Advanced control methods of robotic arm with computer vision integration

Peter Papcun
Technical University of Košice
Faculty of Electrical Engineering and Informatics
Department of Cybernetics and Artificial Intelligence
Košice, Slovakia
peter.papcun@tuke.sk

Oliver Kudzia
Technical University of Košice
Faculty of Electrical Engineering and Informatics
Department of Cybernetics and Artificial Intelligence
Košice, Slovakia
oliver.kudzia@student.tuke.sk

***Abstract*—The aim of this article is to model, physically construct and program a six-axis robotic arm capable of grasping objects based on visual feedback from a computer vision system using a Kinect sensor. This article focuses on the theory behind robotic arm control as well as what kind of actuators are typically used and also on object recognition analysis using depth maps. Next it covers the construction aspects of the arm, describes the software part of the arm control and communication architecture. The object recognition algorithm and its integration into the arm's control system is also described. This article includes visualization and a graphical user interface, which acts like a controller for the robot. Finally, the achieved results based on experiments are presented in evaluation section.**

***Index Terms*—robotic arm, kinematics, control, computer vision, Kinect**

I. Introduction

With the increasing power of computational technology, the boundaries of its applicability have significantly shifted. Gradual progress in the field of microchips has allowed for the creation of new sensors, including camera systems. Nowadays, such systems can be found in various industries - they can be used in autonomous vehicles to detect other participants in traffic or in manufacturing for quality control, sorting products or other tasks. This technology can also greatly expand the capabilities of robotics, as it mimics the optical system of humans and receives visual stimuli from the surrounding environment.

This work builds on the foundations of the bachelor's thesis *Design and realization of the robotic arm's model with base control* [1] and continues in the field of robotic arms and their control. The aim is to create a six degree of freedom robotic arm with stepper motor actuators and expand its control with computer vision integration. The communication between the arm and the computer vision algorithm will be mediated by a client-server architecture. The objects placed on workplane will be recognized using a depth map extracted from the depth camera provided by the Microsoft Kinect V2 sensor. The main purpose is to use this information to grab and manipulate with objects in pre-defined space. The arm will perform a series of actions by moving the detected objects to a storage area. After satisfying certain conditions it will assemble a simple three-component part. Besides creating a physical model of the robotic arm, the work also includes its virtual representation in the form of visualization and a graphical user interface with elements for controlling both the virtual and physical models.

II. Analysis of Kinematics

From a control perspective, it is necessary to initially characterize a robot with parameters that allow working with its mathematical model. Generally, the parametric description of a robot is called kinematic structure, providing values for translational and rotational components. Combining these components leads to the creation of a kinematic chain. As stated in [2], robots are classified based on their kinematic structure to:

- serial robots - an open kinematic chain (*open loop*), such as a robotic arm,
- parallel robots - a closed kinematic chain (*closed loop*), such as a Stewart platform,
- hybrid robots - a combination of both types of chains.

Another essential criteria in robot classification is the number of degrees of freedom. The degree of freedom is defined as the number of parameters needed to specify the mechanism's configuration concerning the number of links, joints, and mobility at each joint [3]. In the context of industrial robotic arms, this concept describes a rotational joint with a single axis of rotation. According to [2], robots are categorized based on the number of degrees of freedom to:

- universal robots - with exactly six degrees of freedom, uniquely defining the position and orientation of the object's manipulation in a Cartesian coordinate system,
- redundant robots - with more than six degrees of freedom, much more flexibility, making it capable of moving in confined space,
- deficient robots - with less than six degrees of freedom, typically SCARA robots.

From these concepts, it is evident that an industrial robotic arm is classified as a serial universal robot. As mentioned earlier, industrial robots use only two basic types of kinematic pairs in their kinematic chains - rotational (R) and translational (T). Based on the axis of the coordinate system, rotations are denoted by R_x, R_y, R_z, and translations by T_x, T_y,

T_z respectively. To fully determine any point in space, six independent coordinates in total are required.

