IT Security Master Program
Department of Economic Informatics and Cybernetics
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest University of Economic Studies

Dissertation Thesis

Coordinator
Assoc. Prof. Toma Cristian Ph.D.

Master Program Student
Milea Carmen

Bucharest 2021

IT Security Master Program
Department of Economic Informatics and Cybernetics
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest University of Economic Studies

Industrial Robots/Cobots Security Framework

Dissertation Thesis

Coordinator
Assoc. Prof. Toma Cristian Ph.D.

Master Program Student
Milea Carmen

Bucharest 2021

Statement regarding the originality of the content

I hereby declare that the results presented in this paper are entirely the result of my own creation unless reference is made to the results of other authors. I confirm that any material used from other sources (magazines, books, articles, and Internet sites) is clearly referenced in the paper and is indicated in the bibliographic reference list.

Contents

List of Figure

Chapter 1

Introduction

This chapter aims to provide some overview information about the topic and problem this thesis addresses. Moreover, similar existing products and specialty papers are analyzed and compared with the current solution. The scope of this section also includes details about the industrial robots' niche and its impact.

1.1 Overview

To begin with a short introduction, in [1] we notice that the term „robota" means, in some Slavic languages, slave labor/mundane work. Moreover, in the story of Rossum's Universal Robots (R.U.R.) [2], the robots are portrayed as the humans' servants, lacking any trace of intelligence or affection. By the end of the story, the robots start a revolution and overtake the whole world.

So, are robots that dangerous and terrifying? Will they be the source of unemployment for many people? Could we end up in a robots-world? Will technology do more harm than good to humankind? These are some philosophical questions regarding robots that have uncertain answers.

It is true that some people perceive (some still do) robots as destructive and dangerous. However, if we take a closer look at nowadays facts, we notice that, in most of the cases, robots are rapidly taking over dangerous, monotonous, heavy tasks. Moreover, they don't need paid time off, rest, medical leave and are able to perform faster and better at certain tasks.

Personally, I do not believe that robots will be able to steal all the jobs from humans (at least not in the foreseeable future). However, I think that most/all the jobs that do not require any creativity can/will be done by robots. I believe that, in the future, people will only have jobs that involve their brain, and not their body.

Moving back to the industry nowadays, FANUC is an important brand, being the largest maker of industrial robots in the world. Their business is present on three continents: North America, Europe, and Asia. FANUC is an acronym for Fuji Automatic Numerical Control [3].

Even though the FANUC robots are heavily used in the industry, little information has been provided on the technical aspect of these robots. Despite its importance, there is little to nothing

at all in terms of tutorials/research providing information on hardware setup, network communication, software creation and actual robot usage.

A possible problem that we have identified within the FANUC environment is that there is no built-in way of remotely controlling those robots. This issue implies the physical constraint of being near the robot in order to load or run programs. We believe that this behavior does not match the latest technological advances that are focused on accessibility through the internet and that this issue should be addressed.

Given all the above facts, this thesis aims to propose a project that handles the remote-control issue. The end-result solution consists of a web portal through which users can issue various commands and a communication pipeline between the web server and robot. One of our main goals is to obtain a solution that is secure in every possible aspect, considering the harmful impact of potential attacks. Moreover, this thesis contains information about the core components and the process of setting up the FANUC robot, establishing a network connection to it, and creating custom Karep/TP programs.

Given the fast paced technology development that increasingly relies on the internet, corroborated with the business needs that depend on robots, we consider that this thesis is anchored in the present through providing a way of interconnecting those two aspects.

1.2 Similar products

Even though the product developed alongside this paper is within a niche, we did manage to find some relatively similar pieces of software (that are related to the remote access/control of the robots).

1.2.1 FANUC ZDT

The first similar product is developed by FANUC itself and it is called FANUC ZDT – Zero Down Time.

We did not manage to test this software hands-on. However, by analyzing [4], we noticed that the purpose of ZDT is to provide useful equipment monitoring solutions. With ZDT, robots and

process equipment are monitored, robot program and process changes are tracked, operational insight reports are provided and component failure is predicted before downtime occurs.

The information gathered by ZDT can be accessed through a website (similar to our product). However, the product is advertised as a secure outbound only connection, therefore, the end user can only see reports about their robots' and processes' health and status, but not trigger any commands or change configurations.

Consequently, when compared to ZTD, our project stands out due to the fact that it supports inbound connections, therefore, the user can send data to the robot and vice-versa.

1.2.2 DATATOOL

The second similar product is called DATATOOL and is developed by ONE ROBOTICS. Even though this product hasn't been launched yet, we did manage to find out some key information about its use cases and purposes.

[5] presents that this piece of software is designed in order to allow remote robot's management through a website. The main focus of this product is configuration oriented, therefore, it handles the altering of data (numeric registers, position registers, flags, user alarms) and inputs/outputs (digital, analog, user, group, robot).

Therefore, what makes our current project different from DATATOOL is that it supports robot movement triggering, apart from configuration data manipulation.

1.3 Literature review

This sub-section aims to provide insights into the literature on the subject of industrial robots and FANUC. As already mentioned, the exact niche of this paper seems not to be covered by other researchers. However, we will comment on adjacent or more general topics.

As a general definition of robots, we found that Isaac Asimov [6] came up with three core laws describing them (the so-called, „Three Laws of Robotics").

1 „A robot may not injure a human being or, through inaction, allow a human being to come to harm." [6]

2 „A robot must obey the orders given to it by human beings except where such orders would conflict with the First Law.” [6]

3 „A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.” [6]

Even though these laws were written in a science-fiction book, the Engineering and Physical Sciences Research Council (EPSRC) came up with a set of similar laws, entitled „principles for designers, builders and users of robots”. These fundamentals aim to describe the perfect robot: safe, lawful and transparent.

In terms of FANUC related research, the authors of [7] propose an interesting approach to controlling the robot. Instead of traditional controls, such as joysticks or buttons, the authors present an innovative manipulation method: vision and gesture recognition using Microsoft Kinect Motion Controller. This system seems to even be able to extend to augmented reality and voice controllers. Even though this solution has its drawbacks, the system seems to have a small overhead (employees not having to undergo long lasting training) and it is also extensible.

Continuing with the same niche, Norberto Pires reports in [8] the findings of human voice controlling of a robot. This demonstration is done using two robots and one computer. The computer has a microphone that enables voice controlling; thus, the robots execute picking-and-placing objects movements. The application seems to mostly use Microsoft Technologies (Microsoft .NET Framework, Microsoft Speech SDK). The paper seems to focus on voice recognition and text-to-speech translation.

In [9], the authors explore the security analyses of an industrial robot controller. The paper theoretically and practically presents an approach of gaining unauthorized access over a robot controller and the possibility of tampering with it.

