EDUCATIONAL MODEL OF THE ROBOT

BOHDAN FETSO[1], MICHAL KELEMEN[1], TATIANA KELEMENOVA[1], IVAN VIRGALA[1], ĽUBICA MIKOVA[1], ERIK PRADA[1], MARTIN VARGA[1], PETER JAN SINCAK[1], LEO BRADA[1]

[1]Technical University of Kosice, Faculty of Mechanical Engineering, Kosice, Slovakia

DOI: 10.17973/MMSJ.2024_11_2024053

michal.kelemen@tuke.sk

The article deals with the design of the educational model of the robot, where, in addition to kinematics, the control system of the robot and the simulation of the robot's activity in the GAZEBO environment are also addressed. Students can train different control algorithms on this model. At the same time, a graphical interface for simulating the robot's activity is also created. The control system is composed of a low-cost embedded Arduino system, which is very easy to program and create control systems. Simulations and experiments showed the correctness of the design methodology of such a robot model.

KEYWORDS

Robot, kinematics, control system, manipulator, mechanism

1 INTRODUCTION

The development of production process management methodology has now significantly accelerated and streamlined the quality of many industrial processes, in which modern management methods and algorithms have played a significant role. It is mainly due to the development of control systems, sensors and communication systems that can optimally control industrial processes in real-time with high accuracy and quality [Wang 2021, Krenicky 2022]. Therefore, robotics is an integral part of product development. A robot is an artificial agent, which means that it serves as a substitute for a person while doing the things for which it is intended. The word "robot" first appeared in 1920 in the drama of the Czech novelist Karel Capek entitled "Rossum's Universal Robots". Although in fiction, robots usually look like humans, in reality, most robots do not, but they are machines controlled by a computer program and electronic circuits.

Robots are beneficial and necessary in the industry because they do a lot of repetitive tasks, dangerous and tedious work instead of people. They are complex systems that require almost all information technology and electrical engineering branches to function correctly. Modern control system components and control algorithms have enabled robotics to research and develop robots with an extensive range of applications in various areas of industry. In addition to industrial production, robotics also appears in healthcare, research, film production, construction, and households.

This work focuses on the types of robots mainly used in industry and has a mechanical structure suitable for handling objects [Zhong 2006]. These types of robots are denoted as industrial manipulators. The work presents what robotics is, what mechanical structures of robots exist and how they are controlled. Furthermore, basic procedures for calculating mathematical models that control robots and procedures for configuring the hardware structure to control mechanical parts will be presented.

One of the vital areas for developing and applying robotic systems is parallel robots, which use cables to achieve movement. This work allows looking deeper into the topic of these types of robots, their advantages over other types of manipulators and what challenges they face. A robot of this kind was chosen for the practical part of this work. Easily accessible software and hardware components were used to assemble such a robot.

2 ROBOT CONFIGURATION DESIGN

The robot manipulator uses three servo motors with mounted arm links (shoulder and elbow). In addition, an end effector is placed on the tip of the robot wrist, which has an articulated connection with the robot arm and base. This connection is required for the robot's end effector to keep a stable horizontal position relative to a surface on which the robot's base is mounted [Krenicky 2022].

The dimensions of the robot are approximately 310.36×124×257 mm, see Figure 1. The maximum designed load capacity of the robot that it should be able to handle is approximately 0.5 kilograms. Its task will be to perform movements set by the operator, grip and move objects with the end effector.

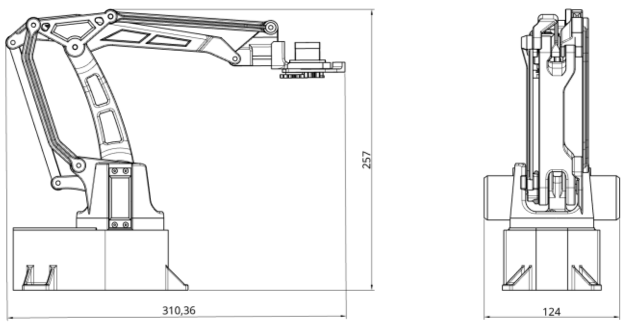


Figure 1. Construction and dimensioning of the robot

3 ROBOT CONTROL AND SIMULATION

To control a robot, we need to create its kinematic model. That is at least its inverse kinematics. Then there is a need to select how the calculated values will be sent to the individual motors, i.e., connect the outputs of the kinematic model to a particular network. When considering how this will be achieved, it is good to examine the options available properly [Bozek 2023]. The following section will show what results were found when searching for a solution.

Simulation is a crucial phase in the design of any robot. Simulators allow us to quickly test new designs and see how this implementation fits the design before starting the hard work of designing a real robot. It can be risky for robots if the designer has designed a model that has not been tested in the simulation. The robot can be unreliable if some parameters are not considered.

Various software platforms have been developed to study spatial serial and parallel robots, using the ROS (Robotic Operating System) control and simulation software. Examples of such existing platforms are MATLAB/Simulink and ZeroSim. These platforms are flexible for the possibility of adding their serial and parallel robots of various types, and it is possible to extend them with other algorithms. However, Gazebo is the ROS-integrated simulation platform for studying any type of robot. It allows creating and analysing of any parallel robot with the ability to customise algorithms.

With Gazebo, it is possible to use different simulation engines. ODE (Open Dynamics Engine) is the default. It also supports Bullet (Bullet Real-Time Physics Simulation), but currently, not all features. Therefore, using Bullet as an extension for Gazebo is better for kinematic calculation and simulation.

The current version of the platform includes analytical tools for each of the following areas:

• Dynamics and control
• Motion and sensor control
• Kinematics
  o Direct kinematics
  o Inverse kinematics
• Workspace analysis
• Design optimisation

It also offers a graphical interface for a complex solution, making the selection of a given robot and kinematic model and the control and triggering of movements very simple.

A new parallel robot model can be added to Gazebo by specifying it in semi-structured XML data files. There are four main files, robot.urdf, config.xacro, spawn.launch and empty.world. Robot.urdf defines the robot body, links, and joints. Config.xacro contains a set of link layouts, connection points of joints, and properties. Spawn.launch is a set of how files must load in Gazebo and Rviz, set properties for corresponding features of the robot through the files mentioned earlier and apply any additional configuration files. Finally, empty.world is a world that surrounds the robot. It is required for the robot to spawn on the ground underneath instead of endlessly falling in the simulation. The simulation can have more results. The most interesting result that can be used for hardware implementation is the generated collision boxes of the robot links. It allows to properly model robot parts and avoids possible jamming. Orientation of each DOF is generated using inverse kinematics. The simulator calculates the circular degree of its joints for each robot's position during the execution of the trajectory and saves them in a table in chronological order.