A. Forward and Inverse Kinematics

An important part of the analysis of robotic arm kinematics is to know the complete kinematic model of the mechanical system, which provides all the essential quantities for both the dynamic model and the control of the robotic system. This mainly involves the position and orientation of the end effector given time t and the corresponding movement of the individual links of the entire mechanism. The position of these links is generally described by coordinates. In robotics, the term *joint variables* is often used to indicate the rotation or translation of individual motion axes. Within kinematics, two important concepts are introduced – forward and inverse kinematics. Figure 1 illustrates the relationship between forward and inverse kinematics. Angles $\alpha, \beta, \gamma, \delta, \epsilon...$ represent all the rotations of

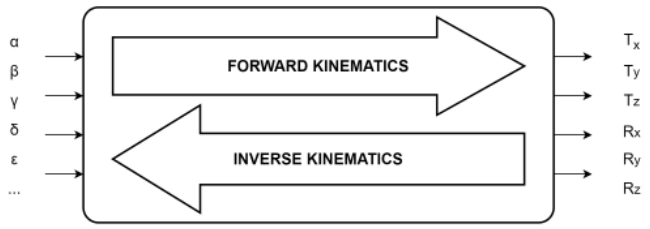

Fig. 1. Schematic representation of forward and inverse kinematics

n joints within the kinematic chain. Coordinates T_x, T_y, T_z and R_x, R_y, R_z represent the final position and orientation of the end effector in a Cartesian coordinate system.

Forward kinematics deals with computing the end effector coordinates of the kinematic chain based on its joint rotations. For example a robotic manipulator consists of serial links, which are attached to each other by rotational or prismatic joints from their base up to the end effector. To systematically solve the forward kinematic task of the robot, an appropriate kinematic model [4] must be used. Each rotation and translation can be mathematically interpreted as a transformation matrix and expresses one coordinate system. Multiplying these matrices in series results in a homogeneous composite transformation matrix. The calculated matrix contains information about the end effector position in space. However, an additional calculation, further explained in [1], is required to determine the orientation of this point.

Conversely, the problem of inverse kinematics is substantially more complex and has been studied for many decades. It is necessary for the control of industrial robotic arms and its solution is computationally expensive. The definition of the inverse kinematic task is opposite to forward kinematics. It refers to the backward process calculating the parameters of joint rotations based on the specified position and orientation of the end effector. The tasks to be performed by the robotic manipulator are defined in Cartesian space, while the actuators operate in so-called joint space. The conversion of the position and orientation of the manipulator's end effector from Cartesian space to joint space is called the inverse kinematic problem. There are two main approaches for solving it – geometric and algebraic – which are used for the analytical derivation of the inverse kinematic solution. [4]

However, inverse kinematic solutions have certain disadvantages, such as non-existence of solutions or multiple solutions, which results in higher computational complexity when the number of joints in the kinematic structure increases. For this reason a lot of research currently attempts to find new methods to solve the problem of inverse kinematics with reduced computational complexity, faster, and more accurately. The most commonly used methods for solving inverse kinematics can be classified into three categories – algebraic, geometric, and iterative methods. These methods are further explained in [4], [5] and [1].

B. Actuators

Each controlled axis of a robotic system requires its own actuation, which must provide precise and slow, or fast movements with sufficient acceleration and deceleration. The drive unit is typically referred to as an actuator and belongs to the set of components that act upon the system. Electric drive is the most commonly used type of drive in practice and is most suitable for robots that do not only require high speeds and power, but also precision and repeatability. Their use in this sector is particularly interesting due to their simple integration into the system as well as their control and reliability. Electric drives are classified as motors or servo motors powered by direct or alternating current and stepper motors.

The most popular motor is the stepper motor, which provides high accuracy and good controllability. It works with individual discrete states called steps. As stated in [6], electromagnetic stepper motors are multi-polar and multi-phase synchronous motors adapted to operate in a stepping mode. They are most commonly used as open-loop positional digital servo drives, without direct sensing of the rotor position of the motor. In terms of coil winding, there are unipolar and bipolar motors. The principle of motor operation involves gradually switching current impulses to the stator coils, creating a rotational magnetic field. The frequency of coil switching determines the rotational speed of the magnetic field, which is directly proportional to the speed of the motor. In robot motion control systems, stepper motors are typically part of a closed loop system, where feedback is provided through the sensor response from an encoder located inside the joint.

