Real Time Computer Vision Based Robotic Arm Controller with ROS and Gazebo Simulation Environment

Egemen Aksoy*,[1], Arif Dorukan Çakır*,[1], Berat Alper Erol[2], and Abdurrahman Gumus[1]

[1]Electrical and Electronics Engineering, İzmir Institute of Technology, Türkiye
[2]Computer Engineering, İzmir Institute of Technology, Türkiye
egemenaksoy35@gmail.com, arifdorukan@gmail.com, beraterol@iyte.edu.tr, abdurrahmangumus@iyte.edu.tr
*Contributed Equally

Abstract

Robotic arms are widely prevalent and find utility in a variety of applications. However, a significant and widespread challenge faced by these arms is their inability to replicate the intricate functionalities of a human hand, primarily due to the distinct structure of the hand. The primary objective of this work is to create a simulation of a robotic arm that can replicate the movements and functions of a human hand in real-time. Data obtained from hand and arm gestures created with Mediapipe will be transferred in real-time to the robotic arm that is visualized and simulated on ROS and Gazebo. Thus, the hand and arm movements of the user in front of the camera will be effective in real-time on the manipulable joints. This advancement holds the potential to facilitate the construction of a robot capable of emulating both the hand and the arm of humans with high fidelity, thereby enabling comprehensive control over the robotic arm's actions in real-time.

1. Introduction

In various professions, daily work routines expose individuals to hazards that can jeopardize their well-being. A notable example is the field of chemistry, where certain experiments yield toxic liquids and gasses that can harm health through inhalation or skin contact. Similarly, precision is vital in delicate surgeries, with even millimeters holding significance. While robots like DaVinci are designed for surgical precision, an arm emulating human movements and adeptly handling hand vibrations could substantially enhance surgical outcomes [1]. Hence, the idea of simulating a robotic arm to achieve human-like control has gained traction. This innovation holds promise in safeguarding health and elevating success rates in critical applications. While there are robots and studies in these fields in the current literature, the idea of a humanoid robot that directly mimics the human hand and arm has not yet been realized. The aim of this project is to simulate this robot.

There are several studies in the literature implementing a robotic system control by interacting with a physical system such as the human body, hand or arm [2–4]. Numerous projects in the field employ computer vision tools for the manipulation of robotic arms; however, these endeavors are primarily limited to executing basic predetermined instructions, such as merely opening and closing specified fingers [2]. Apart from robotic arms, works have also been carried out in which various robots that look like autonomous vehicles can be controlled using only hand gestures. In such projects, the aim is to move the robot in 2 dimensions by using hand movements [3,4]. However, these systems are limited to XY coordinates while giving the control commands. In contrast, the implementation of this study's control approach enables the precise manipulation of joint positions and the emulation of user hand movements. In this way, the simulated robotic arm is allowed to move in three dimensions.

In this study, the goal is to simulate a robotic arm controlled by human hand gestures. The developed system was characterized by precise gestures beyond open-and-close movements, resulting in an enhanced ease of manipulation suitable for a variety of applications. To develop this system, some tools such as Mediapipe for detecting landmarks of the arm and hand using computer vision, ROS (Robot Operating System), and Gazebo for simulation of the designed robot were used.

The remainder of the paper is organized as follows: in Section 2, used methods are explained; in Section 3, results are given and discussions are made; in Section 4, conclusions are given; in Section 5, references are explained.

2. Methods

The article discusses the simulation and control of a robotic arm within the scope of this project. The control part is further divided into two sections as manual command-based control and controller-based control. As seen in the flowchart in Fig. 1, initially, simulation was conducted, followed by manual control through commands. In the final stage, a controller was added to enable real-time control of the robot simulation.