The Gazebo is primarily a simulation platform and does not offer a hardware implementation but leaves room for it to be extended. It offers source code methods and protocols developers can use to utilise simulation results for actual deployment to an existing robot. ROS is one of the best existing means of connecting robotic hardware, which provides a well-supported interface for expansion and integration with Gazebo. In this way, it is convenient to set up easy-to-use robotic hardware and take advantage of the flexibility and robustness of Gazebo.

The Arduino platform will be used for solving this work for hardware implementation. In this way, it is best to use object-oriented principles of the C++ program to cooperate with the Arduino board. Arduino creates an algorithm in which all processes are connected in a loop mode. Any node in the system works with other nodes in synchronisation. Figure 2 shows a simplified Arduino program architecture for the robot. Nodes represent the processes in which the calculations are performed. A system of numerous nodes is created on the Arduino to control various functions. It is better to have numerous nodes that provide only one function than one complex node that creates everything in the system. The program is flashed on the Arduino board through the USB interface via the Arduino IDE or Arduino CLI user programs.

A program written in C++ is used on the PC to control Arduino, which establishes the connections between the PC and the Arduino board and allows them to communicate. This main program is only for communication between the user and the robot and does not contain a program to control the robot's servo motors.

The communication programs on the PC and Arduino communicate by sending messages over a USB serial port. The input contains data that gives Arduino information about which servo motor needs to change position and by what amount.

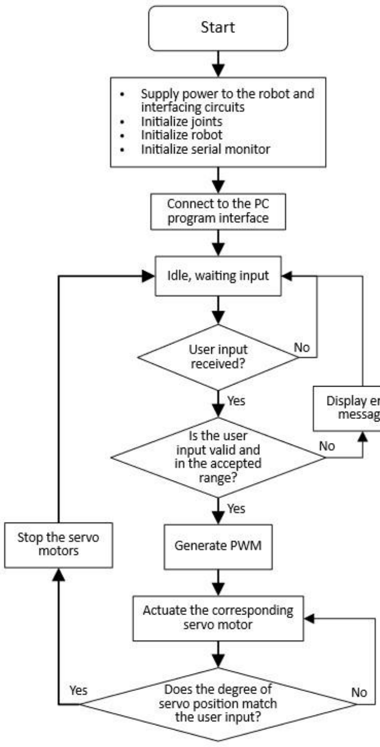

Start
Supply power to the robot and interfacing circuits
Initialize joints
Initialize robot
Initialize serial monitor
Connect to the PC program interface
Idle, waiting input
User input received?
No
Yes
Display error message
Is the user input valid and in the accepted range?
No
Yes
Stop the servo motors
Generate PWM
Actuate the corresponding servo motor
Yes
Does the degree of servo position match the user input?
No

Figure 2. Architecture of the Arduino program

Robot Control System (RCS) is a flexible robot control software for real-time positioning and setting specific pre-built applications for the robot to work with (Figure 3). It is designed to be compatible specifically with the robot for this thesis, but it is possible to reconfigure the application to use it on other robot configurations that use the Arduino board as the central controller. The RCS is a collection of tools and libraries aimed at simplifying the task of creating complex and robust robot behaviours and robot architectures. Its structure consists of many nodes that communicate with each other using the object-oriented programming language C++ principles.

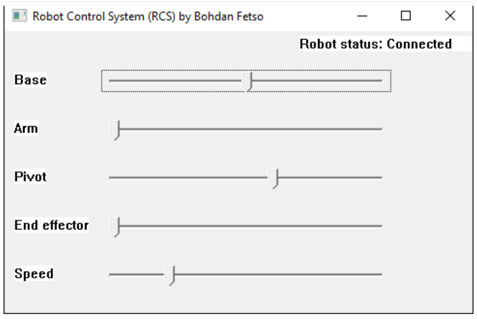


Figure 3. RCS primary control window

4 ROBOT HARDWARE TOPOLOGY

The robot model described in the preceding section was used to design its hardware topology. Easily accessible and cost-effective hardware components are proposed for the solution. In the proposed solution, it was necessary to divide the task of one component into two or more hardware that works together as one.

The control unit consists of a network-connected desktop computer-controlled by the operator, who defines the robot's operation. For simulation, Gazebo will be communicating with ROS through the local host of the computer. At the same time, the Arduino board attached to the robot frame will be connected to the computer via a USB interface. Furthermore, it will be communicating with special written control software. This application waits for a user's input, later sent from the computer, to calculate motor commands on the running Arduino node. The Arduino microcontroller, while connected, executes the motor commands it received through the direct servo motor connections. The next chapter describes more information about connecting and operating the robot in terms of hardware and software.

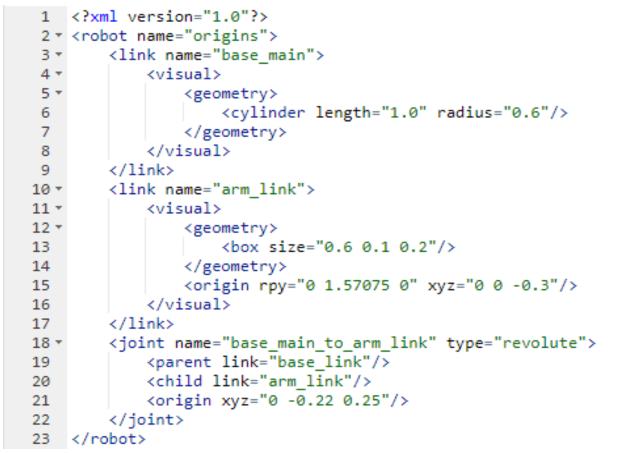
5 ROBOT MANIPULATION CONFIGURATION

The ROS and Gazebo source code is open-source and published on the GitHub website, offering downloadable and cloning options. Due to the constant development of this simulator, any user can install it on a computer and receive new release updates with new functions and bug fixes. Furthermore, it will provide access to the latest features and AI algorithms. The simulator can be launched from Ubuntu using a script that is opened in a terminal.

A new robot can be created and configured using Unified Robotic Description Format (URDF) files. URDF is a universal and open format for robot configurations (similar to SDF, XML). URDF is a simple format that does not carry information about the appearance of a robot but only about its content.

The physical properties of the robot's body, links and joints are defined in the robot.urdf file. The following code will show the basic structure of the robot.urdf file with the data of the robot manipulator of this work (Figure 4).