A significant part of the robotics joint actuation solution are gearboxes. In practice, there are various types of gearboxes, such as spur gears, worm gears, tooth belts or chains. Servo drive speeds are reduced through special robust gearboxes with high reduction ratios and operating without backlash. High reduction ratios make it possible to achieve larger torques required for the movements of robotic arms. [7]

III. Object Recognition Analysis

Besides to conventional color cameras (similar to human vision), there is a special kind able to provide information

about the distance between the camera and the environment – a depth camera. It allows to obtain data from an additional dimension, giving it an advantage especially in more complex tasks where a regular camera system is no longer sufficient. For example in mobile robotics it is used for determining distances of objects surrounding the robot for the purpose of obstacle avoidance or for mapping the environment. It can be also applied in object detection and recognition.

A. Microsoft Kinect V2

In 2010 Microsoft introduced a new type of sensor. The Kinect was designed to serve as an interactive controller for Xbox consoles. It consists of a camera system composed of a color RGB camera and an infrared laser projector. A newer version of the Kinect sensor (labeled as Kinect V2) released in 2012 uses Time-of-Flight technology to capture depth information. The working principle is based on emitting infrared light beams into the environment and measuring the time it takes for them to be reflected back to projector. [8]

TABLE I
TECHNICAL SPECIFICATIONS OF KINECT SENSORS

Parameter	**Kinect V1**	**Kinect V2**
RGB camera	640×480 px	1920×1080 px
Frame rate	30 fps	30 fps
Depth/IR camera	320×240 px	512×424 px
Max distance	~ 4500 mm	~ 4500 mm
Min distance	400 mm	500 mm
Principle	*Structured Light*	*Time-of-flight*

The technical specifications of both versions of the Kinect sensor are compared in Table I. There are many studies describing the differences and comparisons between these two versions. For example, [9] further discusses the use of this technology in robotics and computer vision.

In order to use this sensor in a computer vision application, it is necessary to process image information on a computer. Since Kinect is primarily a peripheral for Xbox consoles, Microsoft has introduced a special adapter to connect Kinect sensor to a computer running Windows operating system on it via USB 3.0. Working with data gathered from sensor is possible in multiple programming languages with appropriate libraries installed.

B. Depth Map

A depth map is represented by a two-dimensional matrix of distances between the depth camera sensor and the surface. The Kinect sensor measures these distances in millimeters, with values ranging from a range of $400mm \leq d \leq 4500mm$. However, the acquired depth map is generated by projecting surface points only and therefore does not obtain full volumetric data of the scenery. As stated in [10], information from the depth map can also be accessed through projection into a point cloud. This approach allows, among the other things, potential visualization of data in the form of isolated points in 3D space. As the Kinect captures RGBD data (three color channels for red, green, and blue and one channel for depth), it also allows assign a color value for each point. The advantage of transforming a depth map into a point cloud is its applicability in algorithms like 3D SIFT for keypoint detection and descriptor extraction. These kind of methods are particularly useful for advanced computer vision tasks.

C. Image processing methods

In general every image information obtained through cameras needs to be pre-processed. From an information theory perspective, the pre-processing does not result in any loss of information, it can only be highlighted or suppressed. This is mainly due to the fact that the captured image contains redundant data, which can be reduced using certain methods. The most common image disturbances are noise, distortions caused by the properties of the imaging device (correction is done using affine transformations) [11], and bad ratios of brightness and contrast. Various types of methods or algorithms from the computer vision domain can be applied later after pre-processing.

