EDUCATIONAL 5 AXIS ROBOT CONTROLLER OPTIMIZATION USING ARM HARDWARE INSTRUCTIONS

Alexandru-Ioan ANASTASIU[1],
Cozmin CRISTOIU[2], Florea Dorel ANANIA[3]

Rezumat. *Această lucrare își propune să prezinte o metodă de programare a unui controller de tip Raspberry Pi pentru un braț articulat cu 5 axe. Scopul este optimizarea cinematicii directe, bazându-se pe o abordare universală. Înmulțirea matricelor este implementată folosind instrucțiuni de hardware prezente în standardul ARM. Avantajul principal al acestei metode este viteză mai mare de programare, ceea ce oferă viteză mai mare de reacție robotului. Cu ajutorul unui set de instrucțiuni creat în acest scop se asigură programabilitatea brațului robotic.*

Abstract. *This paper presents a programming method of a Raspberry Pi controller for a 5-axis articulated arm robot. The goal is optimization of direct kinematics calculations, based on the universal approach for direct kinematics method. Matrix multiplication is implemented using ARM hardware instructions. The main advantage of this method is lower computation time, which means faster robot response time. Programmability of the robot is done by means of a custom-made instruction set, called RASM.*

Keywords: Robot, Programming, Hardware instruction

DOI https://doi.org/10.56082/annalsarscieng.2022.1.32

1. Introduction

Modern advancements in technology have allowed broad access to technology. This is beginning to show even in de domain of Robotics, where more and more kits are becoming readily available. This availability allows students to study and understand the key aspects of industrial robotics, without the need to access prohibitive or specialized hardware. Using one such kits, the possibility of creating a scaled-down but otherwise analogous 5 degree of freedom robot arm was approached. The focus was the robot firmware, in other words the underlying software that allows the robot to function.

2. RASM – The Robot Assembler

One of the goals of this research was development of a custom programming language, in order to allow advanced control of the device. By definition, a robot must be programmable, so as to be able to perform certain actions in a definite,

[1] Student, University POLITEHNICA of Bucharest, IIR Faculty, Spl Independentei 313, ZipCode 060042. E-mail: a.anastasiu@outlook.com
[2] Lecturer, University POLITEHNICA of Bucharest, IIR Faculty, Spl Independentei 313, ZipCode 060042. E-mail: cozmin.cristoiu@gmail.com
[3] Associate Professor, University POLITEHNICA of Bucharest, IIR Faculty, Spl Independentei 313, ZipCode 060042. E-mail: dorel.anania@upb.ro

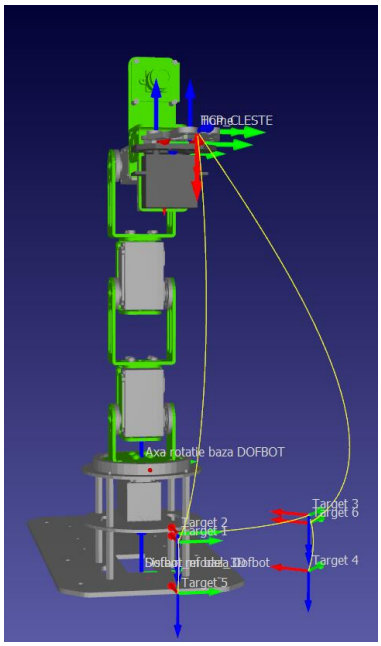
repetitive manner. The first component of the program represents such a feature. Branded RASM (Robot Assembly), it loosely follows a syntax similar to ARM assembly [1], being able to interpret and execute 17 specialized opcodes (instructions), related to the spatial movement of the manipulator. This allows the robot to be completely programmable, which includes arithmetic operations (addition, subtraction, multiplication, as well as variable support) and logic operations (if, else, as well as rudimentary branching support, by means of goto-like instructions).

RASM is a hybrid language, this meaning that while it requires an interpreter (the code does not run directly on hardware, but rather has an abstraction layer that translates the generic instructions to the hardware-specific operations), it also utilizes an assembler, a program that takes human-readable text code and transforms it into machine code, reducing computational power required on the target device.

