



Edgar Kurm

Python Usage for MIR Robot Automation

Graduate Work

Institute of technology

Group: RO2019

Supervisor: Villu Lõhmus

Tallinn 2023



Edgar Kurm

Pythoni kasutamine MIR robotite automatiseerimiseks

Lõputöö

Tehnikainstituut

Õpperühm: RO2019

Juhendaja: Villu Lõhmus

Tallinn 2023

AUTORI DEKLARATSIOON JA LIHTLITSENT

Mina, Edgar Kurm, tõendan, et lõputöö on minu kirjutatud. Töö koostamisel kasutatud teiste autorite, sh juhendaja teostele on viidatud õiguspäraselt.

Kõik isiklikud ja varalised autoriõigused käesoleva lõputöö osas kuuluvad autorile ainuisikuliselt ning need on kaitstud autoriõiguse seadusega.

Juhendaja (nimi, allkiri) Villu Lõhmus, allkirjastatud digitaalselt

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Edgar Kurm

(autori nimi)

sünnikuupäev: 28.02.1999

annan Tallinna Tehnikakõrgkoolile (edaspidi kõrgkool) tasuta loa (lihtlitsentsi) enda loodud teose

Pythoni kasutamine MIR robotite automatiseerimiseks

(lõputöö pealkiri)

1. elektroonseks avaldamiseks kõrgkooli repositooriumi kaudu;
2. kui lõputöö avaldamisele on instituudi direktori korraldusega kehtestatud tähtajaline piirang, lõputöö avaldada pärast piirangu lõppemist.

Olen teadlik, et nimetatud õigused jäävad alles ka autorile ja kinnitan, et:

1. lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid ega muid õigusi;
2. PDF-failina esitatud töö vastab täielikult kirjalikult esitatud tööle.

Tallinnas, allkirjastatud digitaalselt.

TABLE OF CONTENTS

AUTORI DEKLARATSIOON JA LIHTLITSENT	3
TABLE OF CONTENTS	4
ABBREVIATIONS	6
INTRODUCTION	7
1. ACTUALITY OF THE PROBLEM	8
2. ROBOT	9
3. MIR	10
4. REST API.....	12
5. PROGRAMMING LANGUAGE	14
5.1. General about Python	14
5.2. Difference between high level language and low level	15
5.3. Python features	16
5.4. Python weaknesses	17
5.5. PIP in Python	18
6. PYTHON SCRIPT	19
6.1. Imports.....	19
6.2. REST API Basic setup.....	20
6.2.1. REST API Authorization.....	20
6.3. Inputs	21
6.4. Mission Cycle Control.....	22
6.5. Filtering.....	23
6.6. Main Part	24
6.7. Scheduling	24
6.8. Python libraries.....	25
6.8.1. Requests.....	25
6.8.2. Delay	25
6.8.3. Json.....	25
6.8.4. mis_mes function	25
6.8.5. main_api function.....	26
6.8.6. timer function	28
7. AUTOMATING SCRIPT	29

7.1. Python Schedule library	29
7.2. Crontab	29
7.3. Windows Task Scheduler	31
7.4. Best way to schedule tasks	33
8. COMPETITORS	35
9. FUTURE	36
SUMMARY	38
KOKKUVÕTE	39
SOURCES	41
LIST OF ATTACHMENTS	42
ATTACHMENTS	43

ABBREVIATIONS

1. MIR - Mobile Industrial Robot
2. REST - Representational State Transfer
3. API - Application Programming Interface
4. JSON - JavaScript Object Notation
5. AMR – Autonomous Mobile Robot
6. ROS – Robot Operating System
7. GUI – Graphical User Interface

INTRODUCTION

Company where I did my diploma practice is called DemekCNC OÜ. Demek CNC OÜ supplies metal machine tools, accessories and equipment. [2] They offer automation solutions, repair and maintenance services for metal machine tools. [2] They offer a complete solution that includes the selection of equipment necessary for production, its installation and maintenance, technology development, training, programming and automation of production.[2] Also they are distributors of Universal Robots and Mobile Industrial Robots. By the time I came to them, they were working on a problem with MIR robot charging mission automation and they gave this problem to me to solve it. The problem is that the customer want to put MIR robot charging at the exact time, because they all go for lunch at 12:00 am, which means there is nobody in the workspace at this time. MIR robot program is running in a loop, it means that 1 mission is repeating over and over again. MIR by itself is a collaborative mobile industrial robot, it should work in cooperation with humans and because of that it has easy interface for regular users, however it is not possible to solve our problem with MIR interface without buying additional software. This functionality is possible to get by the MIR manufacturer's software license, called MIR Fleet, but for Estonian business, the cost of the license and the available functionality are too expensive to pay off economically. DemekCNC asked to look for alternative solution. As I know that MIR platform supports REST API, I decided to use it to remotely take control over MIR robot to add or delete missions. To use REST API there is a script needed written on a programming language. There are multiple programming languages that support REST API, for example: Python, C# , Ruby, Java, JavaScript, PHP. I decided to use Python over other languages. There are multiple reasons, first of all Python have one of the biggest communities, that means that it is easier to search for information, as well as it supports a lot of different libraries. Secondly, Python syntax is shorter than in other languages, which lets us make our script more compact, it leads our script to be easier for other people to understand. And the most important reason for choosing Python, is portability. Python programmes can be easily tranfered from one system to another. For example, i can write code using Windows system, then save it on usb drive, go to another system, for example Linux and it will run properly without any changes in code.

1. ACTUALITY OF THE PROBLEM

The actuality of the problem is very high. DemekCNC have multiple clients that want to solve same problem of scheduling their robots missions without buying MIR Fleet.

Overall, in the Baltics there is 2 clients that want such solution. The biggest problem for the clients and for the Demek as MIR robots distributors is the lack of MIR robot integrators in the Baltic States. General talking integrators are companies that are specialist in MIR robots and that are able to make different custom solutions depending on a customer needs.

The lack of integrators makes my solution very reliable in Baltics. Also MIR robots REST API and Python combination could be used not only for adding scheduling functionality but also for other functionalities and robot automation in general, which could make my solution even more reliable and make new clients for the company.

2. ROBOT

Robot by itself is very wide term and usually people imagine totally wrong things using term robot. Mostly people are imagining robots in the way how they have seen them in the films, which is human like looking machines.

So what is robot?

A device can be called a robot if it can perceive and understand the world around it, as well as influence it.

There are 2 types of robots in general:

1) Industrial

Industrial robots purpose is to help people automate production. Industrial robots have 2 main subtypes which are: Robot Manipulators and Autonomous Mobile Robots or shortly AMRs. Robot Manipulators are made in the similar way to human hand and is used in the most cases on the industrial production line. Manipulators are doing such things as: welding, screwing, palletizing and etc... The use case of AMRs is transportation of a product from one place to another, usually AMRs are working in the collaboration with Robot Manipulators

2) Service

Service robots are made to help people with difficult and routine tasks. Some examples of service robots are: Robot vacuum cleaner and robot lawn mower

3. MIR

MIR is Mobile Industrial Robot, those type of robots are also called autonomous mobile robots or AMRs. MIR have 4 different mobile robot models :

- 1) MIR100 [3]
- 2) MIR250 [3]
- 3) MIR600 [3]
- 4) MIR1350 [3]

The main difference between 4 robot models are maximum payload and movement speed.

The number after MIR name tells us the maximum payload of a robot.

MIR100 have the maximum payload of 100 kg [3]

MIR250 have the maximum payload of 250kg [3]

MIR600 have the maximum payload of 600kg [3]

And MIR1350 have the maximum payload of 1350kg [3]

For my work I have used MIR100, but everything I have done can be used on all MIR robots, because they all share same software.