The paper underlined the fact that maintenance/security patches might have a lower priority for robot controllers since they would be equivalent to unacceptable periods of downtime (that further translates into money loss). This becomes an exploitation window, increasing the chance/impact of an eventual attack.

According to this paper, most of the issues come from the fact that the robots in warehouses are interconnected and connected to the internet for programming and maintenance purposes. This increased connectivity is, obviously, prone to expose robots to different types of attacks.

Moreover, the report mentions that, even though, initially. The robots were meant to be located in protective cages, nowadays, cobots (collaborative robots) have been introduced. Being able to work nearby humans, an attack would possibly mean putting people in danger.

Additionally, the authors used search engines such as Shodan and ZoomEye to try and find exposed robot controllers. Surprisingly enough, they did find three publicly available controllers, one of which not having an authentication system at all.

1.4 Impact

This subsection aims to underline the importance and increased usage of FANUC and other industrial robots nowadays.

In [10], we notice that Tesla, one of the leading companies in the automotive industry, is extensively using FANUC and Kuka robots in their warehouses. [11] shows a video of them in action.

We notice that [12] presents the usage of FANUC industrial robots even in the case of the COVID pandemic. [13] shows a video of a FANUC Scara robot loading swabs and test tubes.

[14] presents the opinion of a FANUC representative over the translation of downtime of robot controllers in money: „Unplanned downtime is a worst-case scenario for automotive manufacturers. In a factory that's kicking out one vehicle per minute, every minute of stalled production is hemorrhaging profits, labor expenses and more. Unplanned downtime can cost as much as $20,000 potential profit loss per minute, and $2 million for a single incident." profit loss per minute, and $2 million for a single incident."

Therefore, we see that industrial robots have a significant presence in leading companies and day-to-day news. Moreover, it has a significant role in multiple aspects of our lives.

Chapter 2

Architecture

The purpose of this chapter is to establish the project's architecture, main physical/logical components and used programming languages.

2.1 Physical environment components

An important step into the development phase was to get familiar with the FANUC robotic arm physical components, their roles and how they interact.

The environment used for building the end product is the Educational Package cell offered by FANUC. Figure 1 gives a visual representation of the cell. It's content is highly resembling the technologies used in production (warehouses) and it is a useful way of allowing students to gain hands-on experience of interacting with the industrial robots world. More details regarding the Educational Cell can be found at [15].

Figure 1 FANUC Educational package [16]

To give a brief physical description of the cell itself, we are presented with an aluminium protection frame that has a safety switch access fence. The top half is represented by a tempered see-through glass that holds the robotic arm itself, while the bottom part is aluminium and covers the controller.

2.1.1 Robotic Arm

A main component of the physical environment is, of course, the robotic arm itself. The Package includes the FANUC ER-4iA robot. In the Educational cell, the robot has a pre-mounted electric gripper that comes in handy when carrying various objects.

It is considered a „Mini Robot" because, when compared to its industrial counterparts, it has some limitations in terms of height, maximum payload at wrist (4 kg), handling and assembling.

The advanced technologies that FANUC ER-4iA is based on, allow a high-speed smooth motion, while being dust and waterproof. It is equipped with a range of smart features, such as: collision detection and smart link (that allows two or more robots to work together).

2.1.2 Controller

The robotic arm is powered by the controller component, in this case the R-30iB Mate Plus Controller. This controller comes in five different cabinets (sizes), so that a suitable version of it can be found given any physical constraints.

Two main core functions of this component are connection to power and ethernet. Moreover, it is equipped with a high speed CPU that handles process optimization. More details into this component can be found at [17].

2.1.3 Teaching Pendant

The third item included in the Educational Package is a custom touchscreen device, called the Teaching Pendant or iPendant. This gadget represents a user interface that allows configuring

settings, loading programs into the robot's memory and running programs. A tutorial on how to get started on the TP can be found at [18].

The TP allows the addition of custom HTML screens and supports multi-window displays in order to provide extensive information. This device is extremely useful when it comes to developing programs. It has an intuitive and user-friendly interface that allows TP coding. Even though the programmer is not in a classic development environment (with a desktop and keyboard), TP is packet with predefined shortcuts that allow the creation of boilerplate code within seconds.

It is equipped with an USB port that allows doing robot backup or loading/copying files. However, the USB stick must be FAT32 formatted in order to be recognized by the TP.

2.1.4 iRVision

A fourth and last component of this kit is represented by iRVision. This component is basically a camera that allows object training and identification which comes in handy in real life scenarios as the robots are supposed to lift and assemble various objects. [19] presents this component in depth.

iRVision removes the need of using jigs and allows colour sorting, shape identification and bar-codes scanning, no matter the object's position or orientation (as long as it is placed within the camera's range). The camera can be calibrated and new objects can be trained using the TP or the robot's web interface.

2.2 Project architecture

Considering the end-goal result of obtaining a way of remotely controlling the FANUC robot, we might have some hints that the project's architecture should be split into two main layers separated by the host environment (desktop and robot).

Figure 2 contains an overview of the project layers and architecture.

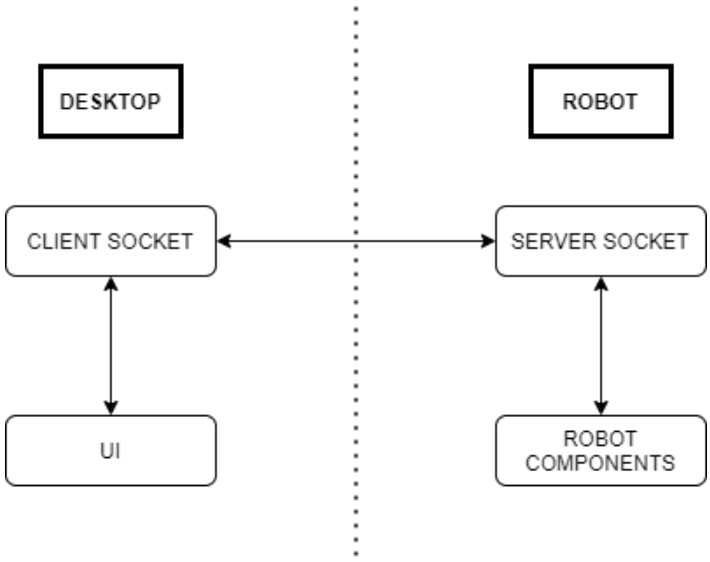

Figure 2 Project architecture

As can be noticed in Figure 2, the application consists of two logical layers, when considering the host machine: desktop and robot.

The machine through which the remote control of the robot is achieved, has two main components. The first one is a web interface containing forms and buttons that allow the user to define and trigger different commands to be executed. The client socket is a second module in the desktop layer. It's primary goal is to capture and parse the instructions initiated by the user and communicate with the server socket.