The contents of the robot.urdf file is enclosed with a </robot> tag, containing at least one link tagged with arm_link. By link is meant the body structure of the simulated robot. Within the tag, the geometric dimensions of the robot parts are given in the order they will be built during the robot simulation. Other tags define the properties of the end effector and arm joints, including the minimum and maximum possible angle orientation and their geometrical representations.

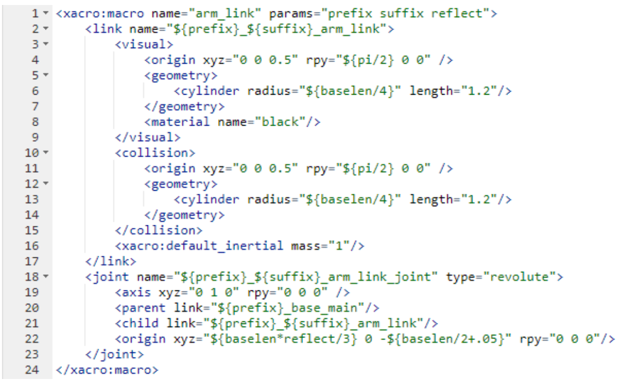


```xml
<?xml version="1.0"?>
<robot name="origins">
    <link name="base_main">
        <visual>
            <geometry>
                <cylinder length="1.0" radius="0.6"/>
            </geometry>
        </visual>
    </link>
    <link name="arm_link">
        <visual>
            <geometry>
                <box size="0.6 0.1 0.2"/>
            </geometry>
            <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
        </visual>
    </link>
    <joint name="base_main_to_arm_link" type="revolute">
        <parent link="base_link"/>
        <child link="arm_link"/>
        <origin xyz="0 -0.22 0.25"/>
    </joint>
</robot>
```

Figure 4. Sample code snippet of the robot.urdf file

Body positions and properties are defined in the config.xacro file. It is necessary to define the exact positions of endpoints in each part. For example, a point that connects the base with the shoulder and provides a vertical rotation for the robot. The maximum load capacity of the end effector is also defined, and the minimum force must be applied to the joint point. The following code shows a file structure that configures the robot manipulator of this project as can be seen on Figure 5.

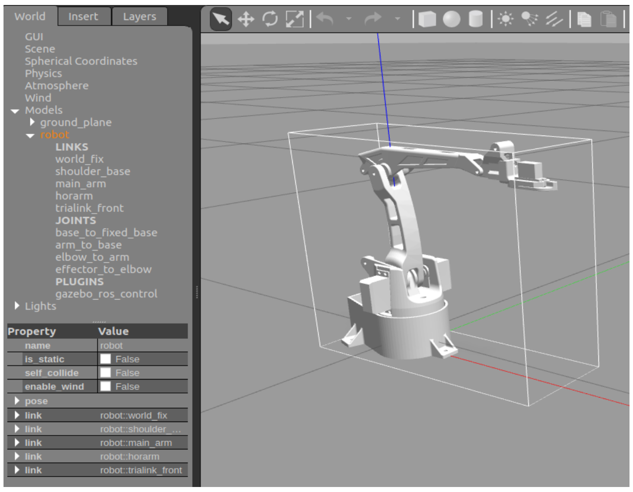
The file precisely defines the arrangement of the links across multiple tags. For example, the link positions and collisions are indicated in the visual and collision tags of the origin tag. In addition, each joint has its tag and other mandatory tags within it that specify its properties. Such mandatory properties are a type of a joint (i.e. revolute, fixed, prismatic), its position, parent body and interaction between parent and child objects.



```xml
<xacro:macro name="arm_link" params="prefix suffix reflect">
    <link name="${prefix}_${suffix}_arm_link">
        <visual>
            <origin xyz="0 0 0.5" rpy="${pi/2} 0 0" />
            <geometry>
                <cylinder radius="${baselen/4}" length="1.2"/>
            </geometry>
            <material name="black"/>
        </visual>
        <collision>
            <origin xyz="0 0 0.5" rpy="${pi/2} 0 0" />
            <geometry>
                <cylinder radius="${baselen/4}" length="1.2"/>
            </geometry>
        </collision>
        <xacro:default_inertial mass="1"/>
    </link>
    <joint name="${prefix}_${suffix}_arm_link_joint" type="revolute">
        <axis xyz="0 1 0" rpy="0 0 0" />
        <parent link="${prefix}_base_main"/>
        <child link="${prefix}_${suffix}_arm_link"/>
        <origin xyz="${baselen*reflect/3} 0 -${baselen/2+.05}" rpy="0 0 0"/>
    </joint>
</xacro:macro>
```

Figure 5. Configuration of the robotic arm in the config.xacro file

These files describing the robot model must be set up correctly so that the robot model can be inserted into the simulator. It is possible to examine whether the model has been defined correctly in ROS. After each simulator initialisation, the available robot models are loaded. Through the graphical interface Gazebo, it is possible to select the robot model with which work must be done according to the name entered in the file (Figure 6). In the displayed robot model area, it is possible to quickly check whether the entered information has been correct.



Property	Value
name	robot
is_static	☐ False
self_collide	☐ False
enable_wind	☐ False
pose	
link	robot::world_fix
link	robot::shoulder_...
link	robot::main_arm
link	robot::horarm
link	robot::trialink_front

Figure 6. Gazebo graphical interface with the loaded robot

Within the Arduino, robot trajectories can be programmed. The determination of one trajectory is possible inside a particular matrix, which is sent to Arduino for processing. The individual degree values are entered into this matrix, determining the end effector *x*, *y*, and *z* positions, respectively, the robot's orientation. These points are preprogrammed inside the point tagging field of the Arduino program. Adding a value with the servo motor speed attribute to the end value makes it possible to specify how long the robot should go from one point to another. In the simulation sample, the robot starts from the starting point and rotates 45 degrees left and moves forward by 30 cm for five seconds (Fig. 7).

In the spawn.launch file; it is possible to manually set initial options for the simulation, i.e., the configuration files of a trajectory generation for the robot (Figure 8). Within a joint_state_publisher node, ROS automatically defines multiple joint positions, each under its name, specified in the robot.urdf file (Figure 9). Each joint state has to be specified inside the URDF file of the robot under a joint tag. Inside the tag is declared a name of a respective joint and its type (i.e., revolute, continuous, prismatic, et cetera). After defining the root tag, the position of robot links relative to each other must be specified in the origin tag. Then, the individual connection points *x*, *y*, and *z* must be inserted. Finally, roll, pitch and yaw parameters are set using *r*, *p* and *y*.

