Robotic automation of medical kit dispensing supported by a graphical domain-specific language*

Milena Almeida[1], Felipe Mendonça[1],
Marcondes da Silva Júnior[1], and Gustavo Carvalho[1]

Abstract— **Different contemporary tasks can be aided by robotic systems. However, developing and programming such systems is not a straightforward task, since it typically requires robotics-specific knowledge in addition to traditional programming proficiency. To facilitate the development of robotic applications for the healthcare domain, this work proposes a graphical domain-specific language (DSL) for specifying missions of robotic arms; a mission comprises a sequence of actions that are required to accomplish a given task. Our DSL (H-ArmDSL) is based on the Blockly library (Google), and it focuses on a specific robotic arm: the Kinova Gen3 Lite one. Nevertheless, it can be customised taking into account other vendors. H-ArmDSL was successfully validated in a real setting, where a robotic arm is employed to automate the assembly of medical kits comprising controlled medicines at a public hospital in Brazil. By regulation, the access to and manipulation of such medicines can only be performed by a chemist. Therefore, our robotic-based solution saves an important working time of a specialist, who can now focus on other activities.**

I. INTRODUCTION

We are in the midst of a new industrial revolution, characterised by advancements in emerging technologies in areas such as robotics, artificial intelligence, nanotechnology, quantum computing, internet of things, autonomous vehicles, among others. Machines are now even more interconnected and communicate with each other, enabling systems to make decisions without human intervention; for instance, autonomous robotic systems have been used in very different settings and applications.

An interesting opportunity in the robotics field is employing robotic arms to perform specific tasks, such as object manipulation. These tasks must be performed within a specified time frame and with a highly precise trajectory. An application example would be handling medications in pharmaceutical environments. The degree of flexibility of the robot and its perception of the environment, in which it operates, are of utmost importance for carrying out precise and timely movements. These robots may operate autonomously, or collaborate with humans (i.e., collaborative robots – cobots). Cobots are considered a new technology (advanced manufacturing technology) that is primarily characterized by the fact that the robots' and operators' work zones overlap, creating a common workspace [1].

Developing robotic applications, such as the one mentioned before (handling medications), is not straightforward, since it typically requires robotics-specific knowledge in addition to traditional programming proficiency. Therefore, the main research problem addressed by this work concerns how to overcome the challenges of programming robotic arms and, thus, reduce the programming effort. Here, we propose to address this problem by using block-based programming. At first, this programming style emerged as a new trend in the field of computer programming with the aim of facilitating learning. The concept of block-based programming has became increasingly popular, and some tools have been developed using this concept, such as Blockly [2]: a library by Google where it is possible to generate visually editable blocks for programming in different languages.

With Blockly, it is possible to design graphical domain-specific languages (DSL). A DSL is a programming language designed for a specific problem rather than providing general solutions for all types of domains [3]. Generally speaking, the goal of DSLs in robotic arm control is focused on generating movements to manipulate elements, allowing for easier modification of kinematic parameters [4].

In this work, we propose a graphical DSL (named H-ArmDSL) based on Blockly for specifying missions of robotic arms employed in a healthcare domain; a mission comprises a sequence of actions that are required to accomplish a given task. It is designed considering a specific robotic arm: the Kinova Gen3 Lite one. Nevertheless, it can be customised taking into account other vendors. The validation of H-ArmDSL considered a real case study provided by *Hospital das Clínicas da Universidade Federal de Pernambuco* (HC-UFPE – a public hospital in Brazil), where a robotic arm is employed automate the assembly of medication kits comprising controlled medicines, by collection and dispensing medications in a pharmacy.

By regulation, the access to and manipulation of controlled medicines can only be performed by a chemist. At HC-UFPE, we have a daily average of 36 tickets involving such medicines, each one comprising about 3 different medications; being 104 tickets the highest recorded number of tickets on a single day. Therefore, our robotic-based solution for automatic manipulation and dispensing of medical kits saves an important working time of a specialist, who can

*This work was supported by the research and innovation cooperation project between Softex (funded by the Ministry of Science, Technology and Innovation through Law 8.248/91 in the scope of the National Priority Program) and CIn-UFPE. This work is partially supported by INES (www.ines.org.br), CNPq grant 465614/2014-0, CAPES grant 88887.136410/2017-00, and FACEPE grants APQ-0399-1.03/17 and PRONEX APQ/0388-1.03/14.