A second main area of the project is defined by the components that are running on the robot. The server's socket main objective is to receive and interpret commands from the client socket. In addition to that, this unit also retrieves certain data in order to pass it back to the client, configure settings to a certain extent and initiate actual robot movements.

2.3 Used technologies

In order to achieve the end-result project, we've used multiple technologies and frameworks, each one being specific to its host platform and general purposes.

2.3.1 Desktop Layer

For the desktop logical layer, we chose C# as our primary language. The web application follows the Model-View-Controller (MVC) approach. To be more specific, we went for ASP.NET MVC Core due to its cross-platform characteristics and robustness. A useful documentation for this framework can be found at [20].With this design paradigm, the separation of concerns is achieved: models store the business logic, views display the information to the end users and controllers capture requests and map them to application actions.

A particular useful characteristic of .NET is that it supports the Dependency Injection (DI) technique. This approach aids in achieving Inversion of Control (IoC)

We chose Entity Framework Core as the Object-Relational Mapper. We went for the Code First approach, in order to have more control over the generated code and to be able to focus on one programming language only: C#. This means that every table in the DB has a corresponding model in our project, called an entity. As a means to decoupling the business and data access layers, we chose the Repository and DbContext pattern. In our project, C# queries are translated into SQL code using Language-Integrated Query (LINQ).

For authentication, authorization and user information storage, we chose Identity, an API provided by .NET that is specialised in user management. This API's documentation can be found at: [21]. We are therefore provided easy-to-use specific methods that allow user creation, connection/disconnection, while allowing us a certain degree of configuration (whan user information is stored, password strength, etc.)

Views contain regular UI/frontend languages: HTML, CSS, JavaScript, JQuery. As can be seen in the views' extension (**.cshtml*), these files contain C# code, apart from the usual frontend languages. This comes in handy in numerous cases, such as displaying particular information to certain roles only or mapping the models' properties

The TCP client socket has also been implemented using C#, in order to aid the communication between the web project and the robot. C# has a suite of built-in socket related classes and services that we've extensively used. The documentation for tcp client can be found at [22].

2.3.2 Robot Layer

For the robot module, we've used the two possible programming languages that are specific to a FANUC environment, each holding its own purposes and capabilities.

The server TCP socket component of the robot is implemented using Karel. This programming language can be considered similar to Pascal and it aids in accomplishing the end goal: performing industrial tasks. It has built in methods of working with TCP sockets, which were heavily used in the project. Karel human-readable ASCII files have the extension *.kl* and, after the file is compiled, the resulting file has the extension *.pc* (ROBOGUIDE software has a built-in Karel compiler). One limitation that Karel has is that it cannot trigger actual robot movements. [23] is an extensive Karel manual that gives insights into this language's capabilities.

A second programming language used on the Robot component is Teach Pendant (TP). It's name resembles the name of the device used to control the robot because, by using it, TP programs can be developed, previewed and debugged (unlike Karel programs). By the looks of it, TP can be considered an assembler-like programming language. It's main purpose is to trigger robot movements. Therefore, by pairing TP with Karel, a complete environment can be achieved; therefore, we have the means of both doing movements and communicating through TCP sockets. Some useful information regarding TP programming can be found at [24], while [25] and [26] offer practical examples in the TP spectrum.

Chapter 3

Implementation

This section provides core details about the physical environment setup, database schema, project use cases and security related aspects.

3.1 Environment setup

Given that the end goal of the project is to be able to remotely control the robot, a first step that needs to be done is connecting it to a network.

Once the robot is connected to power and in an up-and-running state, we can make use of the controller's Ethernet port and connect it to a network. Then, using the Teaching Pendant, we need to configure the network related settings (e.g.: IP, Default Gateway). Once this step is done, there should be successful pings when trying to connect from a computer in the same network to the robot and vice-versa.

In order to facilitate the development process, by removing the need to be physically near the robot in order to test and debug the code, we made use of the ROBOGUIDE software provided by FANUC. Some useful information about this software can be found at [27].

ROBOGUIDE allows the creation of virtual FANUC working environments. This step can be achieved by either manually specifying the environment's components and settings or by using a real-life robot backup. Since we want to create a virtual working environment that is identical to the physical one, we chose the second option. [28] represents a tutorial containing useful information on the process of backing-up FANUC robots.

3.2 Database Schema

Given that the primary role of our project is to trigger reboot directed commands, the database schema is not very complex, nor does it have a large number of tables.

However, having a database is not optional as there is information that must be stored, such as the users information. Figure 3 displays the said database schema.