Picture 1. MIR100 [3]



Picture 2. MIR250 [3]



Picture 3. MIR600 [3]



Picture 4. MIR1350 [3]

The most popular model of a MIR in Estonia is a MIR250. Because it have one of the best money to profit ratio. In depth comparison shows us that MIR250 have same dimensions with MIR100 (890mm x 580), but higher payload(250kg for MIR250 vs 100kg for MIR100), higher move speed (2.0 m/s for MIR250 and 1.5 m/s for MIR100) as well as higher run time (13 hours for MIR250 and 10 hours for MIR100) [3]

4. REST API

REST API , it is usually called Restfull API is Representational State Transfer Application Programming Interface. API is used for the communication between devices through the Internet connection. And REST is a set of rules that applies to API. Those rules were invented by Roy Fielding So general talking , REST is the way how API is created and how it is interacting with the clients using http protocol. So it is architectural style for client and server interaction. In REST client is separated from the server, as well as server is not recording the state of the client, so client is sending to the server only information required to make request and nothing more. The biggest advantage of REST is a single interface for all APIs using this architectural style. That means that all operation over an object should be done using url corresponding to this object. For example if I have object called Missions, all operation with missions should be done using missions url. There is 4 main methods used for the object operation, which are:

1) Delete

This method is used if we need to remove object

2) Get

This method is used for asking some information about object

3) Post

This method is used for adding object

4) Put

This method is used if we need to update information about object

There is also 5th method exists in REST, which is called PATCH.

PATCH method is similar to PUT

The main difference between PATCH AND PUT is that PUT method update entire resource and PATCH gives opportunity to update it partially, but in MIR robots REST API, PATCH is not used.

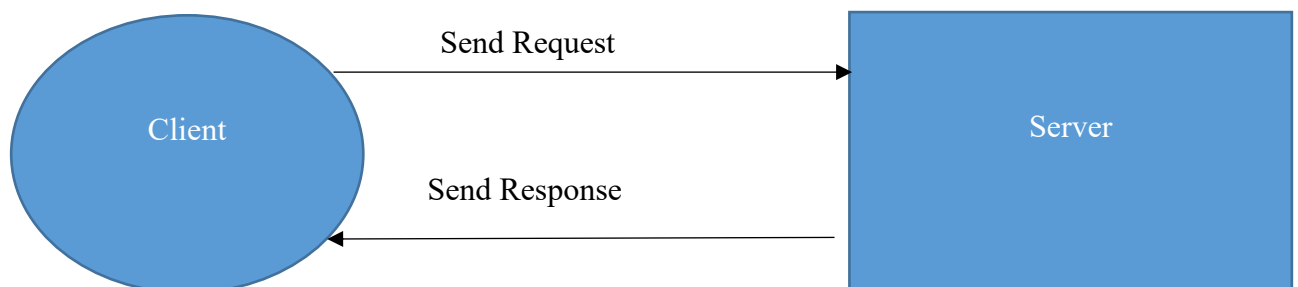
There is another advantage of using REST rules for APIs, which is standardized responses from the server. As we know that all API with REST rules have same interface, that means that we can

understand very well what kind of reply we are getting from server. Is everything alright or we have some kind of problem with response. We are getting numerical output in console, for example 204 for delete request, which means that The element has been deleted successfully [1] or 400, which means: Invalid filters or Invalid JSON or Argument error or Missing content type application/json on the header or Bad request or No fields [1] . This makes troubleshooting much easier and faster.

Responses in REST API could be in such formats as:

- 1) JSON
- 2) XML
- 3) HTML
- 4) Image

But the most popular format for the REST API responses nowadays is JSON.



Picture 5. REST API Scheme[4]

5. PROGRAMMING LANGUAGE

5.1. General about Python

Python is a high-level general-purpose programming language which is nowadays very popular, especially in robotics engineering. Python have different versions, the latest version is Python3. Python is considered as one of the best languages for the beginners. Python is used in a lot of different fields but in robotics the main use cases are:

- 1) Machine Learning
- 2) Machine Vision
- 3) Data Collection
- 4) Hardware Control
- 5) Working with ROS

Another fields where Python is very popular are:

- 1) Data Science

Python is very powerful in collecting information, analyzing information and building different types of models, similar to Matlab, but the biggest advantage of Python over Matlab, is that Matlab requires a paid license, while Python is totally free to use.

- 2) Web Programming

In web service, Python with the newest updates could be used in both Frontend and Backend programming. Python is one of the most popular solutions for programming a backend parts of a web service, because of such Python Frameworks as Django and Flask.

5.2. Difference between high level language and low level

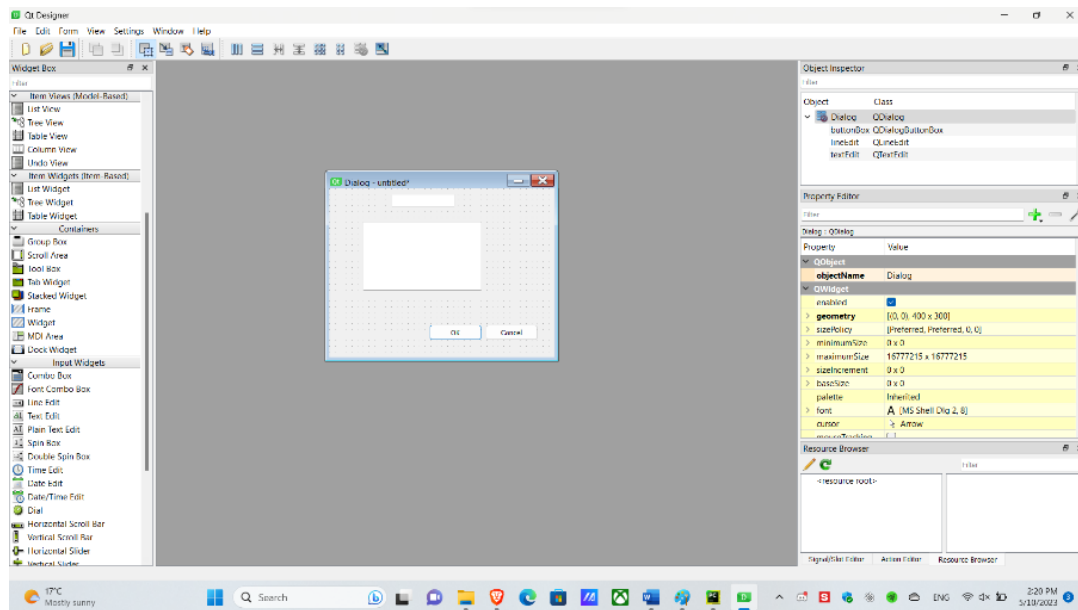
	High Level Language	Low Level Language
1.	It is programmer friendly language. [4]	It is a machine friendly language. [4]
2.	High level language is less memory efficient [4]	Low level language is high memory efficient. [4]
3.	It is easy to understand for human. [4]	It is tough to understand for human. [4]
4.	Debugging is easy. [4]	Debugging is complex comparatively. [4]
5.	It is simple to maintain. [4]	It is complex to maintain comparatively. [4]
6.	It is portable. [4]	It is non-portable. [4]
7.	It can run on any platform. [4]	It is machine-dependent. [4]
8.	It needs compiler or interpreter for translation [4]	It needs assembler for translation. [4]

Table 1. High level language vs low level [4]

5.3. Python features

There is multiple features of the Python language that makes it so popular:

- 1) First of all Python is easy both to read and write code. Code written on the python language is shorter, comparing to other languages, which makes it much more compact and the syntax of the Python is straightforward and looks more similar to the human language.
- 2) Another feature of Python is its libraries. Python have a lot of standard libraries that come already preinstalled with Python and there is even more libraries that you can install by yourself
- 3) Being an high level language is one of the features of Python as well, as we know from the Table 1 from the the topic difference between high level language and low level
- 4) One of the biggest strengths of Python is its portability. Python code can be executed on all popular computer systems without require of changing anything in the code. That means that you can write code using Windows system and then save it on usb drive for example and successfully run it on Linux system without making any changes.
- 5) Dynamical memory allocation is very useful thing as well and also one of the reasons why Python language is so good for beginners. But this is not useful only for beginners, but for a professional programmers as well, because memory allocation is relatively hard thing and takes programmers time to do it.
- 6) Another very usefull Python feature is GUI support. As Python have very large amount of libraries, it is possible to program graphical user interface using Python. There is 2 main libraries for GUI in Python, both should be installed, as they are not a part of the standard Python libraries. Those libraries are: Tkinter and PyQt (latest version PyQt5). Both libraries could be easily installed by using PIP and the syntax is following: `pip install tkinter` and `pip install pyqt5`. PyQt5 is the most popular library for GUI programming and there is reason why it is so. With PyQt5 it is possible to install thing called PyQt Designer, which is Graphical GUI Designer, where you have different boxes, buttons etc... and you can move it as you want and place as you want using graphical interface. When you finish creating your GUI with PyQt Designer you can translate it to the Python language and use it inside your code. To install PyQt Designer there is such command as `pip install pyqt5-tools`