D. Object recognition

By combining multiple image processing methods we can obtain an algorithm capable of object recognition. The task is to identify for example the type of object. Using basic image processing methods, it is able to convert the full-color image into a grayscale image. That way, we can handle data with better computation efficiency. Then, by applying operations such as Gaussian blur (which involves convolution) or edge detection (e.g. Canny edge detector) the resulting image is prepared to be used for specific computer vision methods. For example, the Hough transform has the capability to identify either lines or circles in the image. All methods contribute to the extraction of data which may be useful in solving the given problem. A similar approach can be used to evaluate depth maps. The only difference is that now we also have information about depth of the image. For a better illustration, the parallel between a depth map (in this case represented as a point cloud) and a grayscale image is illustrated in Figure 2. Objects protruding from a homogeneous surface plane (shown

Fig. 2. Transformation of depth map to grayscale image

in dark red color in a point cloud) have lower pixel brightness values than their surroundings. This way, a grayscale image suitable for pre-processing by conventional computer vision methods can be obtained. The advantage of this solution is the gathering more information to support recognition.

IV. Design and Realization of Robotic Arm

The process of developing a robotic arm consisted of several steps. At the beginning, a full design of the robotic arm model was created, focusing on its construction aspects, drive units and sensors. The design of individual parts was made in CAD software Fusion 360. The arm's actuators are NEMA stepper motors and microswitches implemented in each joint are used for arm centering. The development of the robot components started with the design of its base. The base is the most critical part because the entire weight of the arm rests on it. As for the next part, standard industrial GX-12 connectors are located at the rear of the base, providing I/O interface for sensors and actuators. A NEMA 23 stepper motor is propulsing first joint of the arm, providing sufficient rotation torque. The same type of stepper motor also drives the second joint, which is located above the base and must also provide sufficient torque. A NEMA 17 motor is located in the middle of the robotic arm. The top part of the robot contains the last three joints driven by small NEMA 14 motors. Microswitches inside each joint are used to detect certain position and together with motors they form an open-loop system similar to a 3D printer. This microswitch system is later used in centering procedure of the arm. A gripping mechanism is also present as the end effector of the arm. As a robotic gripper we used the one used in open-source project of the company BCN3D [15]. The power supply and signals for gripper are transmitted by a cable with special connector located at the top part of the robot. The final model of the robotic arm is shown in Figure 3. The designed parts were physically manufactured using 3D printing technology. Later they were assembled using standard fastener materials (screws and nuts) of the DIN 912 and DIN 934 standards. As shown in the figure, a two-color combination of blue and black was chosen.

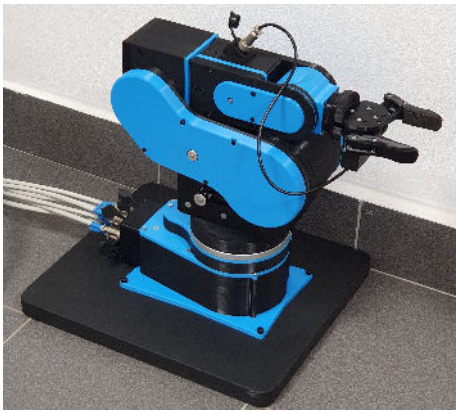

Fig. 3. Physical model of the robotic arm

It was also necessary to resolve how the arm should be controlled. Given that specialized control units are used in practice to control industrial robotic arms [12], we took inspiration from this approach. All key control elements, such as stepper motor drivers, main control unit, single-board computer Raspberry Pi and power supply are stored in one place. This mediate the transfer of important electrical signals between the control system and the arm. We have created a decentralized control architecture that combines a local control box with a laptop via the TCP protocol. A summary diagram of the control architecture is shown in Figure 4 with symbolic representations of the communication flows through arrows. The block structure divided into several parts expresses specific topological connections of control programs and their relationships. The control is divided into a part with

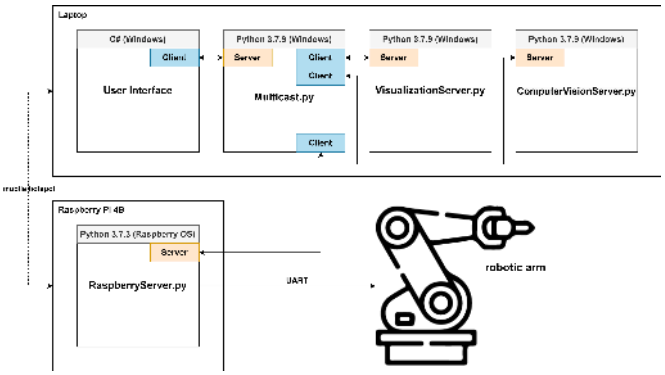

Fig. 4. Communication and control architecture

calculations performed on the laptop and a part with control procedures launched on local devices inside the control box. Communication between the laptop and the Raspberry Pi is done by a client-server architecture via mobile access point.