[1]Milena Almeida, Felipe Mendonça, Marcondes da Silva Júnior, and Gustavo Carvalho are with Centro de Informática, Universidade Federal de Pernambuco, 50.740-560, Recife, Brazil {mcsa2, fasm, mrsj}@softex.cin.ufpe.br, ghpc@cin.ufpe.br

now focus on other activities.

This paper is structured as follows. In Section II, we provide background information, covering fundamental concepts related to the adopted robotic arm, the Kinova Gen3 Lite, as well as to the Blockly library. Section III details the devised DSL (H-ArmDSL). We present its default and domain-specific blocks, in addition to explaining how H-ArmDSL programs are automatically mapped to lower-level code that is executed by the robotic platform. Section IV presents the validation of H-ArmDSL considering robotic automation within a pharmaceutical scenario. Finally, in Section V, we present our final conclusions, besides addressing related and future work.

II. Background

In this section, we provide background information necessary for understanding the remainder of this paper. First, in Section II-A, we describe the robotic arm that is initially targeted by H-ArmDSL. Then, in Section II-B, we present the Blockly library, which is used to define our graphical DSL.

A. *Kinova Gen3 lite*

Being the newest and most compact member of the Kinova ultra-lightweight robot series, the Gen3 lite offers a more cost-efficient option if one is looking for professional-grade robotic arms to perform light manipulation tasks. The Gen3 lite is suited both for practical and academic-level education needs. In Figure 1, one can see this arm at our testing environment, which mimics the pharmacy setting: the robotic arm surrounded by trays where the medications are located. The Gen3 lite runs on the Kinova® Kortex™ API software; one can seamlessly share programming and collaborate with other Gen3 lite or Gen3 units. According to its vendor[1], it is ultra-lightweight, highly portable, power-efficient and ideal for mobile applications.

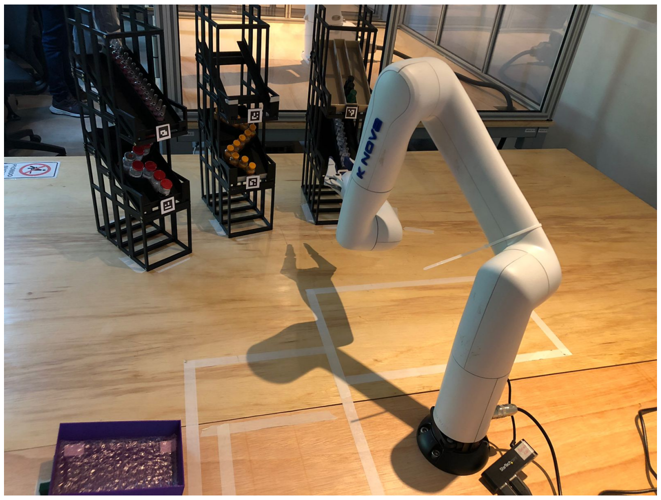


Fig. 1. Kinova Gen3 lite installed at our testing environment.

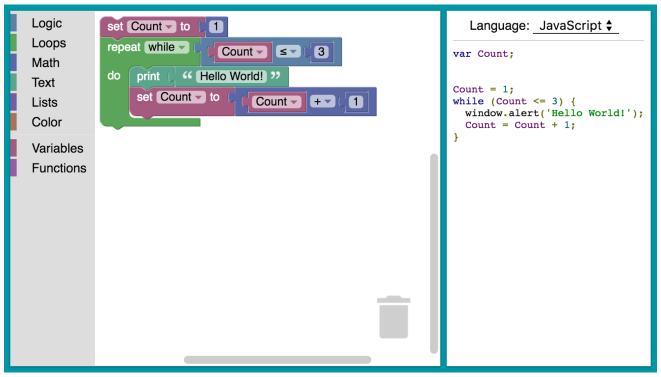
[1]www.kinovarobotics.com/product/gen3-robots

The Gen3 lite is accessible for users with different levels of expertise, and it can be employed to perform simple grasping tasks, but also within the context of more complex manipulation applications. It offers a flexible developing environment, as there is support for MATLAB, ROS, C++, and Python. Other important features of this robotic arm are the following: efficient and transparent actuators, high-level and low-level control, plug-and-play, protection zones, simple connectivity (USB, Ethernet, RNDIS), and connectivity with the Kinova web application from any desktop, laptop, or mobile device [5].