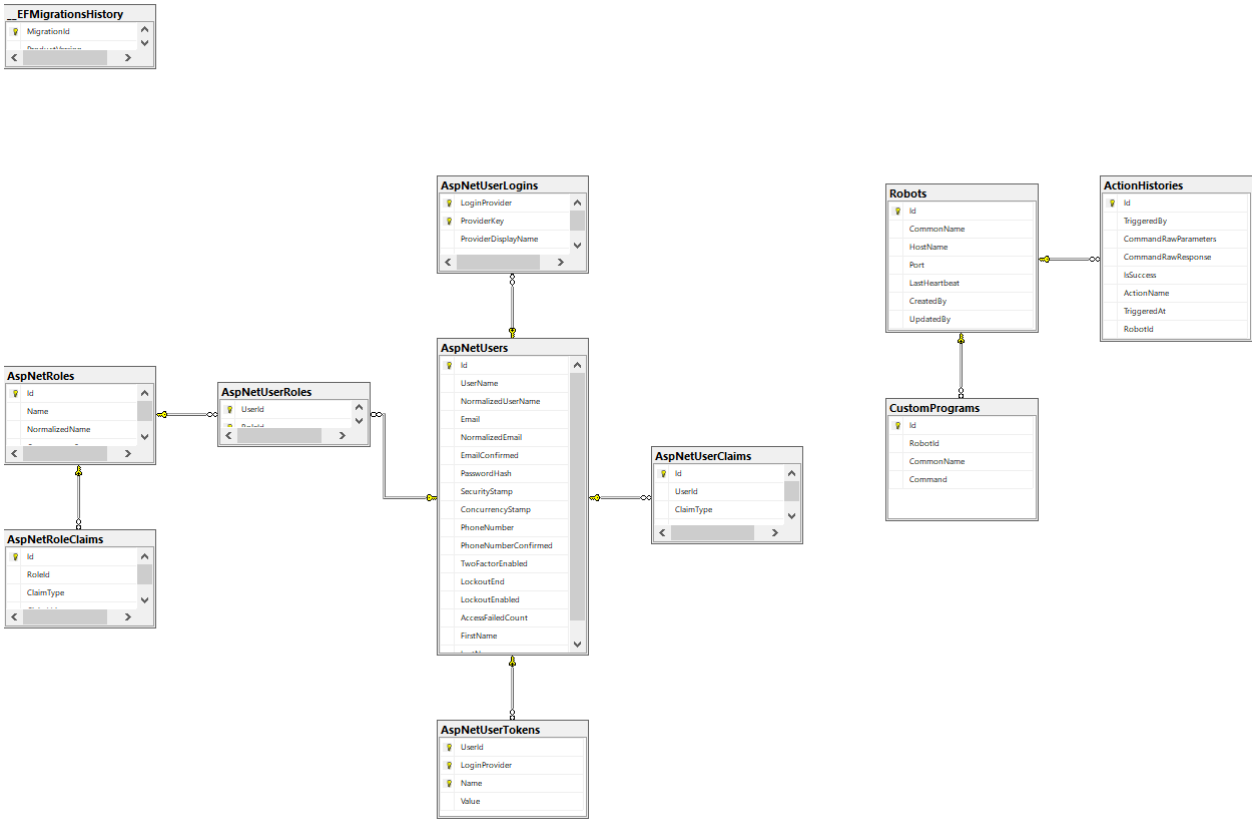

Figure 3 Database Schema

We notice that there are seven user related interconnected tables, *AspNetUsers* being the main one. These are auto-generated by a migration due to our option for using Identity for user manager. Therefore, Identity uses these tables for certain operations, such as registration, connection or authorization. Even though we don't have control over these tables, we did manage to modify the initial schema of *AspNetUsers*, by adding the *FirstName* and *LastName* columns. This operation took place in a Code First manner (like other instances in the application); therefore, we first changed the corresponding User class in code, then we ran the generated migration.

__EFMigrationsHistory is another auto-generated table. This time, EF uses it as a means of keeping track of the migrations that have already been run. Each time a new migration is run, it is logged in this table, so that EF prevents running a second time.

Robots is a table that, as the name suggests, stores robot related records. In terms of this table's columns, we have *HostName* and *Port*, which represent required information necessary for sending

messages to the robot. We also decided to add the *CommonName* as a means of associating the port and host to a suggestive robot name. *CreatedBy* and *UpdatedBy* are audit columns that track the users that make changes to a robot record.

ActionHistories table keeps track of the interaction that certain users have with robots. This table stores information about the action name, inbound and outbound parameters, associated robot, timestamps and whether or not the request had a successful response.

CustomPrograms is a table that allows storing what specific actions a robot can perform. All robots have a set of common programs that they support (such as changing the robot current position). However, through this table, the users can add a series of extra commands. These are characterised by a *CommonName* (that will be displayed in the UI), a *Command* (which will be passed to the robot through TCP sockets) and the corresponding robot.

Neither *Robots*, *ActionHistories*, nor *CustomPrograms* tables are required for the end-goal of being able to remotely control the robot. However, they come in handy because the user doesn't have to specify the host and pot with every request and the values are not hardcoded in the application.

3.3 Implementation Details

As mentioned before, the end-result project consists of a .NET MVC web application, TCP socket communication channel and Karel/TP programs. This sub-chapter aims to provide implementation details about all these components.

3.3.1 Web Application

By opting to use ASP.NET MVC as the framework for building the web platform, we were offered the classic project architecture for such projects. We also customised certain areas to make sure that they accomplish certain specific goals.

wwwroot is the first section of the project, and it is part of the web project template. It's primary goal is to store any frontend related files. Therefore, it is the go-to place for the *css* and *js* files of

our application. It also contains additional frontend libraries; in the current project these are: *bootstrap*, *font-awesome* and *jquery*.

Controllers is the folder that, as the naming indicates, stores the units responsible for the request capture and parsing. These controllers are differentiated by the business logic they handle. For example, we chose to have a dedicated *AccountController* whose primary role is to drive user related operations. We tried to have as little business logic as possible in the controllers and shift it into services. Therefore, the actions in the controller are lightweight and easy to interpret as they just pass and receive data to and from services, data that later gets to the view. Services are injected in controllers through DI.

Data is a custom folder that we added in order to have an organized place for classes that imply database interactions. *FanucDBContext* is the primary class as it represents the DB equivalent in code. This class extends *IdentiyDbContext*, which aids in the user related tables creation in DB. It has *DbSets* corresponding to every table in the DB. There are two additional files included in this folder: *IRepository* and *Repository*. These interfaces and implementations are generic classes that store methods for the basic CRUD operations. Therefore, these are used for data retrieval or manipulation. We chose to go for this approach in order to be able to inject it into services through DI and to prevent repeating the same code for every entity.

Migrations is the folder through which EF stores the DB related updates: migrations. Each time we add a code change that needs to get translated into SQL, we create a new migration. With the *update-database* command, the new migration is run and the end result is obtained. This folder also has a *ModelSnapshot* class that, as the name implies, keeps information about the DB state.

We decided to split the next folder, *Models*, into multiple subfolders in order to have a better organized architecture. Therefore, the first subfolder is named *Entities* and it stores classes that exactly match the DB tables. We decided to have a *BaseEntity* that the entities extend for two primary reasons: all the tables have an Id, so that could be included in the base class, and to aid in the Repository template. We also added three additional subfolders: *Fanuc*, that stores classes related to robot command, *Utilities,* that contains helpers and *ViewModels,* that includes classes used exclusively in views.

Services is the container for classes that follow the business logic. It is divided into two further folders: *Interfaces* and *Implementations* that aid in DI. The services are used by controllers and use models and repositories.

The *Views* folder contains elements that are displayed to the end user. In terms of structure, each controller has a corresponding folder in the *Views* folder and each action is associated with a *.cshtml* file. In both cases, the naming is essential and should be identical. This folder also contains some particular elements. For example, *_Layout* contains some common UI elements that are shared among all pages (the navigation bar). *_ViewImports* is a file that contains library references that are used by all application pages.

In terms of the primary feature of the application (inbound and outbound communication with the robot), we've made use of *System.Net.Sockets* utilities: *TcpClient* and *NetworkStream*. The classes help us in establishing the connection and sending/receiving messages.

Regarding the content of the messages sent through the pipeline, we needed to establish some patterns that both the web application and robot followed. For example, we've established that the command that gets the value of a register should match this template: RX, where X is the register index. Therefore, the robot must be able to interpret this request and return the corresponding value, just like the web application needs to correctly format the request. Likewive, we established that the response for a register should follow this template: X;Y;Z;W;R;P, where the variables represent the position register coordinates (three axes and the rotation for each axis).

As will be noticed in the following subchapter, the messages are sent in plaintext through the pipeline. We will further provide details on why we have this behaviour and how this affects the security of the platform.