Core of the control program consists of five applications:

1) The *WPF GUI* application is a graphical user interface made using the Windows Presentation Foundation [17].
2) The *VisualizationServer* application handles the visualization of the robotic arm using the Open3D library [16].
3) The *ComputerVisionServer* application is responsible for obtaining the depth map from the Kinect sensor, preprocessing it, detecting and identifying the topmost object, and interpreting it into a four-value data structure.
4) The *RasbperryServer* application runs on a single-board computer Raspberry Pi inside the control box, after it receives data from the laptop it transmits them via UART serial communication to the main control unit (custom board based on STM32 microchip).
5) The *Multicast* application serves as a communication mediator between the graphical interface and all other server applications.

Each application performs a specific task in whole control process. The kinematic calculations for the robotic arm are computed on the laptop. Similarly, arm visualization representing a virtual twin is running on the same computer together with the graphical user interface for manual arm control. Also, the computer vision module is operated through this user interface. The Table II lists the IP addresses and corresponding ports of each application. From the control architecture diagram, it follows that the graphical interface connects to the *Multicast* server as a client application (the last row in the table) and communicates with the other applications.

TABLE II
LIST OF IP CONFIGURATIONS FOR APPLICATIONS

Application	IP adress	Port
VisualizationServer	127.0.0.1	27001
ComputerVisionServer	127.0.0.1	27002
RaspberryServer	192.168.137.94	27003
Multicast	127.0.0.1	27000

The transmitted data is encoded in the JSON format, which has been chosen for its convenient manipulation in various programming languages.

V. COMPUTER VISION INTEGRATION

After creating the robotic arm, we moved on to designing a computer vision module and integrating it into the control system. Initially, the location of the Kinect had to be determined. We created a stand from aluminum profiles. Total workspace created beneath the camera system has dimensions of $610mm \times 350mm \times 750mm$, providing enough clearance for the robotic arm to perform its tasks. Optimal sizes and shapes of the objects the robotic arm will manipulate were also chosen during this process. We decided to use three types of objects:

- **box** – a block with a square-shaped plan with a cylindrical opening in the upper part,
- **cylinder** – an object with circle-shaped plan,
- **toroid** – a ring with an annulus-shaped plan.

Combination of these bodies creates simple three-component part symbolizing a metal base (box) with an axis (cylinder) and a bearing (toroid).

The next step was to design the individual parts of the computer vision system. This includes the method of extracting the depth map, its preprocessing, object recognition, and subsequent use of the obtained information in the arm control process. This way, the arm is able to respond to visual stimuli in the form of object recognition and subsequent manipulation of the object to the storage region. The depth map is obtained from the Kinect sensor using a special adapter with a USB cable connected to a desktop computer. At the beginning of the recognition algorithm, an initial sensor calibration is performed to determine the equation of the workspace plane. For this purpose, the RANSAC method [13] is used to determine the distance between the sensor and the plane. The task of the application *ComputerVisionServer.py* running on a laptop is to capture the depth map, preprocess it, and then detect and identify the topmost object if a request is received from the client. The coordinates of found object are remapped from the image coordinate system to the robotic arm coordinate system before being sent. The output from the computer vision system is a four-value data structure, which includes three Cartesian coordinates of the object x, y, z defined as floats and the type of the object defined as a string (for example $300, -15, 64$, "cylinder"). The entire structure of the algorithm is schematically shown as a simple block diagram in Figure 5.