Picture 6. PyQt designer (authors picture)

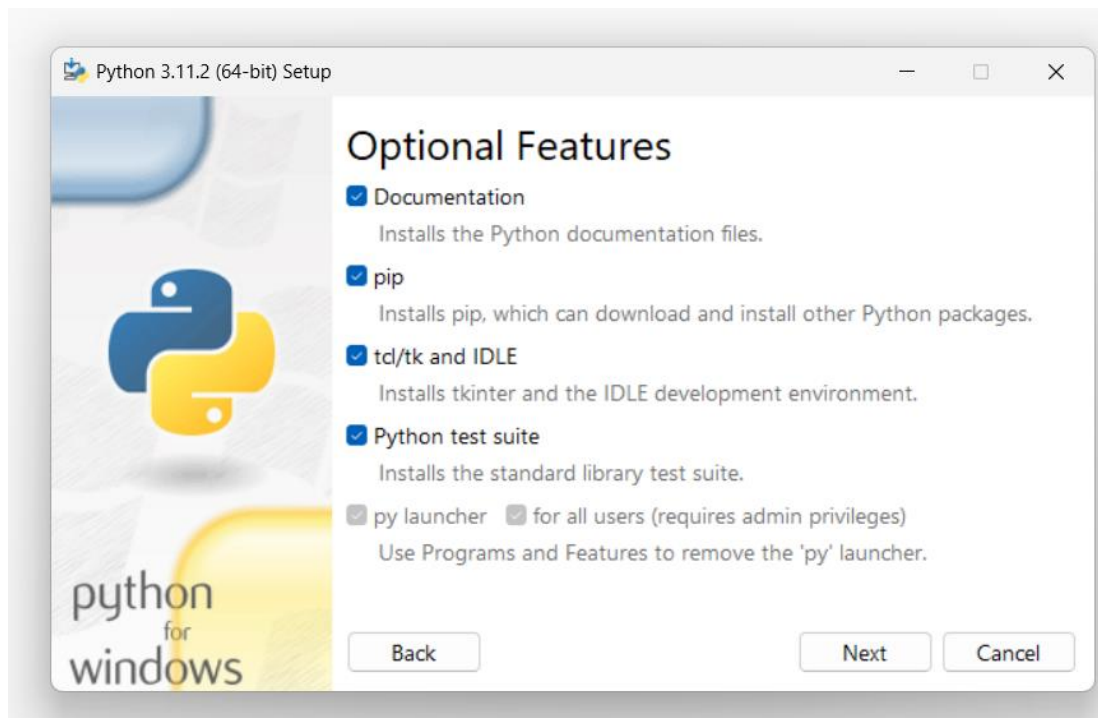
- 7) Another feature of Python is that it is object-oriented language. Object-oriented programming is the set of rules of how code needs to be written. The general principle is that programming is presented in the form of interaction between objects. There is 4 main elements in object-oriented programming, which are: classes, objects, methods and attributes.
- 8) Python is dynamically typed programming language and that means that it is not needed to specify type of the variable during the coding process. Python language is specifying variable types automatically when we press run button.

5.4. Python weaknesses

Everything in this world have its own strength and weaknesses, Python is not an exception. There is 2 main weakness of the Python language, both comes from its strengths, having dynamical memory allocation as well as being dynamically typed language, all those leads to being slow language in comparison to other programming languages as well as being not memory efficient.

5.5. PIP in Python

PIP stands for Preferred Installer Program and is used to install packages in Python. The overall syntax of pip is `pip install + name of the package`. For example `pip install Schedule`. Newest Python versions comes with the build in opportunity to install pip on your device, you just need to make sure you are making custom installation and put check mark in the right box.



Picture 7. Pip installation (authors picture)

6. PYTHON SCRIPT

Scripts consists out of 7 parts, which are:

- 1) Imports
- 2) Basic Setup
- 3) Inputs
- 4) Mission Cycle Control
- 5) Filtering
- 6) Main Part
- 7) Scheduling

6.1. Imports

This is the part where i am importing Python libraries , to use them further in my code.

```
# Imports
import requests
from time import sleep
import schedule
import json
import time as tm
```

Picture 8. Library Imports (authors picture)

6.2. REST API Basic setup

URI scheme:[1]

Host : mir.com/api [1]

BasePath : /v2.0.0 [1]

Schemes : HTTP [1]

Consumes • application/json [1]

Produces • application/json[1]

```
# Setup
ip = 'mir.com'
host = 'http://' + ip + '/api/v2.0.0/'

# Headers
headers = {}
headers['Content-Type'] = 'application/json'
```

Picture 9. Basic setup (authors picture)

6.2.1. REST API Authorization

To authenticate in the API a basic authentication has to be included in the header. [1] It consists in a string that is base64 encoded and it is formed by the username followed by a colon and the password sha-256 encoded. [1] Ex: `BASE64(:SHA-256())` [1]

```
headers['Authorization'] = 'Basic
YWRtaW46OGM2OTc2ZTViNTQxMDQxNWJkZTkxOGJkNGRlZTE1ZGZiMTY3YTljODczZ
mM0YmI4YTgxZjZmMmFiNDQ4YTlxOA=='
```

Picture 10. Authorization (authors picture)

This is the unique key that gives permission to communicate with robot. To get authorization key it is needed to go to the MIR soft, which is available by typing mir.com in web browser, if we are using mir own network, or by typing robots ip adress in to the browser search bar if we are connected to another router.

6.3. Inputs

This is the part that consists of 4 variables that are used to get inputs from the user and store them for the future use in code. First variable is first_position is asking user to enter first position in the mission cycle. Second one is create_mission_1 which is used to get users charging mission name. Third one is create_mission_2 which is used to get users regular mission name. The last variable is scheduler and it asks user to enter time, when user want to execute code.

```
#Inputs
first_position = input('your first position name:')
create_mission_1 = input('enter your charging mission name here :')
create_mission_2 = input('enter your regular mission name here :')
scheduler = input('put your time here:')
```

Picture 11. Inputs (authors picture)

6.4. Mission Cycle Control

This part is used to control if robot have done mission cycle to the end or not. As i know that clients regular robots mission is looped, which means that 1 mission is repeating endlessly. And mission that we add to the robots queue is going to the bottom of the queue. So the only way to replace looped mission is to remove all missions. If there is product on the robot, that robot is transporting to the destination area and we remove mission in the middle of the cycle and simply put charging mission instead, there is very high risk that the product will not be transported to the destination area and robot will go to the charging station with product on the top of it, calling regular mission back after charging with the product from the enterupted cycle being on the top of the robot, gives us very high risk to damage the product, because robot will start its movement from the first action in the mission. To prevent this i have created function `mis_mes`. This function is checking when the robot is starting moving from the last position to the first one, which will designed that mission cycle have been done.

```
#Mission Cycle Control
def mis_mes():
    quated = "Moving to " + "'" + first_position + "' " + "(0.7 meters to goal)"

    while True:
        mission_message = requests.get(host + "/status", headers=headers)
        print(mission_message.text)

        text_list = json.loads(mission_message.content)
        print(text_list)

        if text_list['mission_text'] == quated:
            break
```

Picture 12. Mission Cycle Control (autors picture)