3.3.2 Robot

As mentioned in the previous chapter, the two main languages used for developing the solution from the robot perspective are Karel and TP (those two also being the only available languages to use).

Karel was used for building the main part of the application. It is a single file and it has three main purposes. The first one is to assure the creation and starting of the TCP socket server. We need to make sure that the host name and port on which the robot is actively listening matches the information in the DB about that robot.

A second core feature of the Karel component is to detect oncoming requests, to grab the request body and, in the end, close the connection. These actions are performed using the following built-

in functions: *MSG_CONNECT*, *BYTES_AHEAD*,*READ* and *MSG_DISCO*. Since we want the TCP server socket to be continuously listening for requests, we needed to include the main code

into a *WHILE* statement (that has a *BREAK* exit for when we need to stop the program). Once the program is in an up-and-running state, the program execution will stop at the *MSG_CONNECT* instruction until a request is received. Once that happens, the execution will continue, the message will get processed and the connection will be closed with *MSG_DISCO*. The program execution will then stop at MSG_CONNECT once again and the program will be ready to receive new requests. If, during a message processing, another message is received, the second client will get an error, but the robot will continue to process the initial message.

The third core feature of this component is the message interpretation, processing and response. As mentioned before, the template established for requests and responses should be consistent and followed by both the client and server.

For the message processing phase, Karel can access various information about the robot such as current position or position register values. This information is useful for certain requests type (such as getting the current robot position), but in insufficient for other requests (like moving the robot to a certain position). This is caused by the Karel inability of triggering robot movements.

However, robot movements are the most useful features of the TP language. Therefore, Karel can call TP programs in order to obtain the end result. The main way of data transmission between Karel and TP is through Position Registers.

The way the robot gets moved to the position indicated by the user is a useful and practical example of this TP-Karel communication. First, Karel receives and parses the message and then it creates an object to store the coordinates. We then check if the coordinates make up a valid robot position (that is within reach). Once the validation passes, Karel stores the coordinates into a position register (we chose PR1). The last step that Karel does is to call a TP program that has one single line. That line can be explained as „move robot to PR1”.

There can also be multiple-line, more complex TP programs to which Karel might or might not send parameters. With TP, we can specify the coordinates the robot should move to and the speed and way the movement to be done. We are also able to identify objects by accessing the iRVision capabilities and perform custom actions for each object type.

In order to have the environment setup, we need to upload the compiled Karel file and to start it. In the Teaching Pendant device, we should see the status for the program at running and to notice

that the program is stopped at the line corresponding to *MSG_CONNECT* , waiting for new requests.

3.4 Security aspects

As stated in the incipient part of this paper, the main goal of the project is to provide a secure means of remotely controlling the FANUC robot. This implies three branches that should be secured (presented in the following sub-chapters).

3.4.1 TCP communication security

The TCP communication channel between the robot and the web server should be secured in order to make sure that an alleged attacker wouldn't be able to eavesdrop or replay messages. Usually, TCP socket communication security is done through SSL.

However, even though we understand the necessity of securing such a channel, we faced some challenges in implementing this end goal.

To begin with, Karel, the programming language used for establishing the TCP server socket, does not have any built-in security/cryptography related functionalities. Moreover, it has very limited bitwise operations, is only 32-bit and does not support unsigned integer types. All of these make Karel a language with which any modern cryptography algorithm is quite difficult to implement.

Even [23], the Karel manual we used extensively throughout developing the project as the main source of documentation, does not have any reference to Karel security. The only time security is mentioned in this manual is to specify that „FANUC Robotics does not support any security or encryption features on USB memory sticks".

We continued our research regarding security in Karel by analysing existing papers, articles and speciality forums. Unfortunately, this research concluded to be close to no new information on this topic.

We decided to contact Jason Strybis, the founder of ONE ROBOTICS [24] (a website cited extensively throughout this paper and very useful in the development phase), to have a more experienced opinion of this topic.

Jason Strybis was kind enough to help us with a reply. However, this did not shed any light on our issue. To summarize Jason Strybis' response, he mentioned that he looked into implementing the SHA-1 hash algorithm when trying to develop websockets in Karel, but did not manage to accomplish it. He also underlined the Karel limitations when it comes to core implementation needs for anything security related. He recognised the importance of such characteristics in Karel and mentioned that, from his point of view, FANUC chose not to focus on cryptography in Karel as the robots are typically on isolated network environments and therefore, network security is not a concern.

We've also started a thread related to Karel socket security on one renowned robotics forum, [29]. Unfortunately, the answers received confirmed our previous findings. Most of the responders said that, usually, robots are air-gapped from the Internet, the only interconnection being done through a single computer that provides protections such as firewalls and port filtering.

Therefore, the end-result project doesn't offer any TCP socket encryption at this step due to the Karel language limitations. We will also consider an environment similar to production industrial environments: the robot is isolated from the internet and the gateway server provides extensive security.

3.4.2 Web Server Security

As we have noticed in the previous sub-section, the security of the web server is crucial, given that the TCP socket communication security is not reachable. As this server represents a gateway between the robot and the outside world, we need to make sure that all possible protections are provided.

The first thing that needs to be done is firewall port filtering. We need to make sure that port-based network protocol packets are scanned and blocked according to their destination port. The port that needs to definitely be allowed is 443 (specific to https). We then need to allow other ports that might be necessary and limit/deny the remaining ones.

At the firewall level, we can also do IP filtering. This way, we can setup an IP whitelist, thus making sure that only certain people are able to access the website.

Another important step that needs to be established is changing the default connection to the server – done through a password. A better alternative is to use SSH with a passphrase (based on a RSA key pair). SSH by itself has an additional layer of security when compared to regular password login, but we also chose to add a passphrase to it. Once this step is done, we need to disable the regular password login.

We need to block ICMP redirects as it prevents man-in-the-middle attacks. Also, since the server is not a router, we prevent sending ICMP redirects and accepting IP source route packets.

We should also have our website use IPv6, as it makes IP spoofing more difficult due to additional layers of encryption and authentication.

Apart from all these web server security steps, we need to continuously monitor the server and react to suspicious activity, while also making sure that we do all the OS updates/patches.

3.4.3 Web Application Security

The third security component in our environment is represented by securing the web application itself.

We need to make sure that it is not susceptible to DB injection attacks. In our case, we use Entity Framework for DB mappings and the framework itself is capable of protecting against such attacks.

Apart from that, we need to protect against Cross Site Scripting attacks (XSS), by sanitizing the users' input. We also need to make sure that we protect the forms against Cross Site Request Forgery (CSRF); this is accomplished by using anti-forgery tokens (.NET has a good coverage for this).

We also need to make sure that there are no endpoints that can be accessed by anonymous users (apart from the login form). All routes must be protected by either authorization or authentication.