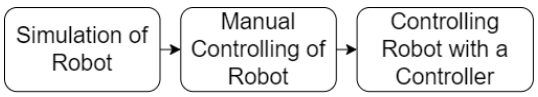

Fig. 1. Block diagram of the project steps

2.1. Preliminaries

To develop the system, some tools such as Mediapipe, ROS, rospy, and Gazebo were used. Mediapipe, an open-source firmware developed by Google, is utilized to facilitate the creation of machine-learning solutions. Its modularity and cross-platform capabilities allow for a swift and straightforward implementation. Multiple operations across domains, including machine learning and image processing, are consolidated within this comprehensive package. The Mediapipe firmware is employed to detect hand and arm landmarks, with implementation carried out in Python3. Additionally, camera image access and manipulation are facilitated through the OpenCV library.

ROS (Robot Operating System) is an open-source middleware framework used to build, develop and control robotic systems on Linux [5]. It is essential for processing 3D location data obtained from Mediapipe. The framework necessitates the installation of specific files to achieve desired results, extending beyond the confines of a single software file. Communication between ROS and Python is enabled through the rospy library, functioning as an API that supports data exchange and data processing, including tasks related to sensor applications and computer vision processes.

The specifics of the robot's design are captured through URDF (Unified Robotics Description Format), an XML (Extensible Markup Language) format tailored for intricate robot modeling down to the component level. To manage complexity, Xacro (XML Macro) files streamline the description of robots, making maintenance and readability more accessible. Within the realm of ROS, the concept of the "world" serves as the immutable foundation upon which the robot operates. "Links," representing individual components, define the robot's shape, while "joints" enable movement, orchestrating dynamic motion sequences [6].

Gazebo, the selected simulation program running on Linux, converts ROS-generated files into 3D simulations that accommodate not only robot simulations but also map design [7]. Gazebo functions as a real-world physics simulator, utilizing physics engines to significantly enhance the realism of the simulation and the accuracy of conclusions that can be drawn in possible implementations [8]. The synthesis of these elements culminates in the creation of an XML file for Gazebo, defining each link to instantiate the robot's presence within the simulation environment. The simulation and computer vision-based robotic arm controller executed in a PC that has Intel Coffee Lake Core i7-8750H CPU, 4GB GDDR5 Nvidia GTX1050Ti 128 Bit GPU hardware. The simulation and controller performance is around 20-30 fps and the latency is 10ms.

2.2. Simulation

The initial phase of the project involves the creation of a simulation for the robotic arm to be controlled. Within the simulation phase, there are four main files required, as illustrated in the flowchart in Fig. 2. These files are material, definition, reference, and launch files.

The Material file plays a pivotal role in specifying RGB color definitions for the model, a critical aspect for ensuring visibility within Gazebo. Without this file or complete color definitions, the robot remains imperceptible, even when devoid of errors.

The comprehensive robot description is housed within the definition file, encapsulating definitions for the world, links, and joints. The initial step involves creating the world, which functions as a fixed joint for defining the ground. Subsequently, the construction of the robot model unfolds. The robot employed in this project comprises 23 links, each meticulously defined with inertial, collision, and visual attributes, characterized by their individual origins and sizes.

Additionally, the project encompasses 23 joints, including both fixed and revolute types, facilitating the robot's dynamic movement. These joints originate from parent links and connect with child links, equipped with attributes such as origins, adjustable parameters like damping coefficients, and axes.

Upon the successful completion of all definitions without errors, the robot model becomes fully operational within Gazebo. However, Gazebo faces a challenge in interpreting these existing

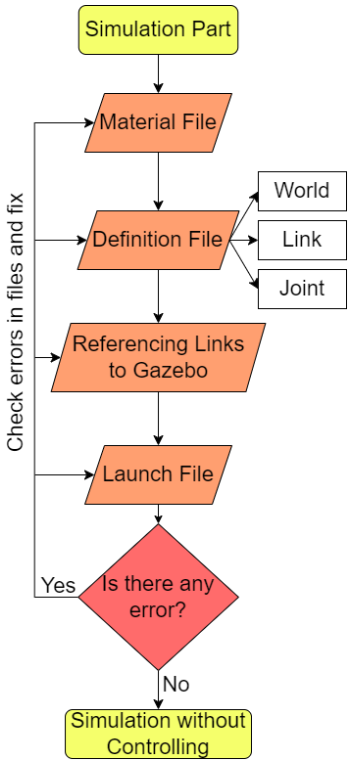

Fig. 2. Flowchart of the simulation part involving four main files as material, definition, reference, and launch files

definitions. To address this, a dedicated reference file tailored specifically for Gazebo is introduced, referencing all the links from the primary file. Furthermore, Gazebo requires the presence of a launch file, leading to the introduction of a specialized launch file designed for Gazebo, thereby enabling the simulation of the robot model by executing the file in the terminal.