6.5. Filtering

In this part i have created filter that gives oppurtunity to call robots missions using their names instead of guids. It is the part of the main_api function. This function consists of 2 parts: 1) Filtering and Main Part. Filtering is required to make code usage easier to the user. Because when someone is working robots GUI , mission names are used instead of mission ids.

```
#Filtering
def main_api():
# Getting Mission Data
get_missions = requests.get(host + 'missions', headers=headers)
print(get_missions.text) # // get missions guids

list_missions = json.loads(get_missions.content)
print (list_missions)
dict_missions = {}

for mission in list_missions:
    dict_missions[mission['name']] = mission
    print(dict_missions)

list_name_missions =dict_missions .keys()

guid_1 = dict_missions[create_mission_1]['guid']
print(guid_1)
guid_2 = dict_missions[create_mission_2]['guid']
print(guid_2)
```

Picture 13. Filtering Part (autors picture)

6.6. Main Part

Main Part is the second part of the main_api function and it does 4 consecutive actions

- 1) Delete all mission in queue
- 2) Call charging mission
- 3) Delete charging mission after time
- 4) Call regular mission back to the robot

```
5) #Main Part
# Delete All Missions In Queue
delete_actions = requests.delete(host + 'mission_queue', headers=headers)
print(delete_actions)

# Calling Charging Mission
mission_id_1 = {"mission_id":guid_1}
charging_mission = requests.post(host + 'mission_queue',
json=mission_id_1, headers=headers)
print(charging_mission)
sleep(35) # // if delay is needed // number in () is seconds

# Delete Charging Mission After Time
delete_charging = requests.delete(host + 'mission_queue',
json=mission_id_1, headers=headers)
print(delete_charging)

# Calling Regular Mission Back #Should be user input
mission_id_2 = {"mission_id":guid_2}
regular_mission = requests.post(host + 'mission_queue',
json=mission_id_2, headers=headers)
print(regular_mission)
```

Picture 14. Main Part (authors picture)

6.7. Scheduling

This part is made to execute code at the entered by user time and is using timer function to do it.

```
#Scheduling
def timer():
    mis_mes()
    main_api()

schedule.every().day.at(scheduler).do(timer)
while True:
    schedule.run_pending()
    tm.sleep(1)
```

Picture 15. Scheduling part (authors picture)

6.8. Python libraries

Library in Python is a premade template that you can fill with your data and get the result. It is made to make coding faster and more convenient, because you do not need to write same code part in every new file, instead you can just import library.

6.8.1. Requests

Request is a library is designed to make http requests to a specific url, which is exactly how REST API works, so it makes request library a perfect choice for using it for the operation on the objects using REST API. Request is not a standard library and need to be installed. The best way to do it is using pip. Pip install request is the syntax for installing this library.

6.8.2. Delay

In 1 part of my code I need my code to wait some time before continuing. This is the part after adding charging mission to the robots queue, because we want to keep charging mission for 1 hour, as we know that at 12:00 starts lunch time in the clients workhouse and lunch is 1 hour long. For this I am importing sleep module from time library and using sleep after calling charging mission. The syntax is sleep(number in seconds), so we just need to add our number inside the brackets, how long we want code to sleep before in continue. Time library is standard Python library and doesn't need to be installed before importing it in the code.

6.8.3. Json

JSON is JavaScript Object Notation. To make operations over json objects there is json library in Python, which is standard library and can be imported without installations, cause it comes preinstalled with the Python.

6.8.4. mis_mes function

First variable inside mis_mes function is quoted, it is translating user input that we got from the Inputs part of the code, which is saved in first_position variable, to the form that robot will be able to read. After quoted variable i have While True loop, this loop will endlessly repeat code inside this loop until it will reach true statement. First variable inside While True loop is mission_message, that is sending get request to the robot with the host + status url. After that we need to save our response

from robot in another variable, which is text_list in my case and then we need to translate response from robot to python language, because by default python cant read what is inside response. This part in programming is called parsing. To parse my response I am using json.loads from Json python library. Then i am writting if statement to my While Loop, this statement is if text_list['mission_text'] == quated: break , which mean that when the mission_text will be equal to Moving to + our entered possition name + (0.7 meters to the goal), loop will break and pass code to the next part.

```
#Mission Cycle Control
def mis_mes():
    quated = "Moving to " + "'" + first_position + "' " + "(0.7 meters to goal)"

    while True:
        mission_message = requests.get(host + "/status", headers=headers)
        print(mission_message.text)

        text_list = json.loads(mission_message.content)
        print(text_list)

        if text_list['mission_text'] == quated:
            break
```

Picture 16. mis_mes function (autors picture)

6.8.5. main_api function

Inside this function first variable is get_missions, it is using get request with host + missions url, which means it is <http://mir.com/api/v2.0.0/missions> , get_missions is used to get list of our missions with their urls, guids and their names. Now i need to parse my response from the robot, same to the previous function mis_mes, i am using json.loads fom Json library. For parsing i have created variable list_missions, it is parsing response using json.load method and saving it . Then I have created empty dictionary dict_missions to save inside this dictionary all robots mission names. Afterwards I have created variable list_name_missions and made it to be equal to dict_missions.keys(), which means that we made dict_missions to be our key for further filtering. As I want to make my code to be easier for user, I am using two following variables in this part, which are create_mission_1 and create_mission_2, those variables are user inputs from Inputs part of the code. After that I have 2 similar variables, guid_1 and guid_2, those variables appeal to dictionary dict_missions which I have made as my key, and are getting specific mission ids, called guids, depending on a entered mission names by user, from create_mission_1 and create_mission_2 variables. Mir robot requires mission id, which is randomly generated code by robot, for any operation with robots missions. We cant simply use mission names in our post mission request, that's why I have variables guid_1 and guid_2, they are storing those ids and we can use them afterwards for request. Then I am starting to work

already on deleting and adding missions to the robot. And this parts starts with deleting all missions that are in queue on the robot at this time. We need to do this, because in my conditions, client have looped mission in queue all the time and the post request on mir works the way that when it adds missions in queue, they go to the bottom of the queue, so this will be the last mission that robot will do and this is not bad, but as we know that clients mission is looped, that means that if I post mission to the bottom of queue, robot will never reach it. For this delete_actions variable is used, which gives delete request to the robot, with the host+ mission_queue url. Thereafter I am starting adding charging mission in queue, to do that I have variable mission_id_1 which is equals to {"mission_id":guid_1}, writing "mission_id" is required for the robot to recognize that I am giving him id of the mission, otherwise it will not understand that and will give response 400 which means "Argument error or Missing content type application/json on the header or Bad request or Invalid JSON" . Then I made variable charging_mission that send post request to the robot, with host+ mission_queue url and I type json = mission_id_1, which is used to give to the robot id of the mission. When charging mission is posted to the robot, mir should charge at station for 1 hour, for this I use sleep block from time library. When 1 hour has passed, robot should replace charging mission with regular one. For deleting charging mission I have delete_charging variable created, which gives delete request to the robot with host+mission_queue url, same way with deleting all missions in queue, but this time I am using json = mission_id_1 again to tell robot that we need to delete specific mission, in this case it is charging mission. Calling regular mission is same to calling charging mission and it is post request with host+mission_queue url, but this time I have variable mission_id_2, that again tells robot we are going to give him mission id with typing "Mission_id": and then giving him regular mission guid that is stored in variable called guid_2 and posting the mission I am entering json = mission_id_2, to call regular mission instead of charging mission.

```
#Filtering
def main_api():
# Getting Mission Data
get_missions = requests.get(host + 'missions', headers=headers)
print(get_missions.text) # // get missions guids

list_missions = json.loads(get_missions.content)
print (list_missions)
dict_missions = {}

for mission in list_missions:
    dict_missions[mission['name']] = mission
    print(dict_missions)

list_name_missions =dict_missions .keys()

guid_1 = dict_missions[create_mission_1]['guid']
print(guid_1)
```

```

guid_2 = dict_missions[create_mission_2]['guid']
print(guid_2)

#Main Part
# Delete All Missions In Queue
delete_actions = requests.delete(host + 'mission_queue', headers=headers)
print(delete_actions)

# Calling Charging Mission
mission_id_1 = {"mission_id":guid_1}
charging_mission = requests.post(host + 'mission_queue', json=mission_id_1,
headers=headers)
print(charging_mission)
sleep(35) # // if delay is needed // number in () is seconds

# Delete Charging Mission After Time
delete_charging = requests.delete(host + 'mission_queue', json=mission_id_1,
headers=headers)
print(delete_charging)

# Calling Regular Mission Back #Should be user input
mission_id_2 = {"mission_id":guid_2}
regular_mission = requests.post(host + 'mission_queue', json=mission_id_2,
headers=headers)
print(regular_mission)