The final stage of the entire algorithm is to identify the type of the topmost detected object. The recognition solution

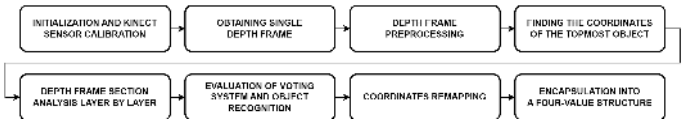

Fig. 5. Simple block diagram of the algorithm

is inspired by the voting system used in the Hough transform [14]. We simplified this idea and adapted it to the needs of our program. The voting process involves a step-by-step search and analysis of each layer of the depth map (Figure 6). Starting from the highest detected point, we move downwards by n layers, as shown in the figure. In each layer, a certain series of operations is performed, which contributes to the voting system. These operations include counting the number of blobs, finding line segments as well as their lengths and counting the number of detected circles and their radii. Voting

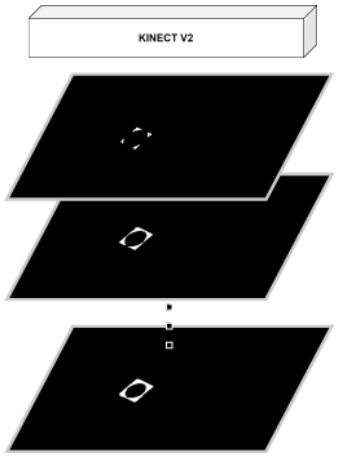

Fig. 6. Cross-section analysis of a depth map used in the algorithm

is done by gradually contributing heuristically chosen floating point numbers within the range $\langle -5.0;\ 5.0 \rangle$ to each of the three object types based on certain pre-defined conditions preprogrammed into the system. The object with the highest accumulated voting number determines the final recognition outcome.

VI. VISUALIZATION AND USER INTERFACE

As part of the solution of robotic arm control system, a visualization and a graphical user interface were created. The visualization software allows to test kinematic calculations and verify both forward and inverse kinematics of the arm. As an example, graphic elements for manual control of forward kinematics are displayed in Figure 7. Additionally, it protects the construction of the physical model from damage caused by illegal kinematic poses. Arbitrary given robot kinematic position can be at first tested virtually and then can be sent to physical model of the arm. The user interface acts as a central controller for the robotic arm with the ability to interactively work with both the virtual and physical models. It also contains control elements for the computer vision module to recognize, manipulate and assemble three-component part.

User interface is divided into three main tabs in the top panel of the window. Tab named *Model* is the most important part of the interface. It consists of a few tabs located in the left panel – Forward Kinematics, Inverse Kinematics, Computer Vision, Visualisation Mesh, Connection and Applications. The functionality of each tab is described by its name. The startup procedure is simple and consists of three operations. The first step is to run each application server from the *Applications* menu. A command-line shell for each application is automatically opened in the background. Within the command-line an appropriate Python script is executed. On

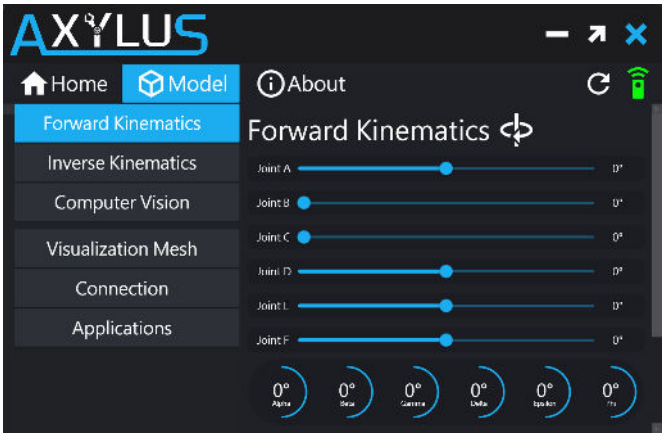

Fig. 7. Graphical user interface for robotic arm

the Raspberry Pi computer this step must be done manually by running the *RaspberryServer.py* application. The order in which the *VisualizationServer.py*, *ComputerVisionServer.py*, and *RaspberryServer.py* applications are launched does not matter. However, it is important to start *Multicast.py* as the last application due to the communication architecture. All servers are interconnected through the *Multicast.py*, allowing the GUI to communicate with them. The connection is established by entering the appropriate IP address and port in the *Connection* GUI section. A green indicator in the upper right corner of the window informs the user of a successful connection.