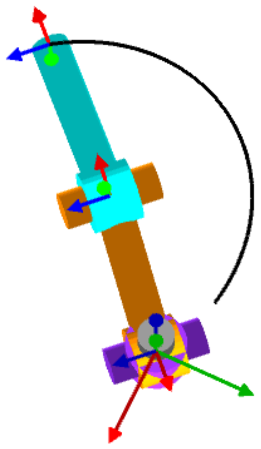
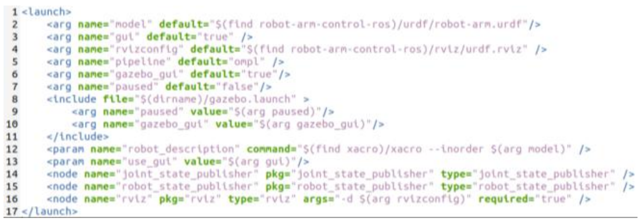


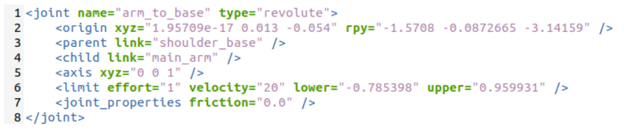
Figure 7. Trajectory generation of the robot



```xml
<launch>
    <arg name="model" default="$(find robot-arm-control-ros)/urdf/robot-arm.urdf"/>
    <arg name="gui" default="true" />
    <arg name="rvizconfig" default="$(find robot-arm-control-ros)/rviz/urdf.rviz" />
    <arg name="pipeline" default="ompl" />
    <arg name="gazebo_gui" default="true"/>
    <arg name="paused" default="false"/>
    <include file="$(dirname)/gazebo.launch" >
        <arg name="paused" value="$(arg paused)"/>
        <arg name="gazebo_gui" value="$(arg gazebo_gui)"/>
    </include>
    <param name="robot_description" command="$(find xacro)/xacro --inorder $(arg model)" />
    <param name="use_gui" value="$(arg gui)"/>
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />
</launch>
```

Figure 8. Configuration of the spawn.launch file

After those parameters, a parent and a child link must be set. It ensures that the links are held together in a tree-manner structure. In the case of the figure example, a connection between the robot's shoulder and main arm link is shown. Then, inside the axis parameter, the direction of the rotation is selected for the revolute joint. Finally, in the limit tag, specific velocity and degree limits of the rotation can be set. They are not mandatory, but they help ensure that the robot arm has the same limits as the actual model.



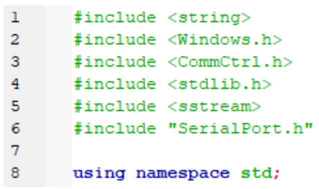
```xml
<joint name="arm_to_base" type="revolute">
    <origin xyz="1.95709e-17 0.013 -0.054" rpy="-1.5708 -0.0872665 -3.14159" />
    <parent link="shoulder_base" />
    <child link="main_arm" />
    <axis xyz="0 0 1" />
    <limit effort="1" velocity="20" lower="-0.785398" upper="0.959931" />
    <joint_properties friction="0.0" />
</joint>
```

Figure 9. Example of a defined joint

After setting up all the necessary files, it is possible to select the trajectory and set the kinematic model processing in Rviz for the Gazebo simulation. This project will show how the simulator generates movement using an inverse kinematics solver. Therefore, the option of the solver an LMA kinematics plugin is selected. This plugin obeys joint limits specified in the URDF (and will use the safety limits if specified in the URDF). ROS only calls the inverse kinematics solver for one pose (it may occur multiple times if the first result is invalid, i.e., due to self-collisions). This solution provides a joint configuration for the Rviz. ROS already knows the current joint configuration from the URDF file and a preset default position. Thus, all trajectory planning and execution at that point is done in a joint space inside Rviz. Collision detection and constraint checking may use inverse kinematics to determine any subgoal joint configuration pose, but the planning is not done in Cartesian space. After a joint trajectory is found, ROS tries to smooth the trajectory to make it less jittering moving, but this does not always result in a path that is the fastest one.

A unique software application was made to control the robot manipulator. This program is created using the native Win32 programming blocks from Windows OS and the open-source Arduino serial library, securing communication between the app and the Arduino board. Since this application uses C++ programming language, it has exact mechanisms for declaring libraries using #include as the Arduino program. However, the list of libraries is different because distinct functions are required for this application to work correctly (Figure 10).

Additionally, in C++, it is recommended to use so-called namespaces. A namespace is a declarative section that delivers a scope to the descriptors (such as names of types, functions, variables). They unite code into logical clusters and avoid name collisions that can transpire mainly when the codebase contains multiple libraries. Furthermore, all identifiers at namespace scope are visible to one another without qualification.

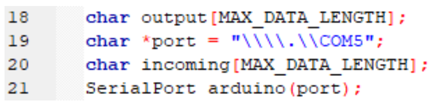


```cpp
#include <string>
#include <Windows.h>
#include <CommCtrl.h>
#include <stdlib.h>
#include <sstream>
#include "SerialPort.h"

using namespace std;
```

Figure 10. Library declaring in RCS source code

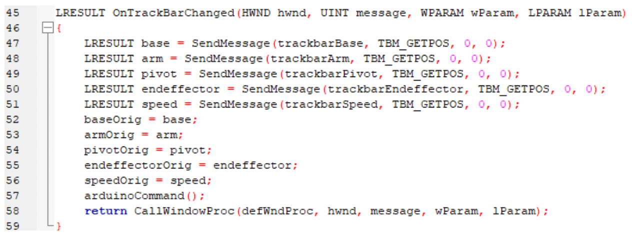
The code snippet in Figure 11 shows how the connection between Arduino and the program is established. Firstly, primary input and output variables are created to guarantee the possibility of data transfer. They also pass the MAX_DATA_LENGTH argument, which gives the ability for the C++ application to send more than 255 character messages to the Arduino. Next, a variable pointer is created, which holds an address to the Arduino COM port. For a C++ program, a computer's memory is like a series of memory blocks, each one byte in size and respectively with a unique address. These single-byte memory blocks are ordered in a way that allows data representations larger than one byte to occupy memory that has consecutive addresses. A fascinating property of pointers is that they can be used to access the variable they point to directly. This property is done by preceding the pointer name with the dereference operator (*). The operator itself can be read as "value pointed to by".



```cpp
char output[MAX_DATA_LENGTH];
char *port = "\\\\.\\COM5";
char incoming[MAX_DATA_LENGTH];
SerialPort arduino(port);
```

Figure 11. RCS serial connection code

In order to receive and process the track bar value from the window process, the LRESULT function is used with a TBM_GETPOS method (Figure 12). This method retrieves the current logical position of the slider in a trackbar. The logical positions are the integer values in the trackbar's minimum to maximum slider positions range. Then this position is stored in the variable of the future corresponding servo motor location. If they are in the range of the allowed area on the track bar itself, they get stored in the continuous variables, which will be passed into the verification function later on.