```

Picture 17. main_api function (authors picture)

6.8.6. timer function

The last function in my script is timer, it is used to read entered time by user, which I am getting with scheduler variable from the Inputs part and then at this time call functions mis_mes and main_api. In this function I am using Schedule python library, this library is very easy to use, we just need to enter how often we want to schedule script, in my case every day, then we need to enter at what time we want it to do and what command should it do. After entering all needed information, for running, this library requires While True loop. Which will control while all entered conditions are true or not.

```

#Scheduling
def timer():
    mis_mes()
    main_api()

schedule.every().day.at(scheduler).do(timer)
while True:
    schedule.run_pending()
    tm.sleep(1)
    tm.sleep(1)

```

Picture 18. timer function (authors picture)

7. AUTOMATING SCRIPT

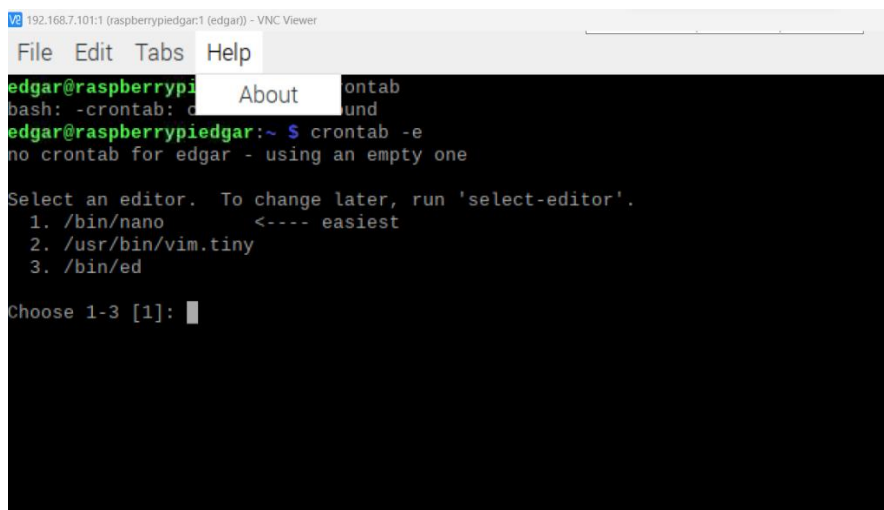
I need my script to run automatically every day at exact time. And there is 3 most popular ways to do it: Windows Task Scheduler on Windows Machines, Crontab on Linux and MacOS machines and Schedule library in Python, which can be used on all 3 systems.

7.1. Python Schedule library

One of the possible options to automate python script is Schedule library in python. Schedule is not a standard Python library, so it is needed to be installed manually. The best way to install Schedule library is by typing: `pip install schedule` in console.

7.2. Crontab

On the Linux and MacOS systems there is build in schedule programm called Crontab, usually called simply cron. Crontab is one of the most popular ways for scheduling programming codes. To open crontab you need to open your terminal and enter `crontab -e`. If you are starting Crontab for the first time then it will ask you to select editor. The most popular editor for Crontab is Nano.



```
192.168.7.101:1 (raspberrypiedgar:1 (edgar)) - VNC Viewer
File Edit Tabs Help
edgar@raspberrypiedgar:~$ crontab -e
bash: -crontab: d
und
edgar@raspberrypiedgar:~$ crontab -e
no crontab for edgar - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano          <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed

Choose 1-3 [1]:
```

Picture 19. Starting Crontab for the first time (authors picture)

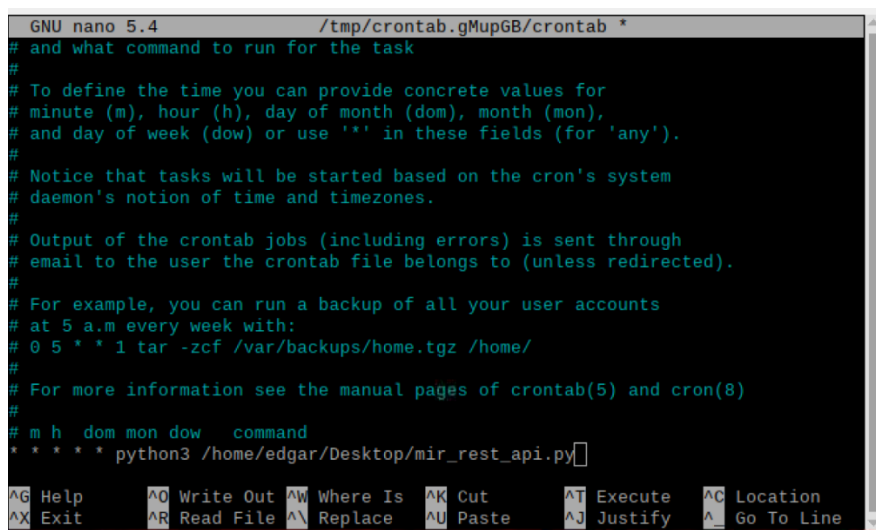
Crontab have 6 parameters that need to be filled:

- 1) Minute parameter
- 2) Hour parameter
- 3) Day of month parameter
- 4) Month parameter
- 5) Day of week parameter
- 6) Command parameter

So the example of the Crontab syntax is 05 05 05 05 05 /home/mir_rest_api.py . This means that python code that is saved as rest_api file, which is located at home will execute at 05:05 am at the 5th of May on Friday.

There is also such thing as * in crontab, which means every, so if we want to say that we want to execute command every minute, we will use * instead of number in minute parameter.

Example of using * in crontab is * * * * * + your command. This means do command every minute. Only in command parameter there could not be * , because we need something to schedule.



```
GNU nano 5.4 /tmp/crontab.gMup6B/crontab *
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
* * * * * python3 /home/edgar/Desktop/mir_rest_api.py
```

Picture 20. Crontab example (authors picture)

Another example could be if we want to execute code at every day at 10:30 am from Monday to Friday. The syntax for this is `30 10 * * 1-5 /home/mir_rest_api.py`

For the client needs, as we know that client want robot to go charging at 12:00 am every day, syntax will be `00 12 * * * + path to the script file`

7.3. Windows Task Scheduler

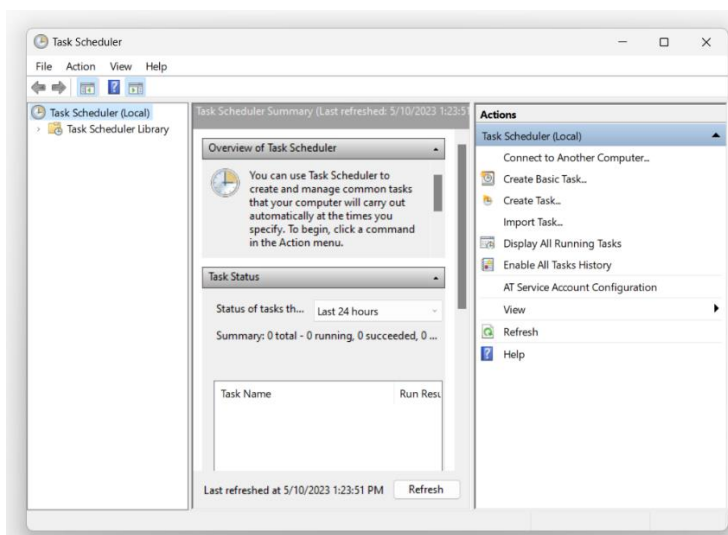
Windows have own program for scheduling tasks, which is called Windows Task Scheduler. It is build in program, so doesn't need to install anything separately.

How to schedule Python code using it?

