the best possible orientation in the environment space. The way we set up camera controls will affect how fast a student can work with the program. The more intuitive way of controlling the camera the faster the student can adapt and the sooner he can control it with ease.

For functions such as zooming, moving, and rotating the camera we used the standard inputs that are found throughout various programs, mouse wheel to zoom in and out, right click to rotate the camera and middle mouse button to pan the camera.

In our environment camera can:

- Rotate
- Focus
- Pan
- Move
- Zoom



```
Camera Controls (Sc
Script               CameraControl

Mouse Settings
Sensitivity          100

Zoom settings
Zoom Sensitivity     1000
Move Speed           100

Min Zoom             20
Max Zoom             130

Pan settings
Pan Speed            15

Camera settings
Focus Distance       10
```

Figure 1 *All variables with a public declaration are presented in the inspector*

While this is a simple task there have been times where camera controls are different if one will use other or commercial programming software, but we have chosen this model to make learning curve steeper.

2.4. ROBOTIC ARM MOVEMENT METHOD

To simulate a robotic arm a model of a robotic arm needs to be made alongside with some objects that could fill the scene to create a more living working environment. These objects include a treadmill, a proximity sensor, a distance sensor, simple cube for robot manipulation and a move tool. These objects open possibility of different working environments when implemented together in a scene.

These objects have been chosen due to their simplicity as well as their common functionalities in the real world.

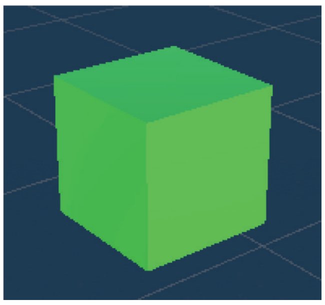


Figure 2 *Green cube*

The green cube is the simplest form, but unlike some objects, it will be able to change in size and way it is used.

The green cube can be either a physical object which means that physics will act on it and have additional customization options like weight changes, or it can be a kinematic object which means that physics does not act on it and acts like a wall that cannot be moved.

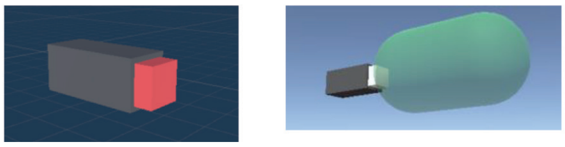


Figure 3 *Distance sensor and presence sensor*

Two sensors are added to the project: distance sensor and a presence sensor. The sensors communicate with the robotic arm. Distance sensor displays the distance from the sensor onwards as close as possible to it while the presence sensor shows if anything is in its presence space. This data can be used by the user to better control the robotic arm through programming.

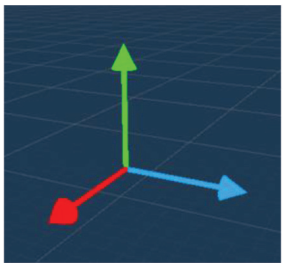


Figure 4 *Moving tool*

Programs when manipulating objects use some kind of visual tool showing process of manipulation of the objects. In our environment we have a moving tool that shows x, y and z coordinates, so student can use it to move objects in a simple way. In this model, a blender was used for the tips, ie arrows, since Unity does not have pointed models, a shader is also added so that the moving tool can be seen through all objects.

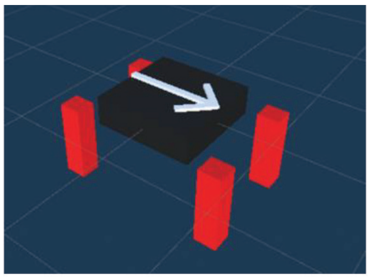


Figure 5 *Treadmill*

Robotic arm works in production and especially on treadmill so it makes sense that a treadmill model exists in the environment so student can come up with their own production model.

Although the model doesn't seem to be connected, there is a script that keeps the treadmill connected to the legs when the program is started and allows the legs to be in the same positions when resizing.