B. *Blockly*

Google Blockly[2] is an open source library that enables visual programming. Its editor uses interlocking graphical blocks to represent code concepts like variables, logical expressions, loops, and more. Therefore, it allows users to apply programming principles without having to worry about syntax details. The visual nature of Blockly adds usability, making it a powerful framework for developing learning applications for novice programmers [6].

In Figure 2, the interface of the Blockly editor is presented. In this example, the first block declares the variable `Count`, and assign 1 to it. Then, we have a loop that occurs `while` the condition `Count` $\leq$ 3 holds. Within the loop, the text `Hello World!` is printed in the console, and the variable `Count` is incremented. On the right side of Figure 2, one can see how the blocks are mapped to a JavaScript code.



```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

Fig. 2. Blockly example (`developers.google.com/blockly`).

In Blockly, it is possible to define new custom blocks, and map them to the desired programming language. Therefore, Blockly offers an intuitive and visual way to create code, but it is also a ready-made UI for defining new visual languages from which syntactically correct code can be generated; this is particularly useful when defining graphical DSLs, which is the situation of this work. It is possible to export blocks to many programming languages, such as: JavaScript, Python, and others. Here, we focus on Python to integrate with the Kinova Gen3 lite API.

In this work, we customise Blockly to create a graphical DSL that facilitates the programming of Kinova Gen3 lite

[2]https://google.github.io/blockly/

robots, tailored for the manipulation of medications. In the next section, we present and detail the proposed DSL.

III. H-ArmDSL

H-ArmDSL is a domain-specific language developed to streamline the programming of robotic arms, particularly the Kinova Gen3 lite, for healthcare applications. As said before, this language is built using the Blockly library, enabling the creation of visual programs through interconnected blocks.

Our DSL employs blocks ranging from classical programming concepts to specific aspects of a robotic arm, such as moving to a particular pose, grasping objects, and others. These blocks are intuitively organized into categories to facilitate program creation. Moreover, a H-ArmDSL program can be automatically translated to a corresponding Python program, which allows for easy integration and execution of the program on the actual robot. Figure 3 shows the initial screen of our provided programming editor.

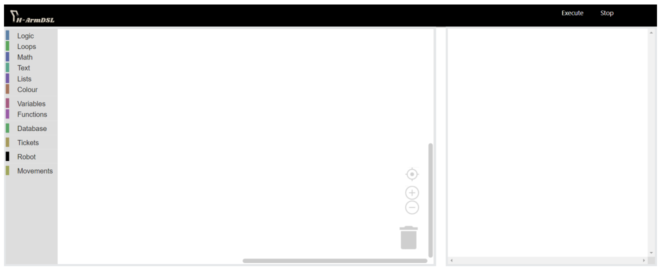


Fig. 3. Programming editor provided by H-ArmDSL.

On the left panel, we have the list of available blocks, grouped by default (e.g., Logic, Loops, etc.) and customised categories (e.g., Movements). On the central panel, we have the space where the blocks are dropped to create a particular program. On the right panel, we have the corresponding Python code, which is automatically generated. Finally, on the top-right panel, we have two buttons (*Execute* and *Stop*) to execute and to stop the execution of the generated Python code on the actual robot, respectively. In what follows, we cover the default (Section III-A) and custom (Section III-B) blocks provided by H-ArmDSL.