2.3. Control

The second phase of the project involves controlling the robotic arm that has been created in a simulation environment for control.

As seen in the flowchart in Fig. 3, there are two distinct methods of control: control with commands and control with controller. In command-based control, manual data input from the terminal is used to test the system's ability to subscribe data to the motor joints.

In controller-based control, data obtained using computer vision by processing hand and arm movements acquired by the camera are used as input. In this section, in addition to the involved Python code, an additional file is required for data acquisition in Gazebo, and a launch file is needed to execute the acquired control data.

In a comprehensive and flawless simulation procedure, the manipulation of the joints can be accomplished through the panels accessible in Gazebo. However, in order to enable control via computer vision instead of manual manipulation, an additional step is required. This involves designating the joints as motor joints within the main definition file. This adjustment is essential for granting controllability to the joints.

The definitions applied alone are insufficient for establishing computer vision-based control in Gazebo. Unlike the simulation

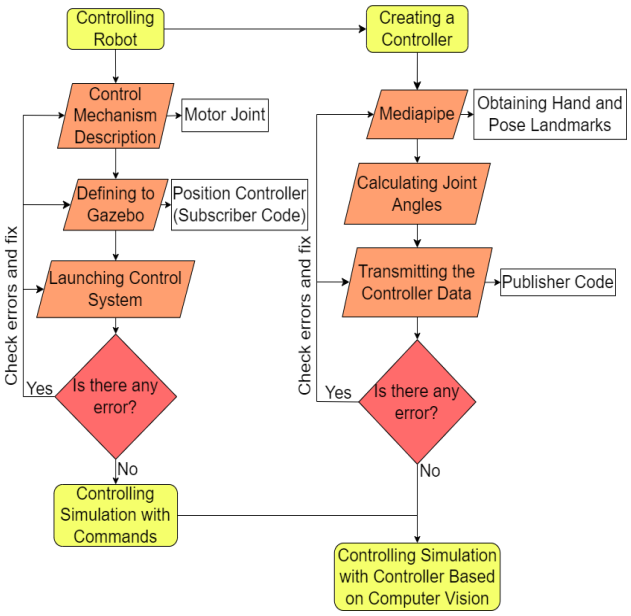

Fig. 3. Flowchart of the controlling part

phase, Gazebo doesn't inherently recognize that these joints can be manipulated solely based on their definitions. As a solution, a new file containing a subscriber code must be generated wherein nodes are created within Gazebo, and these nodes are then linked to the designated motor joints [9]. This code is also called a position controller.

2.4. Joint Angle Calculation

The subsequent phase involves incorporating computer vision algorithms for controlling the robotic arm and calculating the joint angles within the project. The data obtained from Mediapipe was initially categorized into three groups: right hand, left hand, and pose [10, 11]. Subsequently, it was further dissected based on the landmark numbers and their respective positional axes. Fig. 4 illustrates the landmark map used as a reference during this parsing process.

To calculate the joint angles for each particular joint, the distances between specific landmarks are computed. The algorithmic representation of this calculation process is depicted in Algorithm 1 as a pseudo-code.

As explained in Algorithm 1, the distance between the fingertips and wrist landmarks is calculated to determine the desired joint angle position data. To mitigate errors and undesired movements caused by variations in the hand-arm distance from the camera, a ratio is computed by dividing the user's forearm length by the calculated landmark length during each operation cycle.