Last, but not least, our DB should not store passwords in plain text. For this case, we used Identity provided by .NET that handles the users' passwords hashing.

3.5 Use cases

Before getting to the use cases details and demo, the first thing that is essential to know is that the web application does not have a register option, at least not for unauthenticated users. Only users that have an administrator role are able to add either regular users, or other administrators. That means that the root administrator's information needs to be manually seeded into the database before being able to use the web application.

Once the user passes the authentication step (using the email/password model), they are redirected to the main page of the application. This page's goal is to display an overview of the available/ready-to-connect robots. The datasource table for this page is the *Robots* table in the database. Therefore, each line in this grid presents the core information about the robot: common name (an intuitive way of identification), the host name and port (information used when sending requests through the TCP communication channel.

This page also allows administrator users to add new robots. By hitting the *Add New Robot* button, they are redirected to a three input form (for common name, host and port) that, once submitted, performs this action.

Each line in this *Available Robots* page has a Details button that redirects the user to a page with more in-in depth information about that specific robot. Figure 4 presents the design of the *Available Robots* overview page. As mentioned before, only administrators are able to add new robots; therefore, the *Add New Robot* button is hidden for non-administrator (that corresponds to that action is also secured).

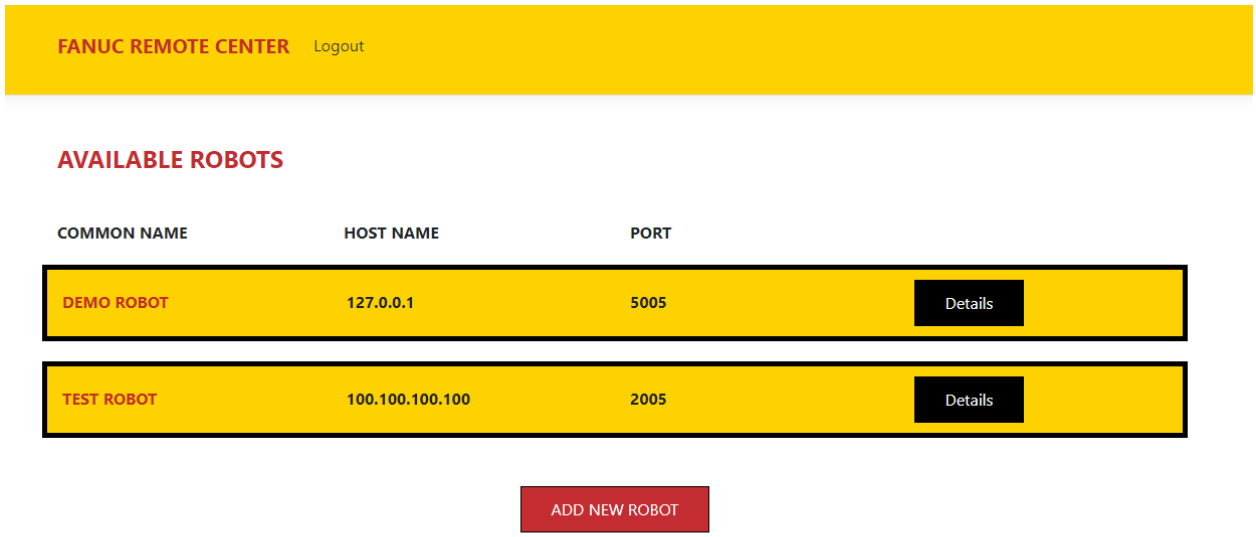

Figure 4 Available Robots Page

The *Details* page the user gets redirected to is displayed in Figure 5.

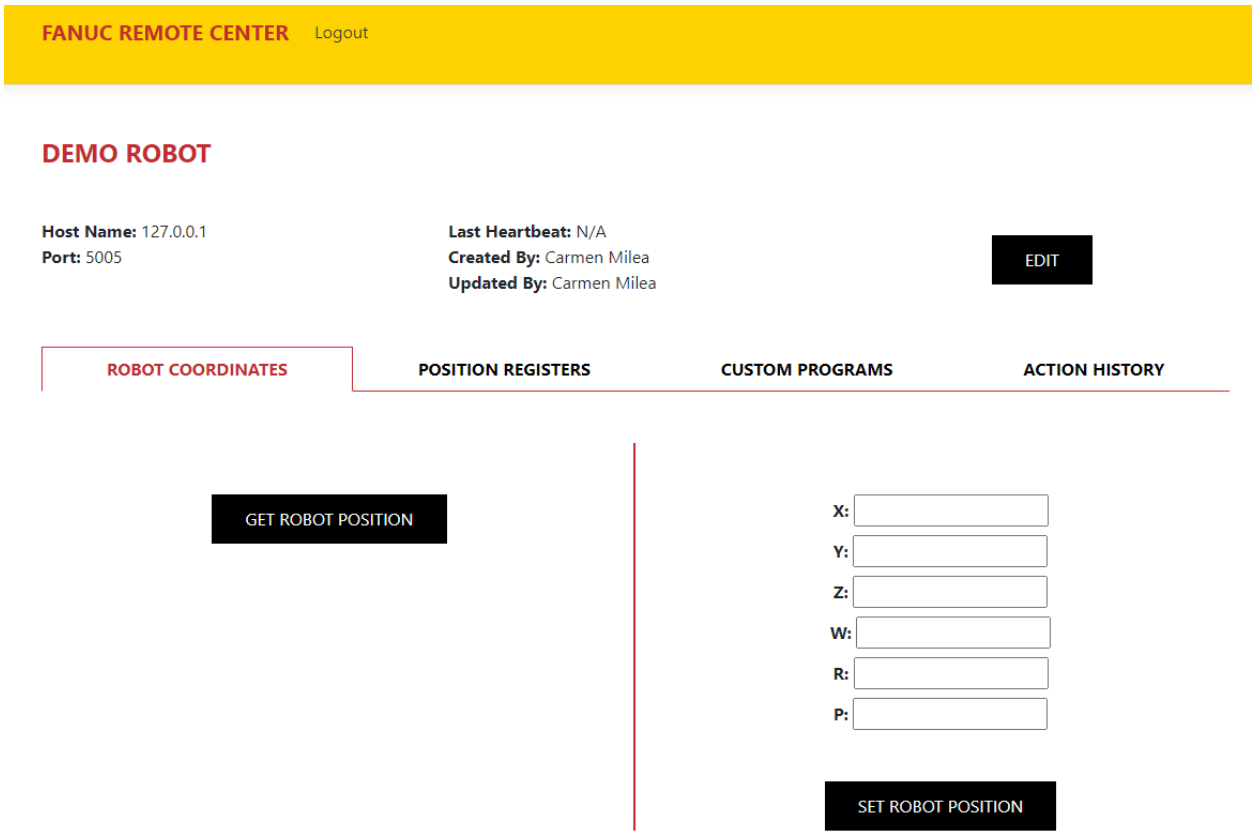

Figure 5 Robot Details Page

As can be noticed, this page is conceptually sectioned into two main components.