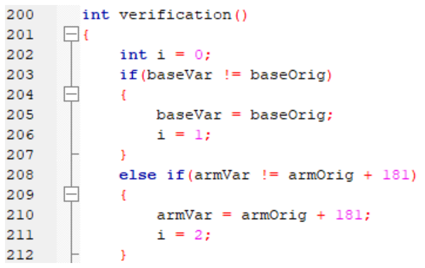


```cpp
LRESULT OnTrackBarChanged(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    LRESULT base = SendMessage(trackbarBase, TBM_GETPOS, 0, 0);
    LRESULT arm = SendMessage(trackbarArm, TBM_GETPOS, 0, 0);
    LRESULT pivot = SendMessage(trackbarPivot, TBM_GETPOS, 0, 0);
    LRESULT endeffector = SendMessage(trackbarEndeffector, TBM_GETPOS, 0, 0);
    LRESULT speed = SendMessage(trackbarSpeed, TBM_GETPOS, 0, 0);
    baseOrig = base;
    armOrig = arm;
    pivotOrig = pivot;
    endeffectorOrig = endeffector;
    speedOrig = speed;
    arduinoCommand();
    return CallWindowProc(defWndProc, hwnd, message, wParam, lParam);
}
```

Figure 12. RCS track bar operation code

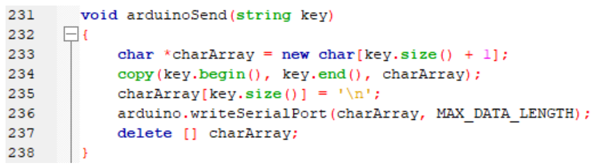
A simple conditional statement is set to verify the correctness of the received data and process it back to a send function. First, it checks if the data was changed compared to the old received data. Then, the corresponding data passes to the callback and gets processed in the messaging function if it was altered (Figure 13).

The message handling function (Figure 14) consists of an array building variable, which processes received data into a character array. This variable is calculated in size to ensure no data leak and parsed additionally inside the Arduino program to ensure the integrity of the received data.



```
int verification()
{
    int i = 0;
    if(baseVar != baseOrig)
    {
        baseVar = baseOrig;
        i = 1;
    }
    else if(armVar != armOrig + 181)
    {
        armVar = armOrig + 181;
        i = 2;
    }
```

Figure 13. Snippet of verification function code



```
void arduinoSend(string key)
{
    char *charArray = new char[key.size() + 1];
    copy(key.begin(), key.end(), charArray);
    charArray[key.size()] = '\n';
    arduino.writeSerialPort(charArray, MAX_DATA_LENGTH);
    delete [] charArray;
}
```

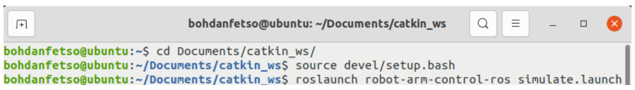
Figure 14. RCS message sending function code

Then the data is sent to the Arduino board using arduino.writeSerialPort(), which passes the charArray and removes the 255 character limitation of the sent data. Finally, in C++, memory can be managed manually. Therefore, removing the character array after it was used to free up the declared memory and prevent any possible memory leaks is a beneficial practice to do it. Also, this prevents any possible variable distortion after the new cycle of the arduinoSend() function begins.

In the following subchapters, tests will be done, demonstrating both Gazebo simulation and actual robot movement using the RCS software.

6 SIMULATION WITH GAZEBO

In order to launch the Gazebo simulation of this project, first, a terminal window needs to be opened inside Ubuntu OS (Figure 15). Then a project directory has to be selected to make it easier to execute ROS commands. This selection is made via the Linux command cd (change directory) and the path to the project folder, which must be separated with a slash (/) symbol for each subfolder. Next, a source file has to be loaded to the bash environment. After compiling a ROS package for this project, this file was obtained and is located in the devel directory.



```
bohdanfetso@ubuntu: ~/Documents/catkin_ws
bohdanfetso@ubuntu:~$ cd Documents/catkin_ws/
bohdanfetso@ubuntu:~/Documents/catkin_ws$ source devel/setup.bash
bohdanfetso@ubuntu:~/Documents/catkin_ws$ roslaunch robot-arm-control-ros simulate.launch
```

Figure 15. Ubuntu terminal with the executed launching commands

Finally, the simulation can start with a roslaunch command with specified arguments of the project's name and which launch file to load. When all these commands are executed in this order, Gazebo and Rviz windows open up on the screen after loading.

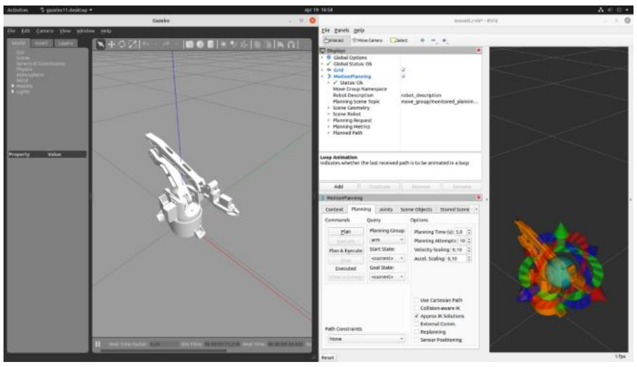
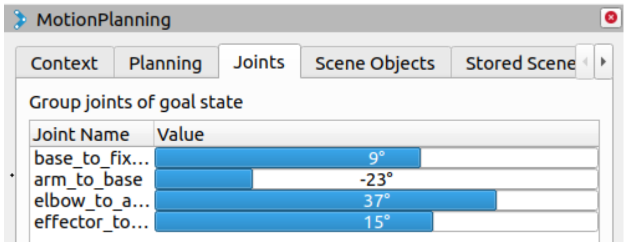


Figure 16. Gazebo and Rviz simulation windows

Then, automatically, the robot files are loaded, and its model with configurations gets displayed in both programs, see Figure 16. On the control panel are shown robot controls. Here can be selected which planning groups will be transformed for the trajectory generation and which kinematic module will be used to generate coordinates for the robot's joints. Furthermore, path-planning can be done in several ways, for example, directly dragging the robot's end effector on a specific position or selecting a joint tab and manually inputting degree values for the respective joints (Figure 17). Finally, the hand-operated angle method can be used as an additional controlling unit for the RCS application. As it will help determine trackbar positions for the anticipated manipulator orientation.