First of all we need to have Python script written and saved as a file, because we need something to execute.

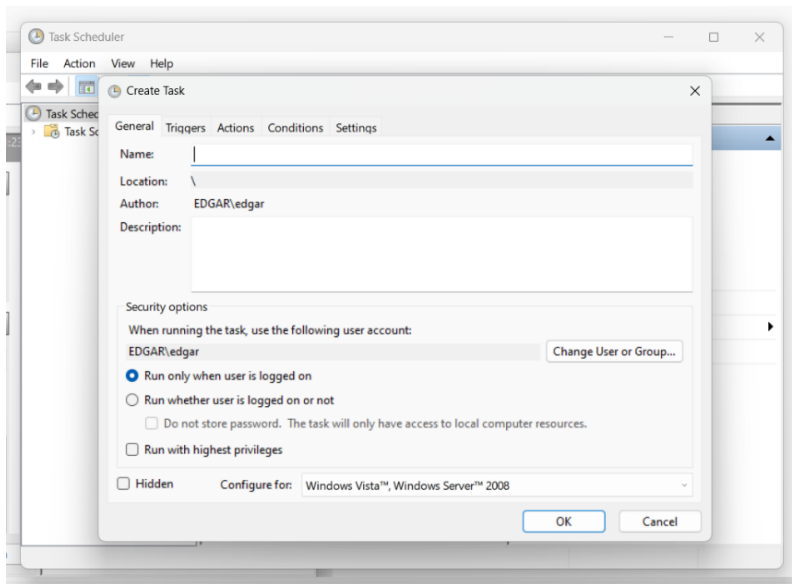
The next thing you need to do is to enter task scheduler in windows search bar, find the program and open it.

After opening task scheduler, need to click create task



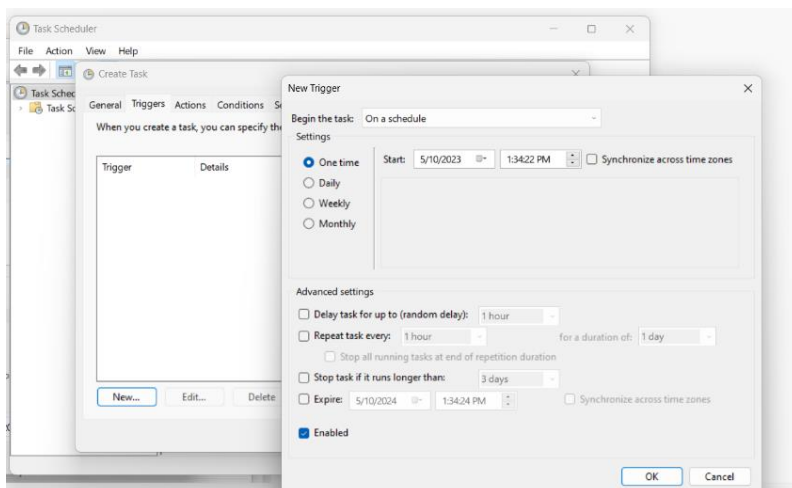
Picture 21. Task Scheduler main menu (authors picture)

Then new window will open, where you need to enter your name and optionally you can add description of what exact scheduler is doing. As well as opportunity to choose if you want this task scheduler to execute only if user is logged on or you want this scheduler to be able to run even if user is not logged on. And also you can put check mark to let this task to run with the highest privileges.



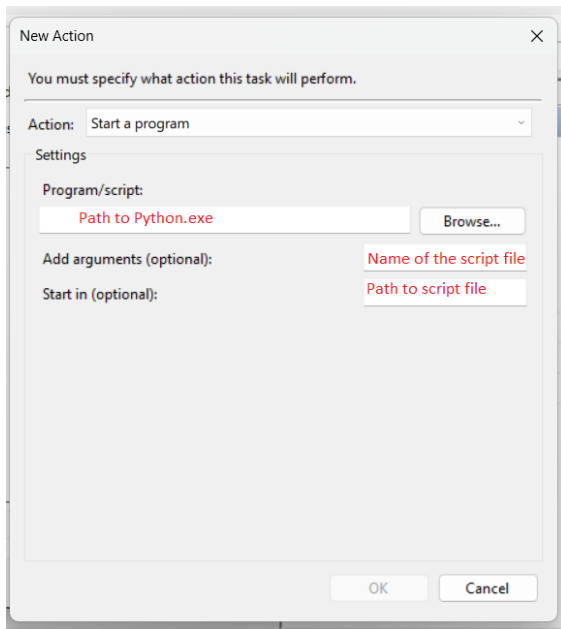
Picture 22. Create task General bar (authors picture)

Then we need to move from General bar to Trigger tab and click new. There we have different very in-depth options of when we want to schedule our task.



Picture 23. Trigger options (authors picture)

Now we go to another bar called Actions and click new. Here in Program/Script part we need to specify the path to our python.exe file. In add argument part we need to specify our python script name as well as specify that this is python file, example mir_rest_api.py. And in Start in part we need to specify the path where our script is located.



Picture 24. Task Scheduler Action Bar (authors picture)

And finally click ok, task scheduler will ask you to enter your PC's password and scheduler is created successfully

7.4. Best way to schedule tasks

So what is the best way for scheduling Python script?

Overall all 3 methods mentioned before are viable, however I want to mention that in my experience Windows Task Scheduler is the worst one, because scheduling task using it takes much more time in comparison to 2 other methods and it is a little bit complicated to use, as well as I had some problems working with Python scripts using it. However this method has wide options for customization.

The crontab is a very solid option for scheduling scripts, it is very easy and fast to use. The biggest disadvantage is that crontab doesn't exist on Windows systems.

Schedule library in Python is overall the best option in my opinion, it has easy for understand syntax and code writing is fast. As this scheduling method comes as a code, from the user side it is required to just enter time and nothing more, everything else is done by Python code. Also this method has good portability, because Python code can be executed on Linux, Windows and on MacOS systems without any changes.

8. COMPETITORS

The main competitor to my solution is MIR Fleet from MIR manufacturer.

MIR Fleet is available in 2 variants:

8) MIR Fleet PC [3]

9) MIR Server Solution [3]

MIR Fleet PC: comes as a physical device, little box computer [3]

MIR Server Solution: is made for intergration into existing server and comes as a software only. [3]

The price of MIR Fleet is: 13000 euro, so i am competing with expensive solution.

The main advantage of MIR Fleet is that it adds Fleet tab into existing MIR software, which makes this solution easy to use for user. The biggest dissasvantage on the other hand is that MIR Fleet doesnt let you to add 1 specific functionatily separatly, that means that you should buy whole sulutions and pay money for all its functionalities, even if you dont need all of them. Advantages of my solution is that it could be sold to the client for a much cheaper price, it could be 3 times less price or even more. It is focusing on a specific functionallity that client want to get and it could be modernised in future if needed. As my solution is software it doesnt require spending money on materials.

9. FUTURE

My project have very good potential for future growth. As REST API gives access to all robot features, it is possible to combine REST API and the power of Python for adding more functionality to it, because right now it is used only for automating charging process of the MIR robot, but REST API and Python combination can be used to automate even more processes.

Script cant run itself, it requires something to execute it. Python code can be executed without any changes on 3 main operation systems:

10) Windows

11) Linux

12) MacOS

So in the future it is possible to add support of 2 more systems, which are:

1) Andriod

2) iOS

Both of them are used as smartphone and tablet operating systems.

Another step in my project is going to be integration of my solution into clients environment. It is needed to analyze the working environment of the MIR robot and decide how exactly we are going to execute script.

Right now,my solution is made as a code part only, so it is made for integrations into existing server solutions, samilar to MIR Fleet Server Solution. However i have ideas of how to make standalone solution. Depending on what i have for now, the best solution in my opinion could be using single-board computer, for example, raspberry pi, which is the most popular single-board computer solution right now.

It is required for executable device to be in the same network with MIR. So there is 3 ways of using raspberry pi with MIR robot:

- 1) The first one is to install raspberry pi inside robot and connect them both to internal MIR robots WIFI

- 2) The second one is to use external router and connect both raspberry pi and MIR robot to the external router
- 3) The last one is probably for smaller rooms. Connecting raspberry pi directly to MIR robots wifi network and use builded in raspberry pi wifi adapter.