A. H-ArmDSL default blocks

In Google Blockly, a toolbox is a key component that provides a set of predefined blocks and categories, facilitating the block-based programming experience. The toolbox serves as a palette from which users can drag and drop blocks into the workspace to construct their programs.

Figure 4 shows some of the default blocks. The `if` block is an example of a *Logic* block. It represents classical conditional testing; if the provided condition holds, its body is executed. There is also an `if-then-else` block.

To perform *Loops*, one can employ `repeat` blocks. The toolbox provides two variants of this block: one performs a sequence of commands a fixed number of times, whereas the other one keeps executing the commands while a given condition holds.

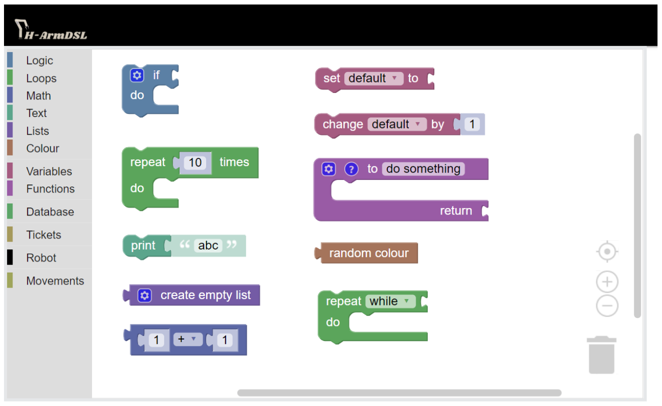


Fig. 4. Default blocks.

In addition to *Logic* and *Loops* blocks, there are also blocks associated with mathematical operations (*Math*), text and list manipulation (*Text* and *Lists*, respectively), and with the declaration and use of variables and functions (*Variables* and *Functions*, respectively). It is also possible to change the colour of blocks (*Colour*). All these default blocks provided by Blockly are also made available by H-ArmDSL.

B. H-ArmDSL custom blocks

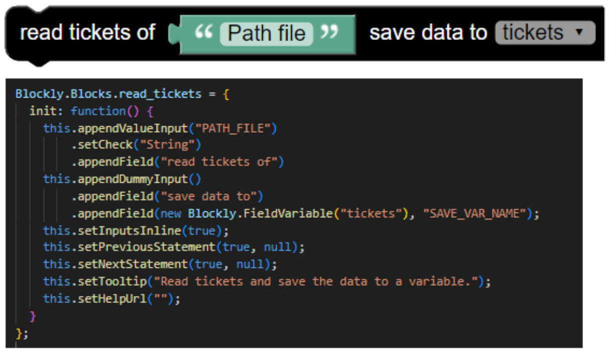
The Blockly API allows one to use JavaScript to define custom blocks. This offers a clear correspondence between visually represented actions and the underlying JavaScript code, enabling a seamless transition from visual design to actual implementation. This visual and block-based approach provided by H-ArmDSL, incorporating domain-specific blocks, significantly simplifies the robotic arm programming process, making it accessible to a broader range of users, including those with no advanced technical training in (robot) programming. The H-ArmDSL custom blocks are grouped into the following categories:

- *Database*: handles communication with databases;
- *Tickets* manages information on tickets and medicines;
- *Robot*: handles communication with the robot;
- *Movements*: moves the robotic arm.

In what follows, we detail how one can define custom blocks. Then, we exemplify some of the H-ArmDSL custom blocks. Finally, we explain how we define code generation to our target language (Python) from block-based programs.

1) Defining custom blocks: Custom blocks are defined in JavaScript. They encapsulate custom functions or procedures in programming, or in a particular domain. In this way, Blockly allows developers to define reusable pieces of code with specific functionalities. For instance, in Figure 5 we see the definition of the `read tickets` block. At the top, we have the graphical representation of this custom block; at the bottom, we have its definition in JavaScript.

This specific block has two input fields. The first one defines the location (path) from where the open tickets are to be read. The second one concerns the variable the retrieved data are saved to. Additionally, besides setting a tooltip, it is defined that this block can be preceded and followed by other