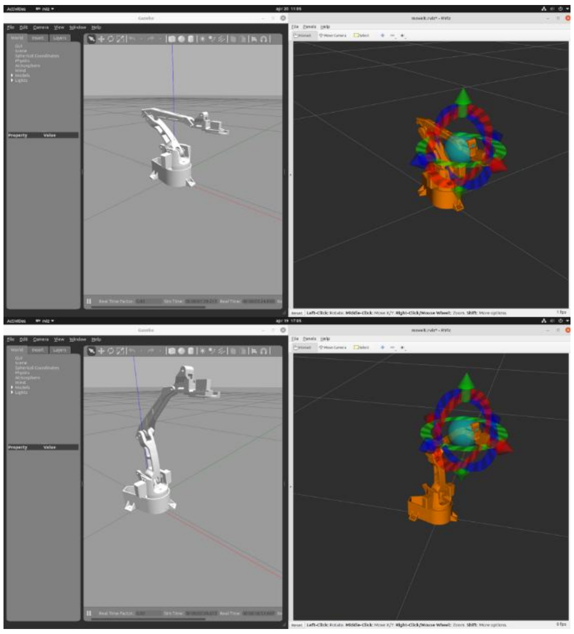
MotionPlanning

Context | Planning | Joints | Scene Objects | Stored Scene

Group joints of goal state

Joint Name	Value
base_to_fix...	9°
arm_to_base	-23°
elbow_to_a...	37°
effector_to...	15°

Figure 17. Rviz manual degree setting for the robot's joints



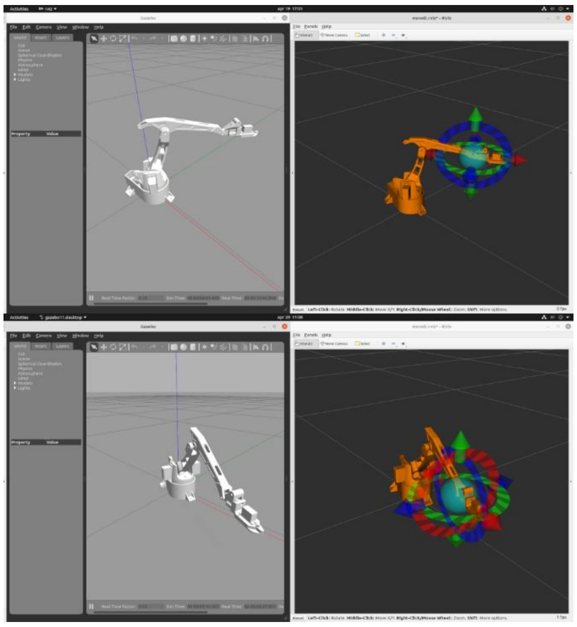


Figure 18. Gazebo robot movement simulation with Rviz

The first test is moving the end effector manually. This motion is made either with arrows forming a Cartesian coordinate system or a blue sphere, indicating the end effector location. Figure 18 shows how inside the Rviz window, the robot arm is moved in a specific direction and how Gazebo simulates the movement in the same direction using inverse kinematics planning.
As shown above, the robot can move just in any direction commanded by Rviz. However, all motion is restricted to avoid exceeding restricted maximal joint angles. These constraints are set in the same way as the actual robot model's limitation, thus ensuring the correctness of the simulation data. Furthermore, it is possible to insert different objects into the Gazebo simulation to simulate a robot's environment (Fig. 19).

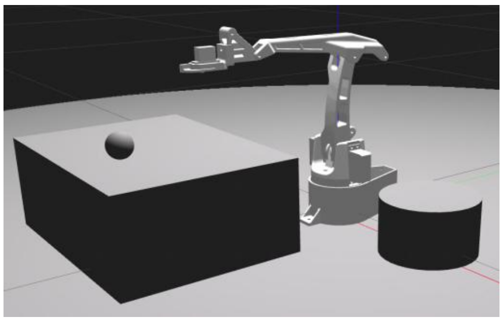


Figure 19. Robot's environment simulation

Figure 19 shows how different geometrical objects are generated inside the simulation window.
This potential makes it possible to create industrial surroundings for the manipulator with collisions and physics. Furthermore, it is possible to generate a trajectory motion for the end effector that avoids hitting obstacles on its way.

7 ROBOT CONTROL SYSTEM TEST

This test requires the actual robot to be present and connected to the host computer with the robot control system (RCS) application installed. The manipulator's Arduino board has a USB type B port. It will connect the robot directly with a cable to any available computer USB. However, by default, the computer will not be able to operate with the board. Thus, an official Arduino driver must be installed to ensure the proper robot operation. Then the RCS application can be launched by simply double-clicking the executable file (Figure 20).

Figure 20. Robot Control System executable

A greeting window will appear on a user's monitor when everything is installed correctly. In addition, the RCS has a robot state notification, which allows the user to see if the robot is connected to the system (Figure 21).

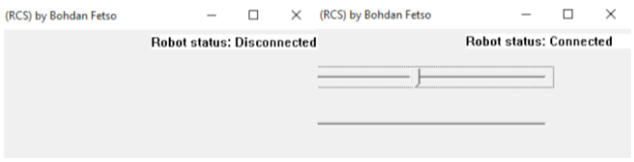


Figure 21. Robot Control System verification

Furthermore, if the connection is suddenly interrupted, RCS will change the state and notify the user about disconnection. If the connection gets re-established, the program automatically proceeds with its workflow.
Next, motion planning can be started with the robot connected to the computer and RCS launched. Each servo motor joint was given a respective name and listed in the same order as the kinematic chain of the Gazebo simulation. Finally, several positioning tests were made and are demonstrated in Figure 22. Moreover, in this figure, the fourth section demonstrates how the end effector works by holding a king's chess piece. On this scale, the end effector is powerful enough to hold more challenging objects, for example, screws, nuts, small wood and metal plates.