The top section contains identification details about the current robot (common name, host and port). Moreover, there is a *Last Heartbeat* field that displays the last time a response was received from the robot (or the *N/A* string if no response was ever received). Audit information about the user name that created and last updated the current robot is also displayed in this section. With the *Edit* button, administrator users can update identification information about robots. One example of this feature use case is when/if the host name or port is updated on a FANUC ROBOT.

The bottom component of the *Details* page is a navigation tab that the user can use for toggling between the available command types for the robot.

By default, the selected navigation item in the tab is *Robot Coordinates*. As hinted by its naming, this section contains commands related to the current robot's position. As mentioned before, FANUC stores positions through six coordinates (the axes and their corresponding rotations).

Therefore, using the right-hand form, the user can input and submit the coordinates the robot should move to. Once they hit *Set Robot Position* and the inputs are validated, the command is formed and submitted to the robot that should perform the movements. The user is informed whether or not the action was successful.

Using the left-hand *Get Robot Position* button, the user can trigger a command that results in the current robot position being displayed on the screen. Using this functionality, we are able to use the setting method. Again, the position resulting from the operation is displayed using the six numerical values.

The second tab, *Position Registers*, is somewhat similar to the previous section, in terms of UI. Functionality-wise, this tab allows the retrieval and change of position registers.There are two newly added inputs for both the get and set methods. Through this input, the user specifies the index of the register their command targets.

Similar to the previous tab, the user can specify the six position coordinates, together with the register index in the right-hand form. They are, likewise, informed of their action status.They can query the existing values of the position registers (and also test the set method), using the left-hand form.

The third tab includes a list of the specific actions the robot can perform. Unlike the commands related to the robot coordinates or position registers that are common among all robots, each FANUC device can have some custom programs.

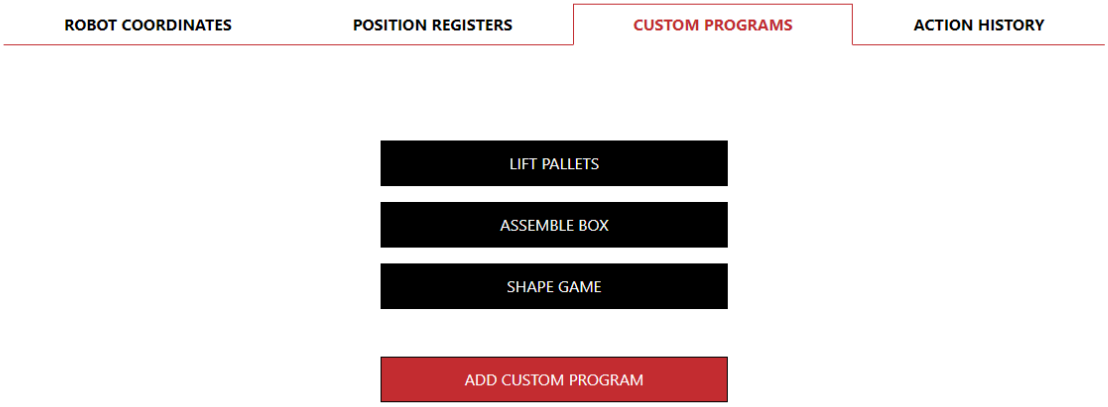

Figure 6 Custom Program Tab

Once a user hits the button associated with one of these programs, that action will be triggered on the robot and the result (success/fail) will be displayed.This page allows the user to add new custom programs as well. When adding one of these, the common name that will be displayed in the UI must be specified; likewise, the user should input the command associated with that name. When adding a custom program, the user needs to make sure that the command matches what the Karel program on the robot expects.

The fourth and last tab of the *Details* page consists of a grid containing historic information about the actions triggered on that robot. For each one of their commands, we display the core information: the action type (e.g. set position, get register or the name of a custom action), the raw request and response body. The last two columns express the information exactly as it is passed through the TCP socket communication. Therefore, it comes in handy in the debug process. This table also contains a Status column that expresses through indicative icons whether or not the action triggered was successful. The last column displayed has audit purposes as it identifies the user that triggered the respective action. The data in this table is sorted by datetime descending, thus making sure that the most recent actions taken are standing out. Figure 7 shows the *Action History* tab.

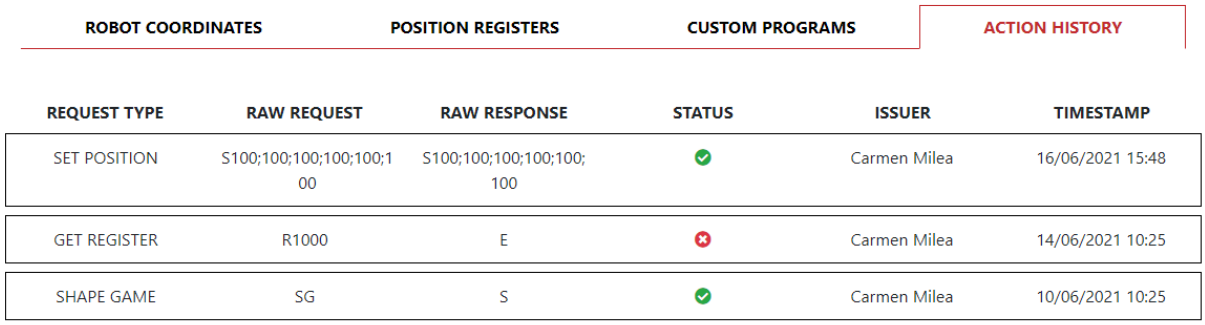

Figure 7 Action History Tab

Chapter 4

Conclusions

Throughout this paper, we emphasized the importance and applicability of industrial robots nowadays, while also analysing the speciality literature and articles on this topic.

We've noticed that the increased importance of industrial robots does not come along with an increased number of practical tutorials or extensive research.

Therefore, this paper aimed to contribute to this niche, by offering a practical approach and detailed implementation details.

The problem that is the root of this paper and that aims to be resolved here, is the need for physical presence in the robot vicinity in order to control the robot (through the Teach Pendant device).

The solution presented in this paper states that, once the robot is in an up-and-running state and connected to a network, having the TCP socket listening on a chosen port is the first step that needs to be done in order to achieve the end goal.

A desktop in the same network is the second component of the application and its role is to submit custom and correctly formatted commands to the TCP server socket on the robot (using a TCP client socket).