VII. Evaluation of the Experiments

In the final process we verified individual parts of the robotic system. Achieving the given task required a series of steps which are hierarchically interconnected. The assembly of the three-component part is conditioned by the correct arrangement and number of objects within the storage area. The positions of these objects on the surface are conditioned by the proper gripping and manipulation with detected object, which relates to the kinematic precision of the robotic arm. In order to ensure an incremental form of testing, the evaluation of the solution was divided into three phases. Firstly, the robotic arm and its ability to grip objects were tested. Secondly, the reliability of the recognition algorithm was evaluated. Finally, the success of assembling a three-component part was experimentally verified.

A. Kinematic Precision Testing

The testing of the robotic arm was performed by measuring the precision of the end effector position. This determined the ability of inverse kinematics to acquire the given point with a certain accuracy. Experimental verification was carried out by randomly generating a set of points within a defined space in front of the arm. The generated values were for the x-axis in the range of $x \in \langle 180.0;\ 400.0; \rangle$, for the y-axis in the range of $y \in \langle -270.0;\ 270.0; \rangle$, and for the z-axis in the range of $z \in \langle 0.0;\ 500.0; \rangle$. We performed 8 independent experiments using the Euclidean distance metrics between the given point and the actual point acquired by the end effector of the manipulator. The measurements showed that the average deviation between the expected and actual point was approximately $17mm$.

B. Object Grip and Manipulation Testing

Further testing of the robotic arm focused on grasping and moving with objects, thus verifying the arm's ability to pick up objects from a pile located in front of the robot. The geometric properties of each object satisfies the requirements for gripping with the gripper mechanism. Also, the weight of the objects are appropriate for this task. A total of 24 independent attempts were performed – 8 attempts for each type of object. We found out that the box did not cause any significant problems in the gripping procedure. The only problematic cases were when the orientation of the box was at an angle of around $45° \pm 5°$ with respect to the reference coordinate frame of the working plane.

C. Object Recognition Evaluation

The process of experimental verification of recognizing the topmost object found in workspace was carried out by randomly placing a varying number of objects in front of the robot. A total of 30 attempts were made, with 10 attempts reserved for each type of object. As each recognition test was conducted with a different number of objects and variable density of their arrangement, we also examined possible effects on our recognition algorithm. No correlation was found between the number of objects and the results, because the algorithm is only set to search for and identify the highest object in the heap. However, some problems arose with objects of similar height and close proximity to each other. The layered scanning implemented in the algorithm captured lines and circles from the vicinity of the highest point, which led to incorrect identification of object type. We summarized the results in the confusion matrix. Analysis of the matrix revealed that the cylinder had the least stable recognition success rate. Categorization of the box showed the best results with 90% accuracy. The cylinder object was successfully recognized in 60% of cases due to the similar radius of the circle found in the box or toroid. The toroid had a higher, 70% success rate, and the algorithm confused this object with the box in 30% of cases.

D. Part Assembly Experiments

In order to evaluate the success of the assembly in some way, we chose a quantitative approach to test this process. We created Table III with a set of three columns representing the number of complete, partial and failed assemblies. After conducting 20 attempts, we recorded the results in the table according to the outcome of each attempt. Results provided in

TABLE III
SUCCESS TABLE OF PART ASSEMBLY

Complete assemblies	Partial assemblies	Failed assemblies
12	4	4

the table shows that the robotic arm was able to successfully assemble the part with a 60% success rate, while the remaining 40% of all attempted trials resulted in the arm assembling the part either partially or unsuccessfully. The achieved results of the robotic system can be improved with sufficient time for fine-tuning critical segments of the robotic system, such as a more precise recognition algorithm or the upgrade of objects to weight more.