Additionally, an offset is assigned based on the length results obtained from multiple experimental tests. Subsequently, this offset value is amplified at a specific rate, leading to the derivation of the desired manipulator angle values. These calculations are recurrently performed in each cycle, exclusively when the relevant landmarks associated with that specific movement are detected.

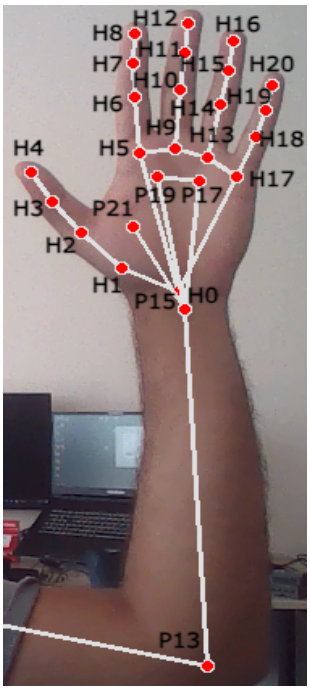

Fig. 4. Hand and pose landmarks

Algorithm 1 Calculate Landmark Distance and Angle

function CALCULATE_DISTANCE(Landmark1, Landmark2)
    lm_dst ← distance between points in xyz space
    return *lm_dst*
end function
if Landmark group is detected **then**
    Calculate reference length
    ref_length ← CALCULATE_DISTANCE(ALM1, ALM2)
    Calculate distance between two landmarks
    lm_distance ← CALCULATE_DISTANCE(LM1, LM2)
    if both Landmark groups are detected **then**
        Get the distance ratio
        $lm\_distance\_ratio \leftarrow \frac{lm\_distance}{ref\_length}$
        Convert distance ratio to motor joint angle
        $desired\_angle \leftarrow (offset - lm\_distance\_ratio) \times Amp$
    end if
end if

2.5. Joint Angle Transmission

Following the calculation of joint manipulation angles for each joint, the rospy library was utilized to transmit the computed joint angle data to the ROS environment. This data transmission process involves a publisher (referred as "talker") and a subscriber (referred as "listener"). In this setup, a Python script functions as the publisher, while the "robot-control" file serves as the subscriber.

The "robot-control" file encompasses information related to joint IDs, motor IDs (specifically, manipulator motors), and angle data. The data that is calculated and sent from the publisher updates the angle information, consequently causing the robot to execute movements in accordance with the updated joint angles. The transmitter node definition is explained in Algorithm 2 as pseudo-code format.

Algorithm 2 Define the Transmitter Node and Motor Joint Controllers

```
procedure TALKER(J1, J2, ..., JN)
    Joint1 ← publisher(Target/J1_contrl/command, data_type, queue_size)
    ...
    JointN ← publisher(Target/JN_contrl/command, data_type, queue_size)
end procedure
                                              ▷ Publish the Desired Data
Joint1.publish(J1)
...
JointN.publish(JN)
```

Referring to Algorithm 2, the control file specifies the joint name for that particular joint, and this information is transmitted in the form of string data. Meanwhile, performance optimization has been achieved by eliminating redundant operations, such as the creation of multiple nodes during data transmission. Instead, the data is transmitted via a single node, encompassing all nineteen motor joint angles within the transmitted data.

3. Results and Discussions

In the field of computer vision-based robotic control, a commonly employed control methodology involves detecting and executing simple and pre-defined motions. However, in this context, achieving real-time and efficient control is needed by processing and interpreting the gesture data obtained directly through computer vision, rather than relying solely on pre-defined motion sequences. In this work, a real-time robotic arm controller was developed to resolve this problem by processing visual data acquired from the human arm and hand gestures, which were then converted into manipulator data. The developed system was able to control the robotic arm independently in a simulation environment without any specific definitions.

In this paper, to create the robotic arm controller, it was essential to design and simulate the unique anthropomorphic robot arm. The design of the robot arm was entirely developed using the ROS. The angle data for each motor joint of the robot has been derived from the relative lengths obtained from landmark position data which are acquired through Mediapipe. These length differences have been processed to obtain manipulator angle data. Subsequently, these obtained manipulator angle data are published to the ROS to control the designed robotic arm.