This assembler takes each text line from a .rasm file and converts them to an Instruction class (defined as standard across RASM implementations) object. A "program" can be defined as a vector (or array) of such Instructions. The assembler also has a secondary role, that of variable optimization. Because a human programmer might name a variable using an alphanumeric combination, this greatly slows down the process of variable lookup on the machine which runs the code. For example, computation time would be greatly increased if, instead of searching for "variable1", a computer would be looking for numeric value of 100. This computation is offset by the assembler, which takes each reference to an alphanumeric variable name and transforms it into a reference to a numeric value.

In order to better explain how logic is handled, one must first define how a program is structured. Since instructions are stored in an array, these can be parsed by means of an index (referred to as a Program Counter, abbreviated PC, analogous to x86 and ARM64 assembly). This index can, hence, be manipulated in any way, in order to skip instructions, jump backwards in the program, or enable branched execution. The assembler takes a label defined by the user and simply replaces all references in the program with the respective PC values.

Both the RASM interpreter and assembler utilize highly optimized switch statements in order to achieve fast execution of user code. Being a specialized language with few instructions, this allows it to be faster than equivalent, general-purpose languages (for example Python), while also having a much lower memory footprint (a 1000-instruction program will consume around 40kb of memory). Additionally, the interpreter is designed to be extendable, with anyone being able to add code to it, in order to better suit their application (this means custom instructions can be added, as well as existing ones being redefinable).




```
NME PICKNPLACE          // set name of the program
GHME                    // return to home pose
ANG 6 0                 // open the gripper

ANGS 0 29 33 28 0 0     // head to target 1
ANG 6 60                // close the gripper

ANGS 0 32 24 34 0 60    // move to target 2
ANGS 50 32 24 34 0 60   // move to target 3
ANGS 50 22 66 3 0 60    // move to target 4
ANG 6 0                 // open the gripper

ANGS 50 32 24 34 0 0    // move to target 3
ANGS 0 32 24 33 0 0     // move to target 2
ANGS 0 22 66 3 0 0      // move to target 5
ANG 6 60                // close the gripper

ANGS 0 32 24 33 0 60    // move to target 2
ANGS 50 29 34 28 0 60   // move to target 3
ANG 6 0                 // open the gripper

ANGS 50 32 24 34 0 0    // move to target 6
SPD 2000                // set speed factor
ANG 2 50

GHME                    // go home
END                     // end program
```

Fig. 1. An example of a pick and place application simulated in RoboDK and corresponding RASM code

3. Teach Pendant

In order to provide a graphical interface for the robot, a program was developed. Designed to run on handheld devices (laptops, tablets or similar), it emulates the functionality of a Teach Pendant. Internally, it is based on two different event loops (Fig. 2), running on a fixed time pattern. The first handles mechanical control of the robot, while the second performs video processing. The two run on different threads, allowing faster execution.