VIII. CONCLUSION

The goal was to design, model, and construct a physical model of a robotic arm, program its control, and expand it to work with image information obtained from a Kinect sensor. The combination of the arm and the camera resulted in a robotic system capable of grasping objects and assembling them into a simple three-component assembly. The entire project can be summarized in several stages. Initially, it was necessary to consider what the robotic arm would look like and what tools would be used in the process of physically constructing the robot. We also became familiar with the Kinect sensor, using the programming language Python to communicate with it. Initial tests were performed to obtain a depth map from the sensor, and an object recognition algorithm was devised. Due to the fact that the robotic arm and computer vision system can work separately, it was necessary to properly connect these units. This stage involved the development of control infrastructure for the robotic system. We came up with a decentralized architecture where part of the operations are performed on a laptop, and a portion is executed locally in the robot's control unit. The entire control system uses a client-server architecture and facilitates data exchange between the software running on the laptop and the control program in the box. Additionally, we created a virtual representation of the arm in the form of visualization and a graphical user interface that provides interactive control of the robot without the presence of its physical form. The interface offers the possibility of manual mode for controlling kinematics and gripping mechanisms. Control with computer vision integration is available to the user in a semi-automatic mode with sequential command triggering. After recognizing an object and moving it to a storage surface via the robotic arm, it is checked whether a simple three-component assembly can be assembled from the recognized objects. If this condition is met, the assembly is assembled without any necessary user intervention.

REFERENCES

[1] Kudzia, O. Návrh a realizácia modelu robotického ramena so základným riadením. *Technická Univerzita V Košiciach, Bakalárska Práca*. (2021)

[2] Skařupa, J. Prumyslové roboty a manipulátory. (VŠB-Technická univerzita,2008)

[3] Pennestri, E., Cavacece, M. & Vita, L. On the computation of degrees-of-freedom: a didactic perspective. *International Design Engineering Technical Conferences And Computers And Information In Engineering Conference*. **47438** pp. 1733-1741 (2005)

[4] Kucuk, S. & Bingul, Z. Robot kinematics: Forward and Inverse kinematics. (INTECH Open Access Publisher,2006)

[5] Dahari, M. & Tan, J. Forward and inverse kinematics model for robotic welding process using KR-16KS KUKA robot. *2011 Fourth International Conference On Modeling, Simulation And Applied Optimization*. pp. 1-6 (2011)

[6] Žalman, M. Akčné členy. (Slovenská technická univerzita v Bratislave, Fakulta elektrotechniky a informatiky,2002)

[7] Levice, S. Základné moduly robota – Pohony. (2015), https://www.spslevice.sk/ucebnice/SOC/SOC%20-%20PRI/107-Zakladne_moduly_robota.htm, [online]. cit. 2023-08-03

[8] Le, T. & Lin, C. Color and depth mapping of Kinect v2. *The 16th International Conference On Automation Technology*. pp. 208-213 (2019)

[9] Wasenmüller, O. & Stricker, D. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. *Computer Vision–ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II 13*. pp. 34-45 (2017)

[10] Šikudová, E., Černeková, Z., Benešová, W., Haladová, Z. & Kučerová, J. Počítačové videnie. *Detekcia A Rozpoznávanie Objektov*. pp. 397 (2013)

[11] Bieniecki, W., Grabowski, S. & Rozenberg, W. Image preprocessing for improving ocr accuracy. *2007 International Conference On Perspective Technologies And Methods In MEMS Design*. pp. 75-80 (2007)

[12] AG, K. KUKA KR C4. (2023), https://www.kuka.com/en-us/products/robotics-systems/robot-controllers/kr-c4, [online]. cit. 2023-23-03

[13] Fischler, M. & Bolles, R. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications Of The ACM*. **24**, 381-395 (1981)

[14] Duda, R. & Hart, P. Use of the Hough transformation to detect lines and curves in pictures. *Communications Of The ACM*. **15**, 11-15 (1972)

[15] BCN3D BCN3D MOVEO: A Fully Open Source 3D Printed Robot Arm. (2016), https://www.bcn3d.com/bcn3d-moveo-the-future-of-learning-robotic-arm/, [online]. cit. 2023-21-03

[16] Zhou, Q., Park, J. & Koltun, V. Open3D: A Modern Library for 3D Data Processing. *ArXiv:1801.09847*. (2018)

[17] Microsoft What is Windows Presentation Foundation - WPF .NET. (2023), https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0, [online]. cit. 2023-05-04