Another opportunity for the future growth of solution and also opportunity to make this solution standalone, could be making executable file with graphical user interface. For the GUI could be used 2 of mentioned in my work Python libraries, which are:

- 1) Tkinter
- 2) PyQt5

And to make an executable file, there exists such tool as py installer, which will convert our .py file into exe file, that could be easily used on all PC systems.

SUMMARY

The problem i was solving is that DemekCNC client wanted to put MIR robot at charging station at exact time automatically at the time when they all are going for lunch and charge during this time. Robots default software don't have this function. MIR manufacturer offers their solution, MIR Fleet, which adds mission scheduling feature to the MIR as well as a lot of other features. The problem of MIR Fleet for the client is its price, which is high and that they don't need other extra features for their robot. After this time, they wanted to put their regular mission back. My suggested solution for this problem was to use MIR robots own REST API and combining it with Python, create code that will be able to add missions to the robot and take them out of the robot. I have added 2 more features to my code, except adding and deleting missions. The first one is control of the mission cycle. Which is used to check if robot have done mission cycle to the end or not. This function is needed because clients regular mission is looped, which means that robot repeats same movements over and over again. Because of that, it is not possible to simply add charging mission to the robots queue, because every time mission comes to the mir robots queue it is going to the bottom of the queue and as we know that clients regular mission is looped, it will never end and charging mission will never execute. So the only way to add charging mission to the mir robots queue is to firstly remove all missions that are already in queue and only then add charging mission. However, if I remove regular mission in the middle of mission cycle there could be such situation, when robot comes to the charging station with the product on top of it. This situation has risk to damage the product. Because after charging during launch time robot will come back to regular mission and start his movements from the first action. So in this situation, if robot haven't delivered product to the destination area before going at charging station and will start mission from the beginning, there is high risk to get new product on the top of the product that remained on the robot, which could lead to product damage. To prevent this, I have function `mis_mes` that will firstly check if mission cycle have been finished and only after that, pass code to the removing and adding mission part. And the second one is mission filtering using mission names. By default, robot can make operations with mission only using specific mission id, which looks this way: 96adef52-cc93-11ed-bf3d-94c691a73491. It is not very easy to find this id, so I decided to add opportunity to make operations with mission using mission names, which are easy to find with MIR robots software. Summarizing everything I have done, can say that I have developed working solution for the problem that I received form the company and it has opportunity for future growth. For example right this code is used to solve problem with charging mission scheduling, but it can be used for adding other functionalities as well. Video with the working code as well as full code and code scheme you can find in attachments.

KOKKUVÕTE

Probleem, mida lahendasin on, et DemekCNC kliendil on vajadus panna MIR roboti laadima kindlal ajal. Kliendil, kõik töötajad lähevad lõunale kell 12:00, ehk tööruumis ei ole mitte kedagi sellel ajal ning töö protsess peatatakse. Klient tahab et MIR robot läheks automaatselt lõuna ajal laadimisjaama ja laeks ennast pausi ajal, kokku 1 tund. Pärast pausi, tahab klient automaatselt taaskäivitada roboti tavamissiooni. Missiooni ajatamise funktsioon puudub tavalisel roboti tarkvaral ja tootja pakub selle funktsionaalsuse lahenduseks MIR Fleet tarkvara, mis on kallis lahendus, praegune hind on 13000 eurot. MIR Fleet lisab võimaluse ajatada roboti missioone ja mitte ainult seda, vaid lisab ka teisi funktsionaalsusi lisaks, kuid klientidel pole vaja täiendavaid funktsioone ja ei ole nõus maksma soovitud litsentsi, et kasutada ainult ajatamise funktsioon MIR Fleetiga. MIR on Mobile Industrial Robot, mis tähendab et see on Mobiilne Tööstus Robot. On olemas 4 erinevaid mudelit: MIR 100, 250, 600, 1350. Peamised erinevused on liikumise kiirus ja kandevõime. Number tähendab kaalu, mis robot suudab tõsta, näiteks MIR100 saab tõsta kuni 100 kilogrammi, MIR250 kuni 250 kilogrammi ja nii edasi. Kuid tarkvara koodi jaoks pole see oluline, sest neil on kõikidel sama tarkvara. MIR Fleetil on olemas 2 versiooni: esimene on MIR Fleet PC, mis tuleb väikese eraldiseisva arvutina ja teine on MIR Fleet Server Solution, mis on arendanud paigaldamiseks olemasolevasse serverisüsteemi.

Minu pakutav lahendus kasutab MIR enda REST API võimalusi, toetudes Python scriptile, mis võimaldab sama funktsionaalsust ja teha seda kõike kasutajate jaoks võimalikult lihtsalt. REST API on Representational State Transfer Application Programming Interface, ehk see on rakendustarkvara liides, mis kasutab REST reeglid ja annab võimaluse suhelda robotiga. Põhiidee on, et meil on olemas REST API klient, meie juhul see on Python script, mis saadab päringu serverile, meie juhul server on MIR robot, mis saadab kliendile vastuse. Python on kõrgetasemeline, üldotstarbeline programmeerimise keel, mis on täna väga populaarne. Pythoni eelised on, et ta on programmeerija sõbralik, Pythoni kood on lühike ja lugemiseks mugavam võrreldes teiste keeltega. Pythonil on olemas palju standardseid raamatukogusid. Raamatukogu on eelvalmistatud koodi lõikude šabloon, mis on võimalik kasutada oma koodis, et kiiremini ja lihtsamini programmeerida. Python on ka portatiivne keel, see tähendab et ilma muudatustega võib panna käima koodi kõikidel platvormidel, vahet pole missugune süsteem oli kasutatud koodi kirjutamiseks. Minu koodis on 3 funktsionaalsust:

Esimene on missioonide kutsumine ja eemaldamine kasutades MIR roboti REST API võimalusi, see on koodi peamine funktsionaalsus. Teine on missiooni lõpetamise kontroll, see on väga tähtis funktsionaalsus, sest kliendi robotil on tehtud ainult 1 missioon, mis jookseb loopis, mis tähendab et

robot kordab lõputult sama missiooni ja kui me lihtsalt lisame roboti missioonile midagi juurde, siis ta läheb järjekorra alla ja kuna roboti tavaline missioon on loopis, siis robot mitte kunagi ei jõua alustada seda missiooni. Ainuke võimalus on eemaldada kõik missioonid järjekorrast ja ainult pärast seda lisada laadimise missioon, siis laadimis missioon on esimene ja ainuke missioon järjekorras ja robot hakkab seda missiooni läbi viima. Seepärast on oluline, et robot teeks missiooni tsükli lõpuni. Kui me võtame roboti ära missiooni tsükli keskel, meil võiks olla selline olukord, kus robot läheb laadima ja roboti peal on toode, sest ta ei jõudnud toodet viia sihtpunkti, siis robot läheb laadima koos tootega. Pärast lõunapausi lõppemist, kui me paneme roboti tema tavamissioon tagasi, robot ei jätkka oma missiooni vaid alustab seda uuesti, esimesest positsioonist siis võib olla selline olukord kui toode peale pannakse teine toode ja see võib tekitada kahju. Selle vältimiseks on olemas koodis `mis_mes` funktsioon , mis kontrollib kas missiooni tsükkel on lõpetanud või mitte ja kuni selle hetkeni, kuni robot teeks tsükli lõpuni ei lubata koodil edasi minna. Siis kood eemaldab kõik missioonid, mis on robotil järjekorras ja lisab laadimise missiooni, siis robot laadib 1 tund ja pärast seda kood eemaldab laadimise missiooni ja paneb tagasi roboti tavalisse missiooni režiimi.. Viimane funktsionaalsus, mis on loodud minu lõputöö raames, on missioonide filtreerimine, kasutades nende nimesid. Selline funktsionaalsus on kasulik, sest MIR ei anna võimalust kutsuda vaikimisi missioone, kasutades nende nimesid, vaid ta tahab saata unikaalse missiooni id, mis ei ole väga lihtne leida. Missiooni id näeb välja järgmiselt: 96adef52-cc93-11ed-bf3d-94c691a73491. Põhimõtte on, et kood rakendub missiooni nimele õige missiooni id.