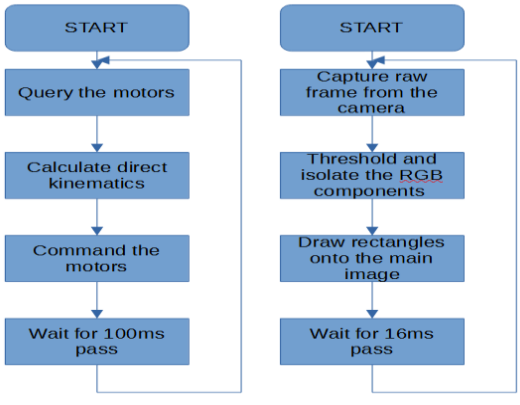


Fig. 2. Two different loops algorithms

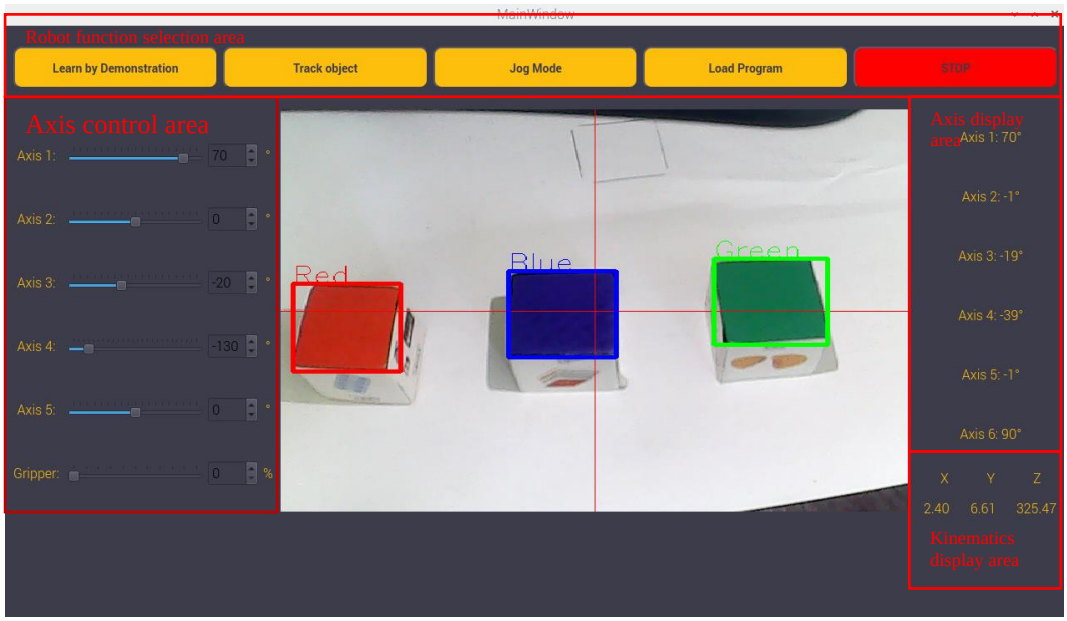


Fig. 3. The teach pendant interface.

The Teach Pendant program allows a user to control every aspect of the robot (Fig. 3). As such, it has independent control over the axes, by means of sliders, text boxes into which absolute angle values can be introduced, or by means of an attached joystick.

3.1. Camera

The robot kit utilized is equipped with a camera for machine vision, whose output is displayed within the center of the interface. This represents a direct stream from the sensor, alongside a crosshair that is meant to aid in positioning of the gripper. Additionally, in order to better exemplify the utility of such a sensor, the robot is able to detect, isolate and follow objects based on color. red, blue, and green items are each assigned a color-matching bounding box. Upon the user pressing the "Track object" button, he can select which color the robot must follow, after which input is disabled (the robot enters an automated mode), in which it attempts to always center the respective object to the viewfinder.

3.2. Learning mode

Analogous to real-life industrial robots, a teach-in mode is available, in which the robot can be positioned by means of the joystick. Upon a certain position and orientation is determined, an operator can press a button to record that target. The robot will then follow the given path, staying as faithful as possible to the given

coordinates. A (virtually) infinite number of points may be recorded, thanks to coordinates taking little in the way of program memory.

3.3. Program

The robot may also be programmed by loading a compiled RASM binary, by an operator pressing the "load program" button. After load, a confirmation prompt is displayed, after which the robot begins automatically following the program. It will do so indefinitely, or until stopped by an operator.

3.4. Outputs

The Teach Pendant also informs an operator of the various parameters of the robot at any time: the side panel shows a live (updated every 100ms) readout of the angles of each axis. Additionally, a triplet of coordinates is displayed, calculated by means of direct kinematics (section 4 describes the means of how this is achieved).

4. The robot firmware

The robot's firmware represents the interface between RASM, the Teach Pendant and the actual robot hardware which comprises the platform. While the formerly discussed components were universal, requiring only minor modifications or even no modification, the firmware is specific to each robot. A new robot kit will require a new firmware.