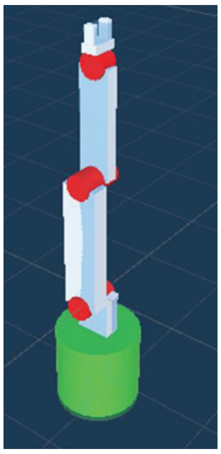


Figure 5 *Robotic arm*

Robotic arm is modeled in the simplest but most practical way, the joints marked with color correspond to the axis along which the joint moves (red - x axis, green - y axis). The robotic arm will use inverse kinematics to be able to move when the user type in commands via the programming language console. The model also has a copy of itself in the same position called "SimRobotArm" which is a mathematical representation of the robotic arm as it calculated to reach a certain point.

2.5. ROBOTIC ARM MOVEMENT METHOD

Robotic arms have multiple ways of movement. And are connected in the following way of hierarchical models that consist of:

• Joint - dimensions are not considered and only the possibility of joint rotation is considered.

• Segment - a rigid element that is connected to the joints.

Combining forward and inverse kinematics in most situations satisfy the necessary accuracy for a given task. To simulate this, we used geometrical inverse kinematics [5][6].

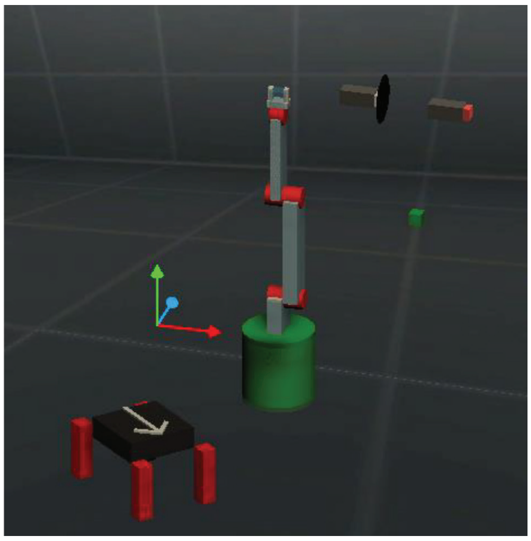


Figure 6 *All object used in RoboSim*

We will have to control only one part of the robotic arm, the end effector, while letting the rest of the joints do the necessary work to achieve the desired position. Geometrical inverse kinematics is just a type of way to get the result of a inverse kinematics problem. Using geometrics in Unity alongside with C# we can force robot segments to be always connected while following some basic rules and principles such as limitations on joints.

The simple way we achieve this is that the end of the segment gets separated, rotated, and positioned to "catch" the end effector while the rest of the segments do the same for the segment in front of it, with the final segment connected. We then reposition them back from the starting segment to the end and get a position where the robot arm is closest to its end effector repeating the process a couple of more times gets us to the desired position.

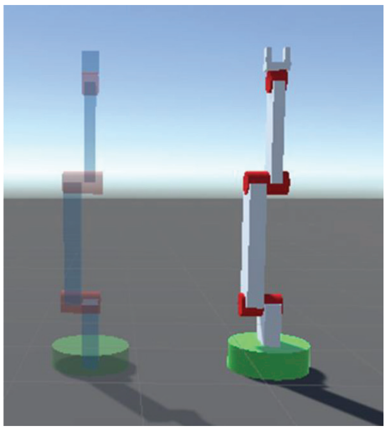


Figure *7 Simulation model left and real model right.*

Student control the robot using a special moving tool called "Hand Tool" which acts as an end point for the robotic arm, moving the end point also moves parts of the robotic arm to position the gripper on the end point. The endpoint is activated by pressing the robot arm tool.

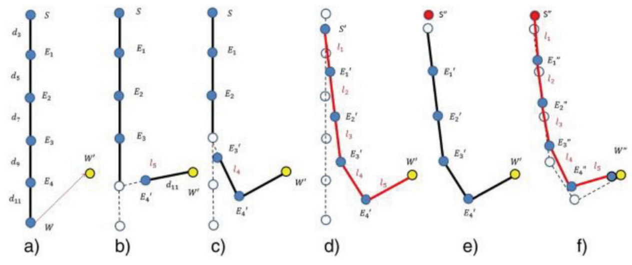


Figure 8 *Method for inverse kinematics*

2.6. PROGRAMMING LANGUAGE FOR THE ROBOTIC ARM

For the user to be able to write program code to manipulate robot arm, a programming window is added called a console with two buttons bellow. The left button is used to transfer the code to the robotic arm while the right one runs the code, and the robotic arm executes the code.