8 CONCLUSIONS

Faster and more accurate algorithms have a significant impact on the overall operation of the robot. For example, choosing a more precise kinematics calculation module could make the robot even more stable. The friction and bending that occurs when the robot bends over the longest point significantly affect the shaking and, thus, the overall system's stability.
With minor alterations to the hardware and software, such a robot could be even more stable and thus suitable for activities that require more precise movements, such as a small conveyor manipulator. So far, such a robot is suitable for moving smaller objects, which can be helpful for personal use, for example, sorting, object handling, and other automation processes.
The robot's interface could be extended to accept commands in multiple forms. For example, it could be controlled via a real-time Gazebo ROS simulation, where the manipulator would directly take commands for each incoming move from ROS, thus performing complicated movements.

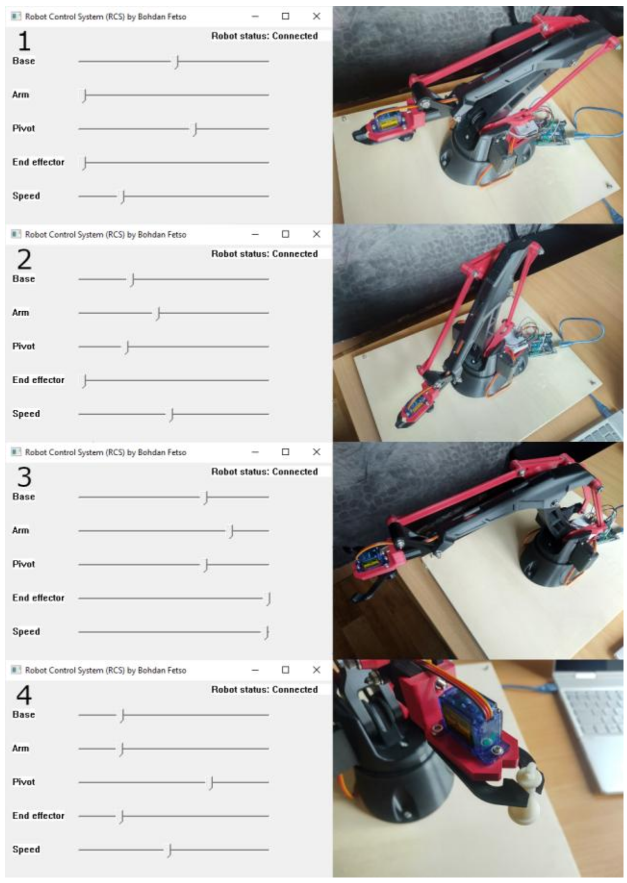


Figure 22. Robot movement test with RCS

The development of similar applications will certainly help to improve the knowledge, abilities and skills of students and graduates [Bozek 2012, 2016 & 2021, Bezak 2014, Koniar 2014, Mikova 2014, Ostertag 2014, Kelemen 2018 & 2021, Liptak 2018, Pavlasek 2018, Zidek 2018, Saga 2019 & 2020, Tlach 2019, Oscadal 2020, Kelemenova 2021, Kuric 2021, Virgala 2012, 2014a,b & 2021, Vagas 2022, Bratan 2023, Vagas 2023 & 2024, Romancik 2024]. To control these models and systems, it is necessary to create a control unit with a microcontroller to effectively use the capabilities of this actuator [Kelemen 2012 & 2014].

ACKNOWLEDGMENTS

The authors would like to thank the Slovak Grant Agency—project KEGA 027TUKE-4/2022, VEGA 1/0201/21 and VEGA 1/0436/22.

REFERENCES

[Bezak 2014] Bezak, P., et al. Advanced Robotic Grasping System Using Deep Learning. Procedia Engineering, 2014, Vol. 96, pp. 10-20. ISSN 1877-7058. https://doi.org/10.1016/j.proeng.2014.12.092.

[Bozek 2012] Bozek, P., Pivarciova, E. Registration of Holographic Images Based on Integral Transformation. Computing and Informatics, 2012, Vol. 31, No. 6, pp. 1369-1383.

[Bozek 2016] Bozek, P., et al. Geometrical Method for Increasing Precision of Machine Building Parts. Procedia Engineering, 2016, Vol. 149, pp. 576-580. https://doi.org/10.1016/j.proeng.2016.06.708.

[Bozek 2021] Bozek, P., Krenicky, T., Nikitin, Y. Editorial for Special Issue "Automation and Robotics: Latest Achievements, Challenges and Prospects". Appl. Sci., 2021, Vol. 11, 12039. https://doi.org/10.3390/app112412039.

[Bozek 2023] Bozek, P., Krenicky, T., Prajova, V. Digital Induction Motor Model Based on the Finite Element Method. Applied Sciences, 2023, Vol. 13, No. 8, 5124. https://doi.org/10.3390/app13085124.

[Bratan 2023] Bratan, S., Sagova, Z., Saga, M., Yakimovich, B., Kuric, I. New Calculation Methodology of the Operations Number of Cold Rolling Rolls Fine Grinding. Applied Sciences, 2023, Vol. 13, No. 6. eISSN 2076-3417, DOI: 10.3390/app13063484.

[Kelemen 2012] Kelemen, M., et al. Design and Development of Lift Didactic Model within Subjects of Mechatronics. Procedia Engineering, 2012, Vol. 48, pp. 280-286. DOI: 10.1016/J.PROENG.2012.09.515.

[Kelemen 2014] Kelemen, M., et al. Rapid Control Prototyping of Embedded Systems Based on Microcontroller. Procedia Engineering, 2014, Vol. 96, Iss. 11, pp. 215-220. https://doi.org/10.1016/j.proeng.2014.12.146.

[Kelemen 2015] Kelemen, M., et al. Experimental Verification of the Shape Memory Alloy (SMA) Spring Actuator for Application on In-Pipe Machine. Metalurgija, 2015, Vol. 54, No. 1, pp. 173-176. ISSN 0543-5846.

[Kelemen 2018] Kelemen, M., et al. A Novel Approach for a Inverse Kinematics Solution of a Redundant Manipulator. Applied Sciences, 2018, Vol. 8, Issue 11, pp. 1-20. https://doi.org/10.3390/app8112229.

[Kelemen 2021] Kelemen, M., et al. Head on Hall Effect Sensor Arrangement for Displacement Measurement. MM Science Journal, 2021, Vol. Oct., Issue 11, pp. 4757-4763. DOI: 10.17973/MMSJ.2021_10_2021026.

[Kelemenova 2021] Kelemenova, T., et al. Verification of Force Transducer for Direct and Indirect Measurements. MM Science J., 2021, Vol. Oct., Issue 11, pp. 4736-4742. DOI: 10.17973/MMSJ.2021_10_2021021.