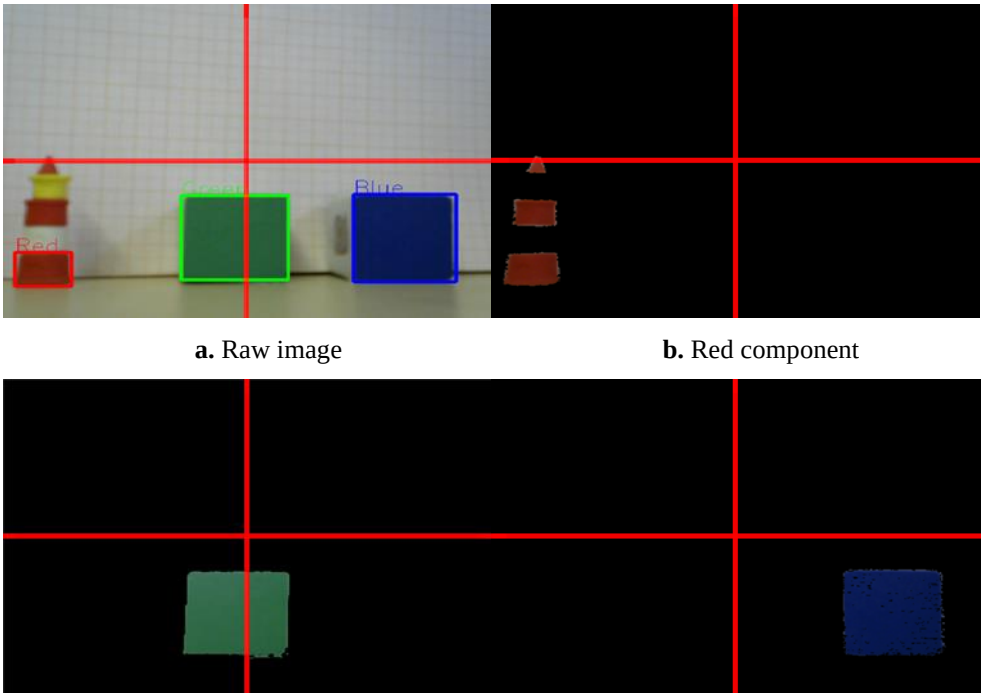
Thus, a word on the hardware is necessary. At the center of the robot lies a Raspberry Pi board, a single-board ARM based computer which acts as the robot controller. However, its role is only achieved when used in conjunction with an on-board coprocessor (an stm-8 microcontroller). Through reverse-engineering the code and documentation provided by the manufacturer, the architecture was understood: The Raspberry issues commands to the microcontroller over the i2c (inter-integrated circuit) bus. The microcontroller then controls the servomotors which make up the robot.

The firmware can generally be divided in 4 sections:

Motor module This component represents the interface with motors. Incoming angle commands are parsed, error-checked (in order to prevent erroneous or dangerous commands, such as an angle that might put the robot in an invalid pose), then issued to the accompanying microcontroller.

Camera module The camera feed is processed on-device, with the output being displayed on the Teach Pendant. The image is taken from the sensor, converted

from RGB to HSV (hue-saturation-value), in order to get better color recognition, then the isolation step begins. During this, an algorithm is used to threshold the different color values, resulting in a decomposition of the initial image into three different profiles: red, green, and blue (Fig. 4).



a. Raw image **b.** Red component

c. Green component **d.** Blue component

Fig. 4. An example of image decomposition into colors

The algorithm takes the initial image, then blurs it. This is done in order to remove much of the picture noise that might confuse the algorithm. While details are lost in this way, the general outline (which is representative for the color) remains unaltered. After that, the image is "thresholded", a process by which pixel values are checked to see if they are in within a certain range on the color spectrum. If they are in that range, they are left untouched, else, they are filled with black. The resulting images contain only the color profile, which is searched for, and nothing else. Then, a rectangle is generated in such a way that it completely encases the areas of non-black color. The coordinates and size of this rectangle are used to draw onto the initial image, resulting in an outline of each colored object. In order to simplify matters, only the largest area of color is considered when drawing the borders.

Kinematics module The kinematics module handles the calculations required for the position of the end-effector of the robot arm. In order to reduce code complexity,

the "Universal method" [3] is used. This approach deviates from the usual Denavit-Hartenberg [4].

method by keeping the coordinate frames constant across axes (the "x" axis is always pointing in the direction of the robot's "home" pose, the "y" axis is always perpendicular to the "x" axis and pointing into the plane, and the "z" axis always points upwards.). Furthermore, a single matrix is used to represent a joint, one that contains 4 rows and 4 columns. The first 3x3 sub-matrix represents the orientation of the axis, the next 3x1 represents the spatial coordinates of it, while the bottom row denotes the scale at which the modeling is done.

Matrix multiplication is done between each one of these joint matrices, resulting in the end effector matrix. Traditionally, this is done in programming using an iterative method, performing repeated multiplications and additions. This approach is valid, but requires great computational power, and due to the fact that it utilizes three loops, puts a great demand on the processor's cache and arithmetic logic unit.