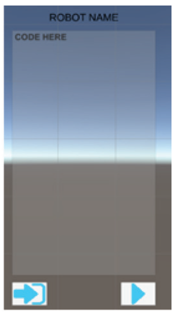


Figure 9 *Programming console*

The programming language is a combination of C code and CNC code using "Commands" to control the robot directly and "Statements" to control the behavior of the robot. Such examples of Command code are MPN which translates to Move to Position N or G and R command which tells the robot to Grab or Release his hand grippers. For "Statement" code there is a classical IF statement and a type of for loop called REPEAT which will repeat a block of code a certain number of times, combining the two can result in a classical for loop.

Command of our programming language are:

• MOVE TO POINT - sets the end point to the robotic arm to reach, the command is invoked from the MPN where N is the position number or variable.

• WAIT - stops at a specified time and is called with WNS or WNMS where N is a number while S or MS indicates the time of a second or millisecond.

• SPEED - determines the speed of the robot and is invoked with SN where N is the speed of the robot (default 30).

• GRAB AND RELEASE - with the commands G and R opens and closes the gripper.

• REPEAT - A type for loop where everything within a REPEAT block is repeated a certain number of times, first set REPEAT (N) where N is the number repeated and close with REND.

• IF - classic if block where it is checked whether the conditions within the if block are correct or not, it is invoked with IF (CONDITION) where the condition can compare numbers or variables with <,>, = i! characters or check the sensors eg if the distance is below 5 or if the presence sensor is active, it closes with an IEND.

• LOOP - is an unconditional while loop, repeats everything inside the loop indefinitely until it encounters a BREAK, invokes with LOOP and closes with LEND.

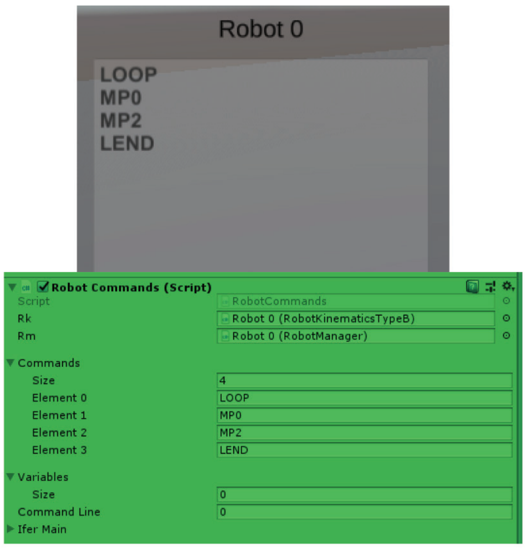
• INT - creates a variable so that it serves as a declaration, is invoked with INT and the name of the variable and what value it contains is placed in front.

• MATHEMATICAL OPERATIONS - the language can be used with +, -, * and / to calculate variables but only two numbers.

For more information reference the github repository.

2.7. ADDING FUNCTIONALITY TO INDIVIDUAL OBJECTS.

We would like to that our modeled objects have desired effect on the environment around them.


Robot 0

```
LOOP
MP0
MP2
LEND
```

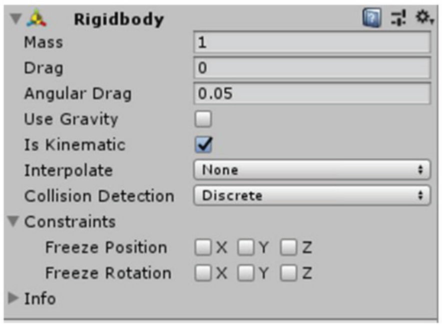
Robot Commands (Script)

Script	RobotCommands
Rk	Robot 0 (RobotKinematicsTypeB)
Rm	Robot 0 (RobotManager)
Commands	
Size	4
Element 0	LOOP
Element 1	MP0
Element 2	MP2
Element 3	LEND
Variables	
Size	0
Command Line	0
Ifer Main	

Figure 10 *Display transfer of code to the robot*

So, they need to have certain functions making them "real". Each object has been given a script for their behavior and how it will intervene with rest of the process (program).

The green cube does not get any special script but instead gets a component that Unity already owns called *Rigidbody.*


Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	✓
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z
Info	

Figure 11 *Rigidbody component*

The component enables physics on the object, i.e., gravity and the possibility of collisions. *Rigidbody* has several functionalities that change the way an object will react:

• Mass - adds mass to the object and the higher the mass the heavier the object

• Drag - friction in the air that slows down the object, the higher the friction the greater the resistance.