Kokkuvõtteks võin öelda et minul on tehtud töötav lahendus esialgsele probleemile ning sellel lahendusel on hea potentsiaal edasiarenduseks. Näiteks praegu see on kasutatud laadimismissiooni ajatamise probleemi lahendamisel, kuid sarnane lahendus võib-olla kasutatud ka teise funktsionaalsuse loomises.

SOURCES

[1] Mir rest api manual [Webmaterial]: (Available only inside MIR robots software) [Used:05.04.2023]

[2] Company DemekCNC website [Webmaterial]: demek.ee [Used:05.04.2023]

[3] MIR robot website [Webmaterial]: mobile-industrial-robots.com [Used:05.04.2023]

[4] GeeksForGeeks website [Webmaterial]: [geeksforgeeks.com](https://www.geeksforgeeks.com) [Used:05.04.2023]

[5] Python website [Webmaterial]: python.org [Used:05.04.2023]

[6] Programmers forum Stackoverflow [Webmaterial]: stackoverflow.com [Used:05.04.2023]

LIST OF ATTACHMENTS

Attachment 1. Full code

Attachment 2. Code scheme

Attachment 3. Code video

ATTACHMENTS

```
# Imports
import requests
from time import sleep
import schedule
import json
import time as tm

# Setup Host
ip = 'mir.com'
host = 'http://' + ip + '/api/v2.0.0/'

# Headers
headers = {}
headers['Content-Type'] = 'application/json'
headers['Authorization'] = 'Basic
dHVkZW5nOmQ2NmQ4MDJiNjQ5NDkxMjY3Njk5NTU1MGUxNGY3MzQ5NGNjYzIxZmJiZWVjOGZiMThhM2Rl
MGU3YjQwMjNjYzY='

#Inputs
first_position = input('your first position name:')
create_mission_1 = input('enter your charging mission name here :')
create_mission_2 = input('enter your regular mission name here :')
scheduler = input('put your time here:')
def mis_mes():
    quated = "Moving to " + "'" + first_position + "' " + "(0.7 meters to goal)"

    while True:
        mission_message = requests.get(host + "/status", headers=headers)
        print(mission_message.text)

        text_list = json.loads(mission_message.content)
        print(text_list)

        if text_list['mission_text'] == quated:
            break

def main_api():
# Getting Mission Data
get_missions = requests.get(host + 'missions', headers=headers)
print(get_missions.text) # // get missions guides

list_missions = json.loads(get_missions.content)
print (list_missions)
dict_missions = {}

for mission in list_missions:
    dict_missions[mission['name']] = mission
    print(dict_missions)

list_name_missions =dict_missions .keys()

guid_1 = dict_missions[create_mission_1]['guid']
print(guid_1)
guid_2 = dict_missions[create_mission_2]['guid']
```

```

print(guid_2)

# Delete All Missions In Queue
delete_actions = requests.delete(host + 'mission_queue', headers=headers)
print(delete_actions)

# Calling Charging Mission #Should be user input
mission_id_1 = {"mission_id":guid_1}
charging_mission = requests.post(host + 'mission_queue', json=mission_id_1,
headers=headers)
print(charging_mission)
sleep(35) # // if delay is needed // number in () is seconds

# Delete Charging Mission After Time #Should be same with calling mission
delete_charging = requests.delete(host + 'mission_queue', json=mission_id_1,
headers=headers)
print(delete_charging)

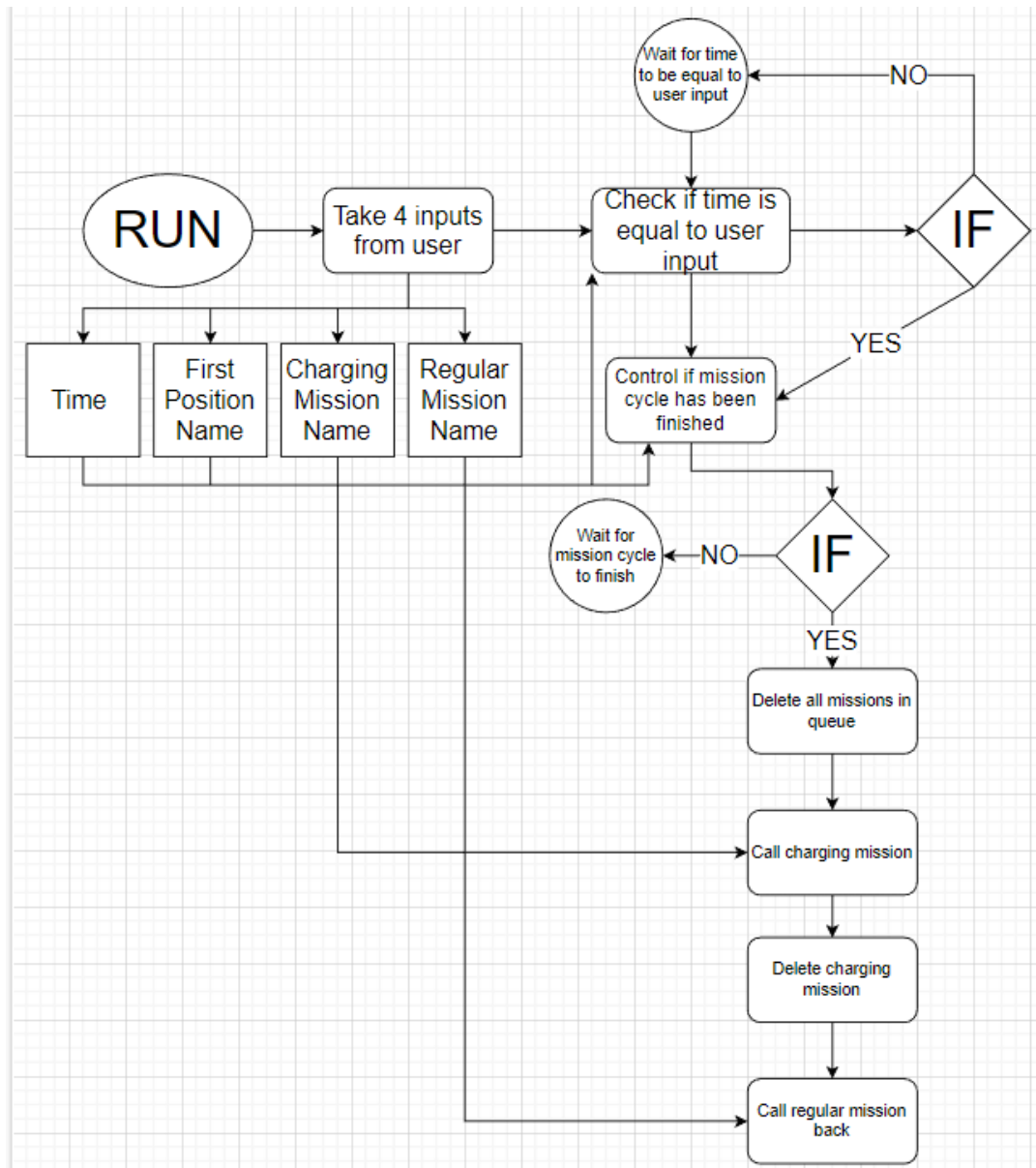
# Calling Regular Mission Back #Should be user input
mission_id_2 = {"mission_id":guid_2}
regular_mission = requests.post(host + 'mission_queue', json=mission_id_2,
headers=headers)
print(regular_mission)

def timer():
    mis_mes()
    main_api()

schedule.every().day.at(scheduler).do(timer)
while True:
    schedule.run_pending()
    tm.sleep(1)

```

Attachment 1. Full code



Attachment 2. Code scheme

Video link for the running code

<https://www.youtube.com/watch?v=gbq7xf4OsFw>

Attachment 3. Code video