```
Blockly.Blocks.read_tickets = {
  init: function() {
    this.appendValueInput("PATH_FILE")
        .setCheck("String")
        .appendField("read tickets of")
    this.appendDummyInput()
        .appendField("save data to")
        .appendField(new Blockly.FieldVariable("tickets"), "SAVE_VAR_NAME");
    this.setInputsInline(true);
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setTooltip("Read tickets and save the data to a variable.");
    this.setHelpUrl("");
  }
};
```

Fig. 5. Defining custom blocks in JavaScript.

blocks. Other aspects of a block, including those related to its graphical representation, can be customised via JavaScript.

2) Database blocks: Database blocks allow one to create a database to store relevant data. At this moment, we target MySQL. To do this, the block `create database` takes as input the path to a Python script that properly configures the database. In this work, our database comprises patients, hospital locations, medicines, and tickets.

3) Tickets blocks: Ticket blocks are responsible for both creating and retrieving data from tickets. In Figure 6, we show the block `create medicine` that registers that a given medicine is available at the pharmacy. As input, the blocks receives the name of the medicine, its quantity at the pharmacy, and its pose. The last information is required to precisely move the robotic arm during the grasping operation. Other blocks for manipulating tickets are shown in Section IV.

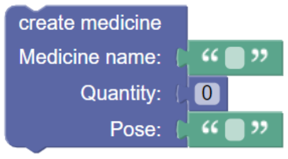
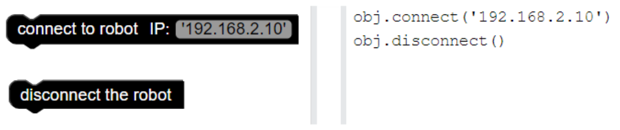


Fig. 6. Block for registering medicines.

4) Robot blocks: The *Robot* blocks handles communication with the robotic arm. The block `connect to robot` has a text input field, which should be used to inform the robot's IP address. Another block belonging to this category is the `disconnect the robot` one, which disconnects from the robot safely. In Figure 7, one can see the graphical representation of these two blocks (on the left side), along with the Python code derived from them (on the right side). We address code generation later.

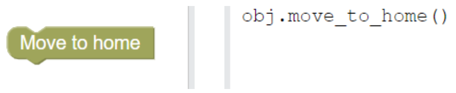
5) Movements blocks: The *Movements* blocks represent actions related to the robot's motion. When used, these blocks provide instructions for the robot to perform specific movement actions. Figure 8 shows one of the blocks belonging to this category; `move to home` makes the robotic arm to move to its home (default) position. In the Python



```
obj.connect('192.168.2.10')
obj.disconnect()
```

Fig. 7. Handling robot communication.

code, this operation is performed by invoking the method `move_to_home()` from our custom API. There are also blocks to move the arm to a given pose, and to grasp medications. These other blocks are illustrated in Section IV.

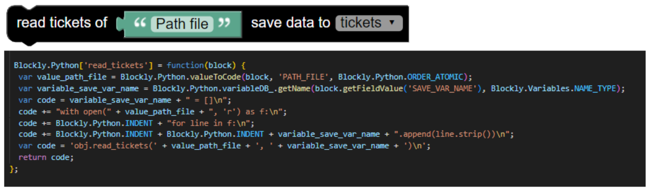


```
obj.move_to_home()
```

Fig. 8. A movement block.

6) Code generation: When one creates a block-based program in H-ArmDSL, this is automatically translated to a Python code, which is run on the actual robot. Therefore, H-ArmDSL provides an intuitive and accessible approach to programming robotic arms, enabling users to create Python scripts without the need for manual coding, nor understanding the underlying details of the Kinova Gen3 lite robot.

Figure 9 shows the function that generates Python code for `read tickets` blocks. The first two lines get from the block the path and the target variable (i.e., the variable the read data should be saved to).