• Angular Drag - friction at an angle

• Use Gravity - activates / deactivates the gravitational force on the object

• Is Kinematic - activates / deactivates the action of Rigidbody

• Interpolate - allows the forces acting on the object to be equalized so that there is not too much squeaking when forces are applied to the object.

• Collision Detection - changes the speed at which the collision will be updated.

• Constraints - you can select ordinates and axes along which the object cannot be rotated or moved.

• Info - keeps the data related to the object

Proximity and distance sensors have been given the ability to detect and measure the environment around them as well as to be referenced in the programing code for certain action such as TRUE and FALSE statements while the move tool has been configured to move other objects around the scene in a more intuitive way. For reference see the git hub repository.

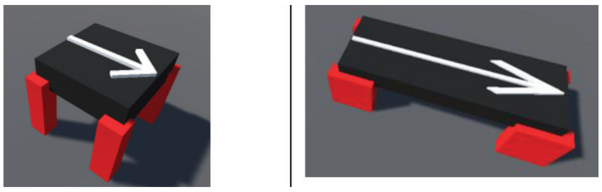
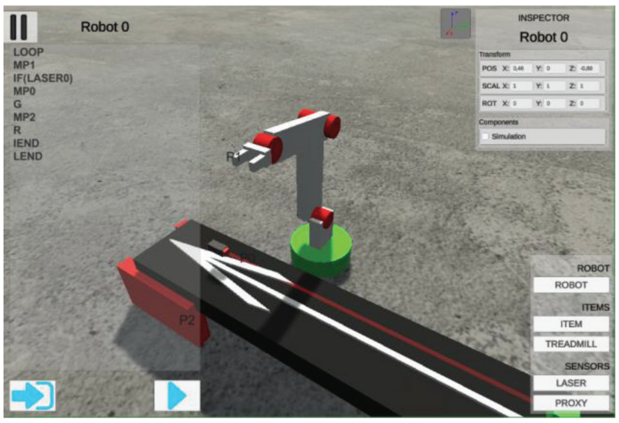


Figure 12 *Resize the trade mill while keeping the legs in the same positions.*

To keep the treadmill legs in positions all four legs are positioned at angles using the length and width of the treadmill. We programmed a function that uses collision objects to check if anything is on the treadmill, and direction and speed of the treadmill is added.


Robot 0

```
LOOP
MP1
IF(LASER0)
MP0
G
MP2
R
IEND
LEND
```

Figure 13 *Robot arm in environment, with code console and program, inspector and description of sensors used*

2.8. ABILITY TO CHANGE THE WORK ENVIRONMENT

With everything constructed it is time to let the student can create and manipulate its own working environment, in order to achieve that a Spawn window and Inspector window have been added. Spawn window lets the student spawn any object into the screen while the Inspector window shows the information of an object as well as the ability to change them or edit them. The student can now edit and change the environment to its own liking and experiment in real world examples.

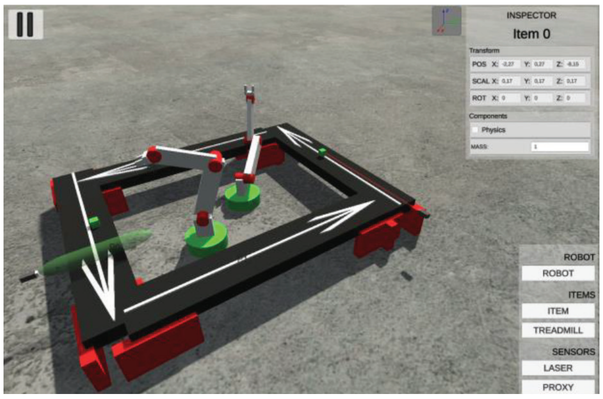


Figure 14 *Robotic arm in action*

Student can develop their workspace so it will have a tool that allows them to create the objects listed in the workspace, so a creation bar will be used. The bar contains "SENSORS" which contains a distance sensor and a presence sensor, "ROBOT" which only creates one robotic arm and "ITEMS" which contains a working bar and a green cube.