One of our main objectives was to come up with a solution that, apart from solving the target problem, manages to be secured in all shapes and forms. However, the result ended up meeting the initial expectations only partially. What we did manage to accomplish is secure the server and web application. However, due to some unexpected limitations regarding the Karel language, we were not able to secure the TCP socket communication pipeline.

The above challenge determined us to consider a change in the physical environment, so that the solution could still be used securely. That means that the robot itself should be air-gapped from the internet, having just one gateway: the web server. Therefore, the security of the entire system relies on the security of the web-server. Considering that we did manage to obtain a satisfactory

result regarding that aspect, we consider that this a good compromise that still meets the business and security requirements.

Some further researches on this area can be focused on one of the following core subjects: making use of the robot's iRVision integrated vision (more details about this can be found at [30]), robot coordination, system management remote tool (which would be an extension to the current paper, by adding the possibility of configuring the robot remotely). Another possible research area, that we consider both challenging and highly important, is the security aspect of the Karel language. We believe that this aspect is not covered enough (maybe not at all) and that a result in this aspect would be greatly beneficial.

Therefore, the paper has reached its initial goal: offering a solution of remotely controlling the FANUC robot. Even though we faced some unexpected limitations that affected the security of the communication channel, we did manage to find a compromise that assures both the security and functionality of our solution.

References

[1] J. Wallén, "The History of the Industrial Robot," 2010.

[2] K. Capek, Rossum's Universal Robots, 1921.

[3] FANUC, "FANUC News," [Online]. Available: https://web.archive.org/web/20160324164024/http://fanuc.co.jp/ja/profile/news/pdf/fanucnews2015iii.pdf. [Accessed 25 04 2021].

[4] FANUC, "FANUC ZDT APPLICATION ZERO DOWN TIME," FANUC, [Online]. Available: https://www.fanucamerica.com/products/robots/zdt-zero-down-time. [Accessed 14 06 2021].

[5] ONE ROBOTICS, "DATATOOL," ONE ROBOTICS, [Online]. Available: https://www.onerobotics.com/datatool/. [Accessed 14 06 2021].

[6] I. Asimov, Runaround, 1942.

[7] K. M. M. S. Miroslaw Pajor, "Kinetic Sensor Implementation in FANUC Robot Manipulation," 2014.

[8] R. e. o. c. a. i. r. u. t. h. voice, "J. Norberto Pires," 2005.

[9] M. P. M. P. F. M. A. M. Z. S. Z. Davide Quarta, "An Experimental Security Analysis of an Industrial Robot Controller," 2017.

[10] F. Lambert, "Tesla receives massive shipment of robots for Model 3 production line," [Online]. Available: https://electrek.co/2017/04/25/tesla-model-3-robot-production-line-pictures/. [Accessed 25 04 2021].

[11] Tesla, "Symphony of robots building Model X," 21 02 2016. [Online]. Available: https://www.youtube.com/watch?v=1o0aDdyIsm4. [Accessed 25 04 2021].

[12] FANUC, "Robots Help Fight the Battle Against COVID-19 Pandemic," 21 03 2021. [Online]. Available: https://ifr.org/case-studies/robots-help-fight-the-battle-against-covid-19-pandemic. [Accessed 25 04 2021].

[13] Interactive Design , "COVID Testing Swab & Tube Feed System - FANUC Scara Robot," 22 06 2021. [Online]. Available: https://www.youtube.com/watch?v=iU6b7TIr7gs. [Accessed 25 04 2021].

[14] L. Cruz, "Digitization and IoT reduce production downtime," 16 05 2016. [Online]. Available: https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1764957. [Accessed 25 04 2021].

[15] FANUC, "Robotics training for schools and universities," FANUC, [Online]. Available: https://www.fanuc.eu/hu/en/robots/educational-package. [Accessed 05 16 2021].

[16] FANUC, "FANUC Training ACADEMY," [Online]. Available: https://www.fanuc.eu/fi/en/lifetime-management/academy. [Accessed 16 05 2021].

[17] FANUC ROBOTICS, "R-30iB Plus Controller," [Online]. Available: https://www.fanuc.eu/pl/en/robots/accessories/robot-controller-and-connectivity. [Accessed 20 05 2021].

[18] Automation Academy, "Fanuc robot programming tutorial Part 1- Teach pendant," 29 June 2020. [Online]. Available: https://www.youtube.com/watch?v=UHSZeU5eyKY. [Accessed 29 March 2021].

[19] FANUC ROBOTICS, [Online]. Available: https://www.fanucamerica.com/products/robots/vision-products. [Accessed 15 14 2021].

[20] Microsoft, "Get started with ASP.NET Core MVC," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-5.0&tabs=visual-studio. [Accessed 26 04 2021].

[21] Microsoft, "Introduction to Identity on ASP.NET Core," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio. [Accessed 05 05 2021].

[22] Microsoft, "TcpClient Class," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcpclient?view=net-5.0. [Accessed 23 04 2021].

[23] FANUC Robotics, FANUC Robotics SYSTEM R-30iA and R-30iB Controller KAREL Reference Manual, 2021.

[24] One Robotics, "TP PROGRAMMING," [Online]. Available: https://www.onerobotics.com/tags/tp-programming/. [Accessed 16 05 2021].

[25] Automation Academy, "FANUC Robot programming Tutorial part 2 - Pick and Place," 22 September 2020. [Online]. Available: https://www.youtube.com/watch?v=Em11oXd3J5s. [Accessed 29 March 2021].

[26] Future Robotics, "FANUC programming tutorial - Create your first program. How to create a TP (teach pendant) program ?," 9 October 2020. [Online]. Available: https://www.youtube.com/watch?v=7miAJodFLgA. [Accessed 15 March 2021].

[27] FANUC, "FANUC ROBOGUIDE SIMULATION SOFTWARE," [Online]. Available: https://www.fanucamerica.com/products/robots/robot-simulation-software-FANUC-ROBOGUIDE. [Accessed 16 05 2021].

[28] Motion Controls Robotics, "How to do a Quick FANUC Robot Backup," [Online]. Available: FANUC Robot Backup. [Accessed 05 16 2021].

[29] ROBOT-FORUM, "FANUC Karel TCP Socket security," [Online]. Available: https://www.robot-forum.com/robotforum/thread/38361-fanuc-karel-tcp-socket-security/. [Accessed 15 06 2021].

[30] FANUC, "FANUC ROBOT VISION PRODUCTS," [Online]. Available: https://www.fanucamerica.com/products/robots/vision-products#:~:text=iRVision%20is%20FANUC's%20unique%2C%20fully,faster%2C%20

smarter%20and%20more%20reliable.&text=Our%20entire%20range%20of%20robots,hardware%20is%20ready%20for%20vision.. [Accessed 17 05 2021].