[Koniar 2014] Koniar, D., et al. Virtual Instrumentation for Visual Inspection in Mechatronic Applications. Procedia Engineering, 2014, Vol. 96, pp. 227-234. DOI: 10.1016/j.proeng.2014.12.148.

[Krenicky 2022] Krenicky, T., Nikitin, Y., Bozek, P. Model-Based Design of Induction Motor Control System in MATLAB. Appl. Sci., 2022, Vol. 12, 11957.

[Krenicky 2022] Krenicky, T., Olejarova, S., Servatka, M. Assessment of the Influence of Selected Technological Parameters on the Morphology Parameters of the Cutting Surfaces of the Hardox 500 Material Cut by Abrasive Water Jet Technology. Materials, 2022, Vol. 15, 1381.

[Kuric 2021] Kuric, I., et al. Analysis of Diagnostic Methods and Energy of Production Systems Drives. Processes, 2021, Vol. 9, 843. doi.org/10.3390/pr9050843.

[Liptak 2018] Liptak, T., et al. Modeling and control of two-link snake. International Journal of Advanced Robotic Systems, 2018, Vol. 15, Issue 2, pp. 1-13. DOI: 10.1177/1729881418760638.

[Mikova 2014] Mikova, L., et al. Simulation Model of Manipulator for Model Based Design. Applied Mechanics and Materials, 2014, Vol. 611, No. 1, pp. 175-182. https://doi.org/10.4028/www.scientific.net/AMM.611.175.

[Mikova 2015] Mikova, L., et al. Application of Shape Memory Alloy (SMA) as Actuator. Metalurgija, 2015, Vol. 54, No. 1, pp. 169-172. ISSN 0543-5846.

[Oscadal 2020] Oscadal, P., et al. Improved Pose Estimation of Aruco Tags Using a Novel 3D Placement Strategy.

Sensors, 2020, Vol. 20, Issue 17, pp. 1-16. ISSN 1424-3210. DOI: 10.3390/S20174825.

[Ostertag 2014] Ostertag, O., et al. Miniature Mobile Bristled In-Pipe Machine. International J. of Advanced Robotic Systems, 2014, Vol. 11, pp. 1-9. ISSN 1729-8806. https://doi.org/10.5772/59499.

[Pavlasek 2018] Pavlasek, P., et al. Flexible Education Environment: Learning Style Insights to Increase Engineering Students Key Competences. Edulearn18 Proceedings, 2018.

[Romancik 2024] Romancik, J., et al. Design, Implementation, And Testing of a 3D printed Gripper Actuated by Nitinol Springs. MM Science J., Vol. June, pp. 7352-7356. DOI: 10.17973/MMSJ.2024_06_2024009.

[Saga 2019] Saga, M., et al. Contribution to Random Vibration Numerical Simulation and Optimisation of Nonlinear Mechanical Systems. Sci. J. of Silesian Uni. of Technology-Series Transport, 2019, Vol. 103, pp. 143-154. DOI: 10.20858/sjsutst.2019.103.11.

[Saga 2020] Saga, M., et al. Case study: Performance analysis and development of robotized screwing application with integrated vision sensing system for automotive industry. International J. of Advanced Robotic Systems, 2020, Vol. 17, No. 3, pp. 1-23. https://doi.org/10.1177/1729881420923997.

[Tlach 2019] Tlach, V., et al. Collaborative assembly task realization using selected type of a human-robot interaction. Transportation Research Procedia, 2019, Vol. 40, pp. 541-547. DOI: 10.1016/j.trpro.2019.07.078, 2019.

[Vagas 2022] Vagas, M., et al. Testing of Ethernet-based communication between control PLC and collaborative mechatronic system. In: 20th Int. Conf. on Mechatronics (ME). Pilsen, Czech Republic, 2022. DOI: 10.1109/ME54704.2022.9983428.

[Vagas 2023] Vagas, M., et al. Calibration of an intuitive machine vision system based on an external high-performance image processing unit. In: 24th International Conference on Process Control (PC), Strbske Pleso, Slovakia, 2023, pp. 186-191. DOI: 10.1109/PC58330.2023.10217606.

[Vagas 2024] Vagas, M., et al. Implementation of IO-the Handling and Sorting Sub-Station of the Festo FMS 500 Automated Line. MM Science Journal, 2024, Vol. June, pp. 7348-7351. DOI: 10.17973/MMSJ.2024_06_2024008.

[Virgala 2012] Virgala, I., et al. Manipulator End-Effector Position Control. Procedia Eng., 2012, Vol. 48, pp. 684-692. doi.org/10.1016/j.proeng.2012.09.571.

[Virgala 2014a] Virgala, I., et al. Analyzing, Modeling and Simulation of Humanoid Robot Hand Motion. Procedia Engineering, 2014, Vol. 611, pp. 75-82. doi.org/10.4028/www.scientific.net/AMM.611.75.

[Virgala 2014b] Virgala, I., et al. Inverse Kinematic Model of Humanoid Robot Hand. Applied Mechanics and Materials, 2014, Vol. 96, pp. 489-499. https://doi.org/10.1016/j.proeng.2014.12.121.

[Virgala 2021] Virgala, I., et al. A snake robot for locomotion in a pipe using trapezium-like travelling wave. Mechanism and Machine Theory, 2021, Vol. 158, 104221. DOI: 10.1016/J.MECHMACHTHEORY.2020.104221.

[Wang 2021] Wang, W., et al. Controlling bending deformation of a shape memory alloy-based soft planar gripper to grip deformable objects. International Journal of Mechanical Sciences, 2021, Vol. 193, No. 1, pp. 1-8. https://doi.org/10.1016/j.ijmecsci.2020.106181.

[Zidek 2018] Zidek, K., et al. Auxiliary Device for Accurate Measurement by the Smartvision System. MM Science Journal, 2018, Volume March, pp. 2136-2139. DOI: 10.17973/MMSJ.2018_03_201722.

[Zhong 2006] Zhong, Z.W. and Yeong, C.K. Development of a gripper using SMA wire. Sensors and Actuators A: Physical, 2006, Vol. 126, Issue 2, pp. 375-381. https://doi.org/10.1016/j.sna.2005.10.017.

CONTACTS:

Michal Kelemen, Prof. Ing. PhD.
Technical University of Kosice, Faculty of Mechanical Engineering
Institute of Automation, Mechatronics, Robotics and Production Techniques
Letna 9, 04200 Kosice, Slovak Republic
michal.kelemen@tuke.sk