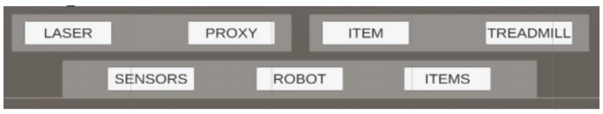


Figure 15 *Object creation bar with and without additional windows.*

3. CONCLUSION

3.1. DISCUSSION

We have shown that it is possible to create a simulation of a robotic arm with free tools to the point of being useful, intuitive, and educationally beneficial. Benefits we can see are in simplicity in design and basic programming language which associate to the language used in professional robot machines. Nevertheless, this simulator could be very useful when one has to teach elements of robotics in remote classroom where school robot or industrial one are not accessible. Students can work on this simulator in their home. Having open source of the code give opportunity for students exercise in improving their skills and developing new functionalities and adding more command for programming. This could be a way of integration different skills students are acquiring during the study program as a team projects what could lead to development of more complex simulator.

3.2. NEXT STEPS

Improvements can be made and we started a team to add some more complicated functions: multiple selection, rotation and scaling tool and much more. Along with most software improvements we also intend to connect the simulation to a real life model of a robotic arm constructed from a 3D printer which would add to the experience. Further down the line we would add custom robot designs to be implemented at the time of writing this article we are trying to develop a module that all robots can run on starting with a modulated inverse kinematics system. While the current project seems rather simple and unpolished with enough effort and time it could become a benefactor to students and educational institutes.

ACKNOWLEDGMENT

We thank DOK-ING for allowing P.Mamić to experiment on inverse kinematics in a professional working environment. We thank the team of people who have and are currently contributing to the project Filip Bozov, Emil Jagnić, Patrik Kušević and Karlo Gašjak, and we thank colleague Anamarija Sedlar to inspiring us to explore Interactive Educational Tools.

5. REFERENCE
5. REFERENCES

[1.] P. Mamić:" Računalna simulacija rada robotske ruke", Bacherlor thesis, Zagreb University of Applied Sciences, 2020.

[2.] J. Velagić, "Laboratory for Robotics and Autonomous Systems", Zagreb University of Applied Sciences, http://moj.tvz.hr, 04/21/2020.

[3.] D. Matika, "Lessons 1-9", Zagreb University of Applied Sciences,10.03 - 23.05.2020

[4.] M. Spong, S. Hutchinson, M. Vidyasagar "Robot Modeling and Control", 2020.

[5.] The Coding Train, Coding Challenge: "Inverse Kinematics", https://www.youtube.com/watch?v=hbgDqyy8bIw&t=1914s, 2020.

[6.] UConn HKN, "Robotics Inverse Kinematics -Example", https://www.youtube.com/watch?v=f9kxhj5bR6w, 2020.

[7.] "Unity User Manual (2019.4 LTS) ", Unity Documentation, https://docs.unity3d.com/Manual/index.html.

[8.] "RoboSim" GitHub repository https://github.com/Dero1014/RobotSimulation

AUTORI · *AUTHORS*

● **Petar Mamić**
Rođen 1998. u Zagrebu, gdje je završio TŠRB kao mehatroničar i nastavio dalje s TVZom na preddiplomski studij mehatronike i trenutačno studira kao specijalist strojarstva modul mehatronika na istom faksu. Tokom TVZa vježbao je svoje znanje programiranja razvijajući igre preko Unity Engina i C#. Za svoj završni rad je radio na edukacijskom alatu za robotiku te paralelno s DOK-INGom razvijao prototip inverzne kinematike za njihovog robota. Za vrijeme specijalističkog studija se prijavio kao demonstrator osnovama programiranja i kasnije počeo raditi za Ericsson kao softver developer.

Korespondencija · *Correspondence*
petar.mamic@tvz.hr

● **Vesna Alić Kostešić -** nepromjenjena biografija nalazi se u časopisu Polytechnic & Design Vol. 4, No. 2, 2006.

Korespondencija · *Correspondence*
vak@tvz.hr

● **Goran Sirovatka -** nepromjenjena biografija
nalazi se u časopisu Polytechnic & Design
Vol. 5, No. 1, 2017.

Korespondencija · ***Correspondence***
gsirovatka@tvz.hr

● **Vlatko Mićković -** nepromjenjena biografija
nalazi se u časopisu Polytechnic & Design
Vol.9, No.1. 2021

Korespondencija · ***Correspondence***
vmickovic@tvz.hr