Below are presented the matrices used in calculating the coordinates and orientation of the robot's tool flange.

$$\begin{bmatrix} R_1 & R_2 & R_3 & T_x \\ R_4 & R_5 & R_6 & T_y \\ R_7 & R_8 & R_9 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta 1) & -\sin(\theta 1) & 0 & 0 \\ \sin(\theta 1) & \cos(\theta 1) & 0 & 0 \\ 0 & 0 & 1 & 74 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta 2) & 0 & \sin(\theta 2) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta 2) & 0 & \cos(\theta 2) & 29 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta 3) & 0 & \sin(\theta 3) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta 3) & 0 & \cos(\theta 3) & 82.85 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta 4) & 0 & \sin(\theta 4) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta 4) & 0 & \cos(\theta 4) & 82.85 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 71.5 \\ 0 & \cos(\theta 5) & -\sin(\theta 5) & 0 \\ 0 & \sin(\theta 5) & \cos(\theta 5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Explained briefly, each joint can be represented through an X-Y-Z coordinate system. As illustrated in Fig. 4 and detailed above, these frames maintain orientation throughout the length of the robot. For rotations around each axis, the rotation matrix component stays the same (the only difference being the argument of the trigonometric functions, which represents the value of the angle on each axis). The resulting matrix contains the 3x3 sub-matrix which represents rotation of the end effector (R_1-R_9) and the translation (T_x, T_y, T_z), relative to the robot's base. Furthermore, the rotation matrix may be decomposed into Euler angles, utilizing the method described in [5].

First, create the generalized rotation matrix of any spatial point, by multiplying the matrices which describe rotation around each axis (x, y, z):

$$G = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix} \quad (2)$$

From this, it can be concluded that

$$G = \begin{bmatrix} \cos(\theta)\cos(\phi) & \sin(\theta)\sin(\psi)\cos(\phi)-\cos(\psi)\sin(\phi) & \sin(\theta)\cos(\psi)\cos(\phi)+\sin(\psi)\sin(\phi) \\ \cos(\theta)\sin(\phi) & \sin(\theta)\sin(\psi)\sin(\phi)+\cos(\psi)\cos(\phi) & \sin(\theta)\cos(\psi)\sin(\phi)-\sin(\psi)\cos(\phi) \\ -\sin(\theta) & \cos(\theta)\ \sin(\psi) & \cos(\theta)\cos(\psi)) \end{bmatrix} \quad (3)$$

And thus, if $R_7 \neq \pm 1$:

$$\begin{aligned} \theta 1 &= -\arcsin(R_7) \\ \theta 2 &= \pi + \arcsin(R_7) \\ \psi 1 &= \arctan2\left(\frac{R_8}{\cos\theta 1}, \frac{R_9}{\cos\theta 1}\right) \\ \psi 2 &= \arctan2\left(\frac{R_8}{\cos\theta 2}, \frac{R_9}{\cos\theta 2}\right) \\ \phi 2 &= \arctan2\left(\frac{R_4}{\cos\theta 1}, \frac{R_1}{\cos\theta 1}\right) \\ \phi 2 &= \arctan2\left(\frac{R_4}{\cos\theta 2}, \frac{R_1}{\cos\theta 2}\right) \end{aligned} \quad (4)$$

Else:

$$\begin{aligned} &\phi 1 = \phi 2 = 0 \ \text{(or any value)} \\ &\theta = \pi / 2, \text{ if R7 = -1, else } \theta = -\pi / 2 \\ &\psi = \phi + \arctan2(R_2, R_3) \text{ if R7 = -1 , else } \psi = -\phi + \arctan2(-R_2, -R_3) \end{aligned} \quad (4)$$

These values must then be checked against the actual manipulator, in order to verify that the position generated is achievable (Fig.5).