The computer vision-based robotic arm controller and the simulated robotic arm are used to demonstrate the capabilities of the developed system, as depicted in Fig. 5, Fig. 6, and Fig. 7.

In Fig. 5, an open hand position is implemented using the computer vision-based robotic arm controller. In Fig. 6, a more challenging operation is depicted to better reflect the project's objectives, which is to realize robotic control with more sophisticated gestures. Fig. 7 is chosen to demonstrate that even in a fully closed hand position, with landmarks closely aligned, no issues were detected during the wrist rotation operation.

As a result, a humanoid robotic arm system is designed using ROS and Gazebo. This system is controlled with a real-time robotic arm controller that replicates the hand and arm movements of the user by processing landmark positions captured with a computer vision algorithm using Mediapipe. The developed robotic system demonstrated promising results when replicating sophisticated hand gestures which is the main purpose of this project.

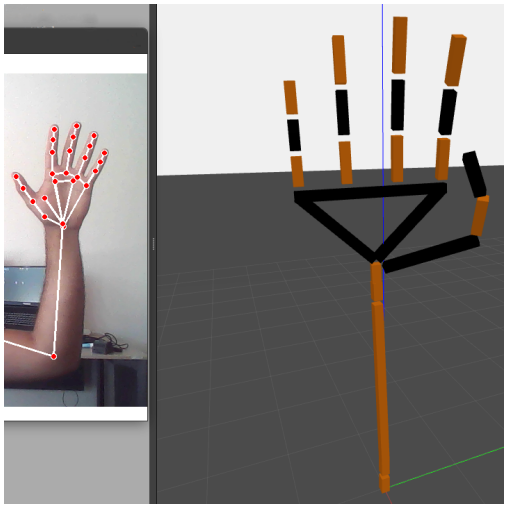

Fig. 5. Simulated open hand position

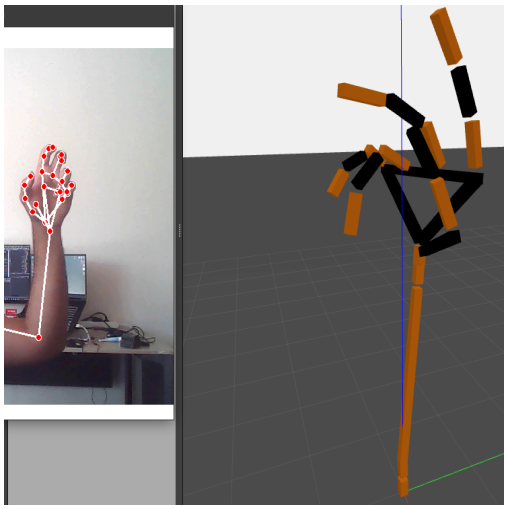

Fig. 6. Simulated random hand position

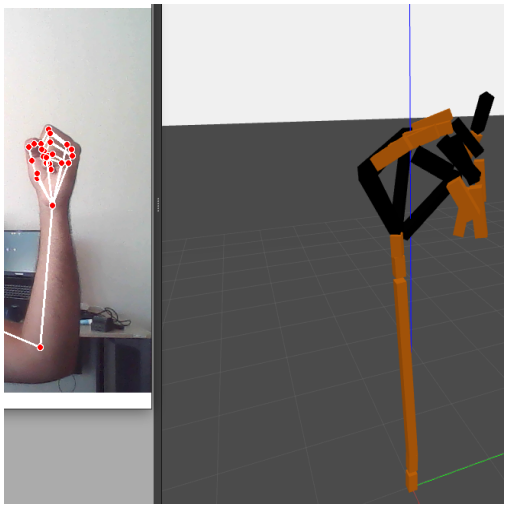

Fig. 7. Simulated closed hand position

4. Conclusions

In conclusion, the focus of the project was to develop a real-time robotic arm controlled by user hand and arm gestures. This innovation was characterized by a departure from simple open-and-close movements, resulting in an enhanced ease of manipulation suitable for a variety of applications and promises to increase precision in various applications. The primary objective of this project involved the transformation of intricate hand and arm movements, captured by a camera, into corresponding robotic arm actions, thereby creating a heightened sense of realism. The initial step comprised the identification of landmarks on the hand and arm, which, in turn, facilitated the calculation of crucial joint angle data necessary for the robotic arm's motion.