```
Blockly.Python['read_tickets'] = function(block) {
  var value_path_file = Blockly.Python.valueToCode(block, 'PATH_FILE', Blockly.Python.ORDER_ATOMIC);
  var variable_save_var_name = Blockly.Python.variableDB_.getName(block.getFieldValue('SAVE_VAR_NAME'), Blockly.Variables.NAME_TYPE);
  var code = variable_save_var_name + " = []\n";
  code += "with open(" + value_path_file + ", 'r') as f:\n";
  code += Blockly.Python.INDENT + "for line in f:\n";
  code += Blockly.Python.INDENT + Blockly.Python.INDENT + variable_save_var_name + ".append(line.strip())\n";
  var code = 'obj.read_tickets(' + value_path_file + ', ' + variable_save_var_name + ')\n';
  return code;
};
```

Fig. 9. Python code generation.

The generated Python code is stored in `code`. Initially, the target variable comprises an empty list. The data associated with the file path is read and, for each line, the corresponding ticket information is retrieved and saved into the target variable. Finally, this function yields the value of `code`, which performs the expected computation. Although we target a specific database management system and robotic arm in this work, other databases and vendors can be considered by customising the code generation part.

IV. APPLICATION IN A HOSPITAL SETTING

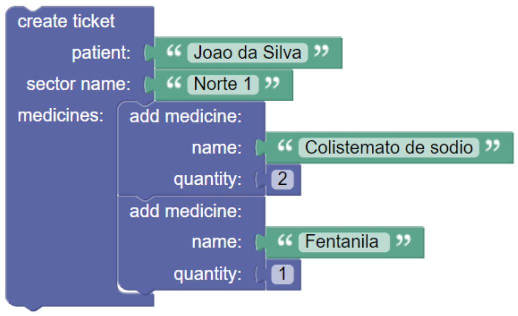
Hospital das Clínicas da Universidade Federal de Pernambuco (HC-UFPE) is a public reference hospital in Pernambuco, Brazil. The pharmacy of HC-UFPE serves a population of approximately 1 million people and is responsible for dispensing medications to both inpatient and outpatient individuals. One of the current main challenges faced by the HC-UFPE pharmacy is the creation of medication kits. These kits consist of medications that are frequently prescribed for the treatment of a specific condition. The manual creation of these kits is a labour-intensive and time-consuming process prone to errors. Moreover, according to regulation, kits involving controlled medicines can only be assembled by a chemist, since the access to these medications is restricted.

With the aim of enhancing the efficiency of assembling medication kits, first, of those involving controlled medications, we employed the Kinova Gen3 lite robotic arm and H-ArmDSL to automate such a process. This robotic arm, upon executing the requested movements, collects the medications and assemble the kits.

A typical kit is assembled by the robot in 2 minutes. Considering the average number of kits, the robot operates about 1 hour per day. The feedback provided by the hospital stakeholders is that our solution standardises the assembly process, and saves an important working time of a specialist (i.e., the chemist), who can now focus on other activities. In what follows, we provide more details on the application of H-ArmDSL to this case study.

A. *Creating tickets*

Tickets are created by different hospital sectors. Figure 10 shows how tickets can be easily created by using custom blocks of H-ArmDSL. The block `create ticket` creates a new ticket. It registers the patient name, the delivery location (i.e., the name of the hospital sector), and the associated medicines. Medicines are added to a ticket via the `add medicine` block. It provides the medicine name, as well as the quantity to be administrated to the patient.



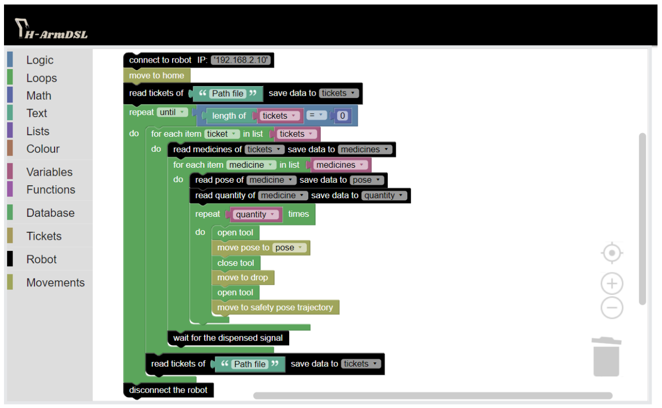
```
create ticket
      patient:  " Joao da Silva "
  sector name:  " Norte 1 "
  medicines:  add medicine:
                      name:  " Colistemato de sodio "
                  quantity:  2
              add medicine:
                      name:  " Fentanila "
                  quantity:  1
```

Fig. 10. Creating tickets.

When the user clicks on the *Execute* button (see Figure 3), the Python code generated from the block-based program is executed, updating the database accordingly. This signals to the central hospital pharmacy that a new kit needs to be assembled.

B. *Assembling kits*

Figure 11 shows the main program that runs on the central pharmacy; it is responsible for controlling the robotic arm to assembly medical kits. First, we stablish connection with the robot. Then, we ensure that the arm is located at the home (default and safe) position. Afterwards, the program reads the open tickets from the database and save this information into the variable *tickets*. The program keeps running until the length of *tickets* is equal to zero; in other words, until all open tickets are attended.