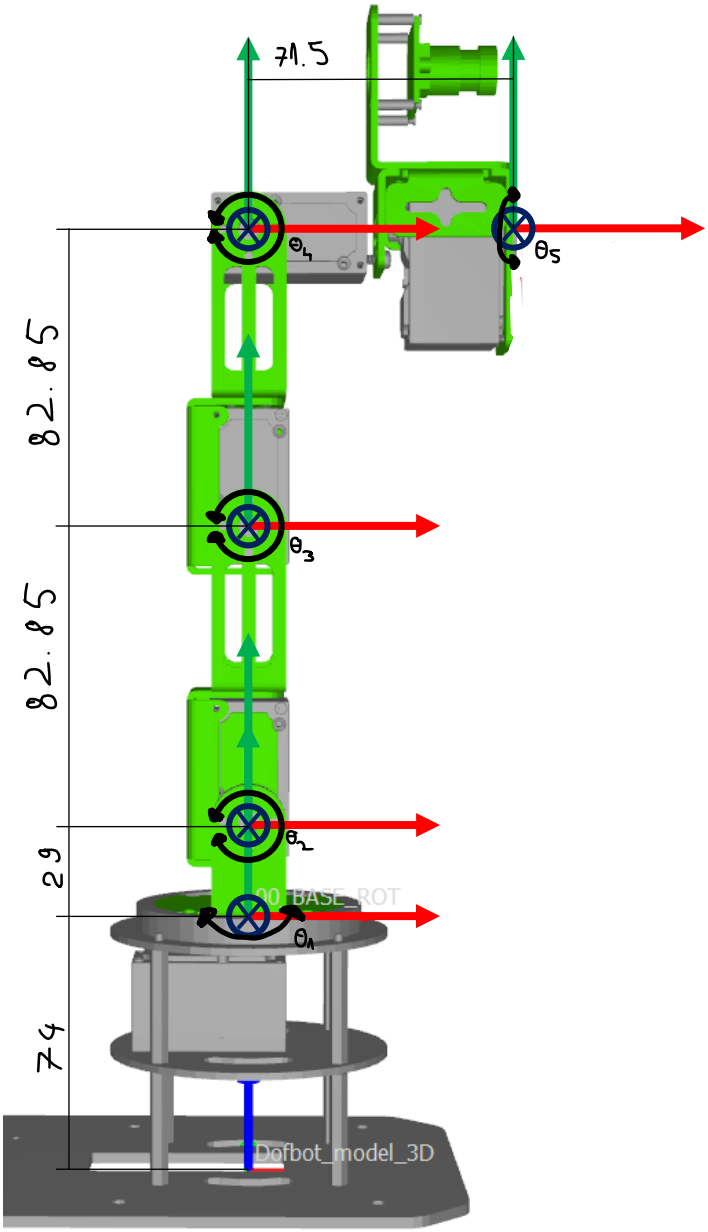


Fig. 5. Positions of the axes along the robot

A newly proposed method is to leverage AARCH64 (the instruction set utilized on arm64 processors hardware instructions, in the form of ARM NEON [6]. NEON is a SIMD (single-instruction, multiple-data) instruction set, designed to accelerate computation of vector operations.

At a silicon level, a processor has a set of small, fast-access memory locations called registers. These registers are used by the processor during operation, as a location to execute calculations, logical operations, as a fast storage for the stack and frame pointers. ARM64 offers 64 such registers (32 for integer operations, 32 for floating-point), each register being 64 bits wide.

A processor is limited in computational speed by the number of operations it must undertake in order to execute a certain task. This is where SIMD is utilized. Suppose a mathematical operation, such as addition of two vectors, were to be performed. An iterative method will load a register with the first value of the vector, the next with the first value of the first vector, then may allocate a third register to store the result of the addition. This results in a complexity (the number of operations that must be performed in order to execute a task of size "N") of O(N).

For an array of 4 elements, 4 operations must be performed. Using ARM NEON, this may be optimized in such a way that only *one* operation is performed [7]. This is done by combining 4 floating point registers into 1 *quadword* register. Effectively, these behave as vectors, with the added benefit of being able to do arithmetic operations using just one instruction

Applying this technology to the use case presented, matrix multiplication is greatly simplified. The resulting code executes almost two times faster (determined experimentally). The robot is therefore able to execute cycles that would require faster movement, spending less processing time on kinematics calculations.

Program module This represents the RASM interpreter, which loads the program from a compiled binary into memory, then fetches instructions, executing them in a programmed manner. The program module gives a user full control over the robot's hardware, allowing full mobility, trajectory generation, integration with the camera module [8], [9].

By developing a postprocessor for a robot simulation program (such as RoboDK), programs can be developed that integrate the robot into a real application, beyond the demonstration purpose [10], [11].