Simultaneously, as the development of the computer-based robotic arm controller progressed, a completely novel humanoid robotic arm was designed. Using ROS and Gazebo, this engineered arm was visualized and simulated. Following the simulation phase, individual motor joints were carefully defined within the structure of the robotic arm. Each of these joints received a dedicated node name to enable the manipulation of its respective angle data. A computer vision-based algorithm is used to calculate the landmark distances and corresponding angles using Mediapipe landmarks.

The orchestrated interaction of robotic arm joints facilitated the seamless publication of calculated joint angle data from the computer vision-based robotic arm controller via rospy, thereby directing the motion of the designed robotic arm. The developed system showed promising results in different scenarios such as open and closed hand positions, along with wrist rotation operation. This approach can make a contribution to the healthcare sector, factories, and laboratories for several applications.

As a potential avenue for future development, this project offers the possibility of refinement. Enhancements may involve optimizing the joint angle controller algorithm to ensure increased precision in angle calculations, especially for complex hand and arm gestures. Furthermore, the exploration of dual-hand control could be implemented, allowing for the simultaneous manipulation of two robotic arms, and expanding the range of potential applications. Ultimately, the envisioned path for this project includes real-world implementation. This could entail the integration of a 3D printed chassis and the inclusion of PWM-controlled electronic motors, which would be overseen by powerful microcontrollers like 32-bit ARM Cortex microcontrollers. This advancement would enable further practical exploration and research, propelling the project into new frontiers.

5. References

[1] U. Health. (2018) About the davinci surgical system — uc health. [Online]. Available: https://www.uchealth.com/services/robotic-surgery/patient-information/davinci-surgical-system/

[2] R. P. D, J. G. Sampathkumar, A. K. T, and S. R. Senthilkumar, "Gesture based robot arm control," *NVEO - NATURAL VOLATILES ESSENTIAL OILS Journal — NVEO*, pp. 3133–3143, Nov 2021. [Online]. Available: https://www.nveo.org/index.php/journal/article/view/893

[3] M. Wameed and A. M. Alkamachi, "Hand gestures robotic control based on computer vision," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 2, pp. 1013–1021, Feb 2023. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/2984/1562

[4] M. Wameed, A. M. Alkamachi, and E. Erçelebi, "Tracked robot control with hand gesture based on mediapipe," *Al-Khwarizmi Engineering Journal*, vol. 19, no. 3, pp. 56–71, Sep 2023.

[5] ROS. (2020) Ros.org — powering the world's robots. [Online]. Available: https://www.ros.org/

[6] Gazebo : Tutorial : Urdf in gazebo. http://classic.gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros.

[7] Gazebo : Tutorial : Installing gazebo-ros-pkgs (ros 1). [Online]. Available: https://classic.gazebosim.org/tutorials?tut=ros_installing

[8] J. Kapukotuwa, B. Lee, D. M. Devine, and Y. Qiao, "Multi-ros: Ros-based robot simulation environment for concurrent deep reinforcement learning," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, Aug 2022.

[9] (2019) tr/ros/tutorials/python kullanılarak yayıncı (publisher) ve İzleyici (subscriber) düğümleri yazma - ros wiki. [Online]. Available: http://wiki.ros.org/tr/ROS/Tutorials/Python%20Kullanlarak%20Yaync%20%28Publisher%29%20ve%20zleyici%20%28Subscriber%29%20Dmleri%20Yazma#Publisher_D.2BAPwBHwD8-m.2BAPw_Yazma

[10] Hand landmark detection guide for web — mediapipe. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[11] Pose landmark detection guide for web — mediapipe. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/web_js