```
connect to robot  IP: '192.168.2.10'
move to home
read tickets of " Path file " save data to tickets
repeat until  length of tickets = 0
do  for each item ticket in list tickets
    do  read medicines of tickets save data to medicines
        for each item medicine in list medicines
        do  read pose of medicine save data to pose
            read quantity of medicine save data to quantity
            repeat quantity times
            do  open tool
                move pose to pose
                close tool
                move to drop
                open tool
                move to safety pose trajectory
        wait for the dispensed signal
    read tickets of " Path file " save data to tickets
disconnect the robot
```

Fig. 11. Assemblying medical kits.

For each ticket in *tickets*, we read the associated medicines and save them to *medicines*. For each medicine in *medicines*, we read the medicine's pose and quantity. Then, we have a loop to collect the correct quantity of the given medicine. This involves moving the robotic arm to the given pose, grasping the medication (i.e., opening and closing the grasping tool), moving the arm to the drop position (where the tray is located), and dispensing the medication by opening the grasping tool. Then, the arm is moved to a safety pose before moving on to the next medication in the current ticket; this is necessary to prevent the arm from colliding with the medication supports.

After collecting all medicines, the arm is moved to its home position, and waits for the dispense signal, which needs to be provided by a human indicating that the kit has been collected. Finally, the robot updates the *tickets* variable, since new tickets may have been created. This process continues until all open tickets have been attended. In Figure 12, one can see the robotic arm operating at the hospital environment, handling real ampoules (medications).

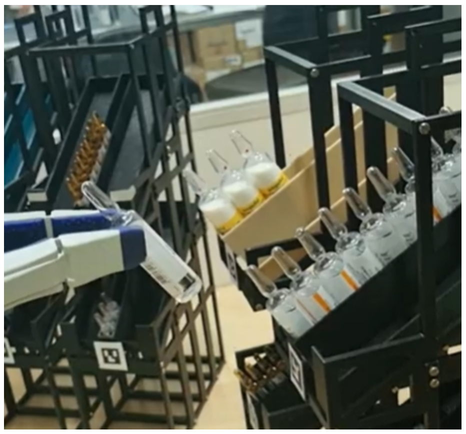


Fig. 12. Robotic arm approaching medicines for collection.

The successful execution of the generated Python code on the actual robot ensures the systematic collection of medicines, but also underscores the precision and reliability of the robotic system in real-world applications. Furthermore, this case study also serves as a real-world validation of the devised graphical domain-specific language for specifying missions of robotic arms (H-ArmDSL) for the healthcare domain. This innovative programming approach employs a series of intuitive blocks, each one representing distinct stages of the automated process involved in assembling medication kits.

V. CONCLUSIONS

This study addressed the imperative to simplify the programming of robotic arms, with a focus on the healthcare domain. We introduced a domain-specific language named H-ArmDSL, developed using a block-based programming approach with the support of Google Blockly. The study demonstrated an intuitive and accessible solution for robotic arm programming, with a notable application at the pharmacy of *Hospital das Clínicas da Universidade Federal de Pernambuco* (HC-UFPE).

The block-based visual programming approach supported through H-ArmDSL offers an effective alternative to overcome the challenges associated with the intricate programming of robotic arms. H-ArmDSL structure encompasses both default and domain-specific blocks, intuitively organized to simplify program creation, even for users lacking advanced programming knowledge. The seamless connection between visual programming and the generated Python code enables a smooth transition from design to execution.

In the HC-UFPE pharmacy case study, the manual creation of medication kits, particularly those involving controlled medications, was identified as a significant challenge. We have successfully devised a block-based program in H-ArmDSL that showcases the automatic collection and assembly of these kits, providing a practical and efficient solution for pharmaceutical demands.

A. *Related work*