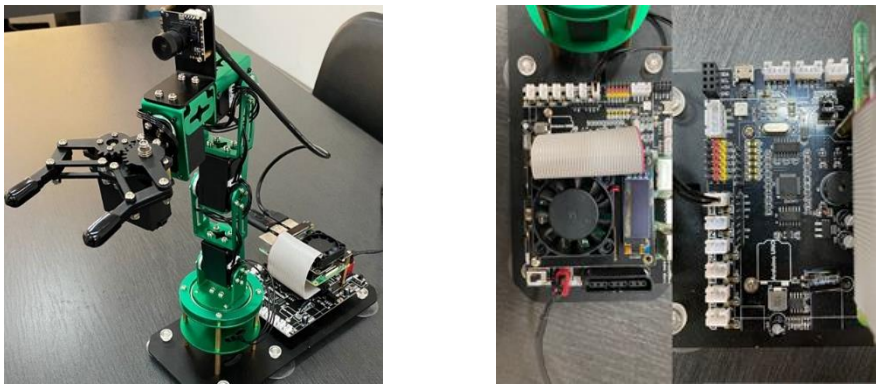


Fig. 2. Final construction of the robot arm
a. The Robot arm **b.** The Raspberry Pi and microcontroller

Conclusions

This software presented a novel method of exploiting a low-cost, commercially available robotics kit, with applicability in the educational domain, by simulating the functional principles of an analogous industrial robot (programmability, teach-pendant functionality). The approach used in this direction implements current technologies, by using hardware instructions in order to accelerate computation. The application is easily adaptable to any robot kit, bringing real-life engineering notions into the grasp of students. Such software had not been created at the time of writing.

Micro robotics is an area of technology that has seen rapid development in the past years, having great applicability in areas such as personal services and entertainment. However, such robots may also be utilized in educational applications, since they mimic the real-life construction of industry-used robots.

Articulated arm type robots have great utility thanks to their flexibility and versatility, given by the kinematic structure. Moreover, robots equipped with cameras have the capability to be semi-self-programmable, reacting to their environment and adapting to the conditions. This paper presented a possible such application, through the ability of the arm to do color-based object tracking. This was done utilizing a thresholding algorithm, by isolating specific areas of the spectrum and defining them as a specific color, then matching pixels recorded from a camera.

One of the great challenges in robotics is represented by programming. This paper proposed a new pseudo-programming language, named RASM, that aims to greatly

simplify the way such a robot may be controlled. It is compliant with the rules of direct kinematics and could in the future be interfaced with different off-line programming environments, through the use of postprocessors. Thanks to the fact that the interpreter is built using C++, it maintains speed and performance, critical in any application.

In order to further optimize the execution of instructions, hardware acceleration was used where possible, fully utilizing the ARM-based architecture. This results in matrix multiplication that is twice as fast compared to an iterative approach.

REFERENCES

[1] ARM Architecture Reference Manual, 2022 Arm Limited, https://developer.arm.com/documentation/ddi0487/latest

[2] Basic Thresholding Operations, https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html

[3] Cozmin CRISTOIU, Adrian NICOLESCU*, "New approach for forward kinematics modeling of industrial robots with closed kinematic chain", 2017*

[4] Balasubramanian, Ravi. *"The Denavit Hartenberg Convention.",* USA: Robotics Institute Carnegie Mellon University (2011)

[5] Slabaugh, Gregory G. "Computing Euler angles from a rotation matrix." Retrieved on August 6.2000 (1999): 39-63.

[6] Reddy, Venu Gopal. "Neon technology introduction." ARM Corporation 4.1 (2008): 1-33

[7] Kim, Dae-Hwan. "ARM NEON Assembly Optimization."

[8] Dobrescu, Tiberiu Gabriel & Dorin, Alexandru & Nicoleta-Elisabeta, Pascu & Ivan, Ioana. (2011). CINEMATICA ROBOTILOR INDUSTRIALI.

[9] Blanchette, Jasmin, and Mark Summerfield. C++ GUI programming with Qt 4. Prentice Hall Professional, 2006.

[10] Garbev, Atanas, and Atanas Atanassov. "Comparative Analysis of RoboDK and Robot Operating System for Solving Diagnostics Tasks in Off-Line Programming." 2020 International Conference Automatics and Informatics (ICAI). IEEE, 2020.

[11] Holubek, Radovan, et al. "Offline programming of an ABB robot using imported CAD models in the RobotStudio software environment." Applied Mechanics and Materials. Vol. 693. Trans Tech Publications Ltd, 2014.