There are other domain-specific languages for specifying missions of robotic applications. In [7], the authors compare and contrast some of them. In what follows, we briefly present some of these other languages, and highlight the distinguishing aspects of H-ArmDSL.

- NaoText [8] is a DSL created by the QualiTune research group. This function-based language allows specifying missions for NAO robots through a textual notation. From our perspective, a visual block-based programming approach, as the one provided by H-ArmDSL, provides more abstraction, enabling the development of robotic systems to a broader public.
- MissionLab [9] is a DSL created at Georgia Tech that facilitates mission specification through a visual language based on state machines. The DSL uses assembly and temporal sequencing constructs to create a mission as a chain of behaviours. Differently, H-ArmDSL considers a visual block-based programming approach for specifying missions.
- Open Roberta [10] is a web-based educational DSL developed by the Fraunhofer Institute. It is free for individual use but requires payment for institutional application. This DSL, also built with Blockly, allows programming various types of robots. It can run in the cloud or be installed locally on a server. Open Roberta generates code in Python, Java, Javascript, and C/C++, depending on the target robot. However, differently from H-ArmDSL, it is not tailored for the healthcare domain.

B. *Future work*

There are many opportunities for future work. In the near future, we envisage addressing the following ones. We are working on using computer vision to identify the medications and, thus, do not rely on predefined poses. To this end, we have integrated a camera to the robotic arm. In order to automate the delivery of medical kits to the different hospital sectors, we also plan to integrate our current solution of creating medical kits with mobile robots. To facilitate the development of such integrated applications, new blocks are to be devised to H-ArmDSL. Finally, in collaboration with our partners, we want to explore new applications of H-ArmDSL; a candidate application concerns the manipulation of sterile equipments also in a hospital setting. To foster further applications of H-ArmDSL, we are about to release it to the public domain.

REFERENCES

[1] A. C. Simões, A. L. Soares, and A. C. Barros, "Factors influencing the intention of managers to adopt collaborative robots (cobots) in manufacturing organizations," *Journal of Engineering and Technology Management - JET-M*, vol. 57, no. May, p. 101574, 2020.

[2] Blockly. Accessed in March, 2024. [Online]. Available: https://developers.google.com/blockly/

[3] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, p. 26–36, jun 2000.

[4] A. C. Jiménez, J. P. Anzola, V. García-Díaz, R. González Crespo, and L. Zhao, "PyDSLRep: a domain-specific language for robotic simulation in V-Rep," *PLoS ONE*, vol. 15, no. 7, p. e0235271, 2020.

[5] Kinova® Kortex™ firmware update v2.3.2 release notes, Kinova Gen3 lite robot users. Accessed in March, 2024. [Online]. Available: https://www.kinovarobotics.com/product/gen3-lite-robots#ProductSpecsOverview

[6] C. S. Crawford, M. Andujar, F. Jackson, I. Applyrs, and J. E. Gilbert, "Using a visual programming language to interact with visualizations of electroencephalogram signals," in *ASEE-SE Annual Meeting*, 2016.

[7] S. Dragule, S. G. Gonzalo, T. Berger, and P. Pelliccione, *Languages for Specifying Missions of Robotic Applications*. Cham: Springer International Publishing, 2021, pp. 377–411.

[8] S. Götz, M. Leuthäuser, J. Reimann, J. Schroeter, C. Wende, C. Wilke, and U. Aßmann, "A role-based language for collaborative robot applications," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2011, pp. 1–15.

[9] R. Arkin, "Missionlab: multiagent robotics meets visual programming," in *Working Notes of Tutorial on Mobile Robot Programming Paradigms, ICRA 15*, 2002, p. 745.

[10] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open Roberta - yet another one?" in *2014 IEEE International Symposium on Multimedia*, 2014, pp. 381–386.