# ZÁRÓDOLGOZAT

**Czene Attila**

**2023**

**Magyar Agrár- és Élettudományi Egyetem**

**Károly Róbert Campus**

**Műszaki Intézet**

**Programtervező Informatikus felsőoktatási szakképzés szak**


# IPARI AUTOMATIZÁLÁS PYTHON NYELV HASZNÁLATÁVAL


| | |
|---|---|
| **Belső konzulens:** | Dr. Novák Tamás<br>tanszékvezető |
| **Belső konzulens intézete/tanszéke:** | **Műszaki Intézet / Alkalmazott Informatika Tanszék** |
| **Készítette:** | **Czene Attila** |



**Gyöngyös**

**2023**

# Tartalomjegyzék

## Tartalom


1. Bevezetés ..... 4
2. Vállalat bemutatása ..... 5
3. Az ipari automatizációs folyamat ..... 6
   3.1. A folyamat célja ..... 6
   3.2. A termék ismertetése ..... 6
   3.3. Eszközök csatlakoztatása ..... 7
   3.4. A forráskód bevezetése ..... 9
      3.4.1. Python nyelvről ..... 9
      3.4.2. Installáció ..... 9
      3.4.3. A programozási környezet ..... 10
   3.5. A forráskód bemutatása ..... 10
      3.5.1. Bevezetés ..... 10
      3.5.2. Folyamatábra ..... 11
      3.5.3. Importálás Blockly-ból ..... 12
      3.5.4. A robotkar mozgásának meghatározása ..... 15
4. Összegzés ..... 25
5. Internetes irodalom ..... 26
6. Ábrajegyzék ..... 26

# 1. Bevezetés

Egyetemen a Programtervező Informatikus szakos éveim alatt, volt lehetőségem az informatika számos területébe betekintést nyerni, ezekről tanulni, gondolok itt akár az üzemeltetés-, akár a programozás témakörére, mely esetekben a való életben, munkám során is sok és hasznos tudást szereztem. Bár az üzemeltetés egy nagyon érdekes, színes, hozzám közel álló terület, munkámba mégis próbálok egyre több programozást belevinni és karrieremet ezirány felé terelni.

A programozáson belül a Python nyelvet kedveltem meg leginkább. A nyelv népszerűsége az évek során folyamatosan nőtt és ma széles körben használják a szoftverfejlesztés számos területén, beleértve a webfejlesztést, adatfeldolgozást, gépi tanulást, mesterséges intelligenciát, automatizálást és számos más területet. Személy szerint, én az egyetemen beadandó feladatként játékot és mobilalkalmazást is készítettem a segítségével.

Úgy gondolom, ipari környezetben lassan elengedhetetlen a munkafolyamatok automatizálása, napjainkban ezt már a megrendelők is elvárják. Az automatizálási folyamatok javítják a termelékenységet és a termékek minőségét, miközben csökkentik a munkaerőigényt és az emberi hibákat. Jelenlegi munkahelyemen a termelősorokon, már régóra használnak PLC vezérelt, robotizált megoldásokat pl. a ragasztás, préselés folyamatára, de csak nemrég kezdték el bevezetni olyan robotkar használatát, ami már a gyártósorra egy külön beilleszthető, emberi munkát helyettesítő megoldás, mely esetben a folyamat egy másfajta beállítást, programozást igényel. Erre a programozásra a Python nyelvet használjuk, aminek ismeretének köszönhetően kaptam lehetőséget és tudtam csatlakozni egy cégen belüli automatizálási folyamathoz.

# 2. Vállalat bemutatása

2021.03.16-a óta dolgozom jelenlegi munkahelyemen a Magnetec Ungarn Kft.-nél, melynek fő profilja lágymágneses beépülő alkatrészek gyártása. Az 1989-ben alapított gyártó üzem a Magnetec Csoport gyártó központja. A Magnetec az elmúlt 30 évben nyújtott teljesítményével elérte azt, hogy azon kevés piaci szereplők közé tartozik, akik vezető szerepet töltenek be a speciális, nanokristályos mágnesmagok globális piacán. A Magnetec márkanevek, mint a Nanoperm, CoolBlue, NaLa és a Cool Tube régóta keresettek a legmagasabb minőségi követelményeket felállító induktív alkatrészek piacán. A cég termékeit az anyavállalaton keresztül a világ minden táján értékesíti, legnagyobb részben az Európai Unióban. A jelenleg kb. 300 főt foglalkoztató cég termékei induktív alkatrészként, észrevétlenül épülnek be az élet számos területére: ott vannak az ipari és a háztartási szektorok élet- és tűzvédelemében, az energetikában, a közlekedésben. A termékek alkalmazási területei számára kiemelkedően fontos, hogy a végfelhasználók kompakt kialakítású, kis tömegű és mindenekelőtt magas energiahatékonyságú készülékeket, illetve berendezéseket gyárthassanak.

A termékfejlesztések több területet is átfognak. Az egyik legjelentősebb részt a megújuló energiák hasznosítására szakosodott ipari gyártás igényei képviselik. Ez az alkalmazási terület érinti a megújuló energiatermelést (szél, nap és vízenergia), az energiaátalakítás területeit, a villamosenergiaelosztó hálózatokat és a villamosenergia felhasználás és fogyasztás, pl. e-mobilitás területeit, valamint mindemellett ugyancsak fontos kiemelni a magas hozzáadott értékű innovatív termékeket igénylő elektromos fogyasztásmérők piacát.

A cégnél egy három fős rendszergazdai csapat tagja vagyok, mely mellett egy szintén három fős programozói csapat is dolgozik, akik a vállalatirányítási rendszer (Abas) programozói munkái mellett, saját fejlesztésű szoftvereinken is dolgoznak, mint pl.: mérőprogram a mágnesmagok mérésére, illetve webes felület a legyártott termékek lejelentésére.

# 3. Az ipari automatizációs folyamat

## 3.1. A folyamat célja

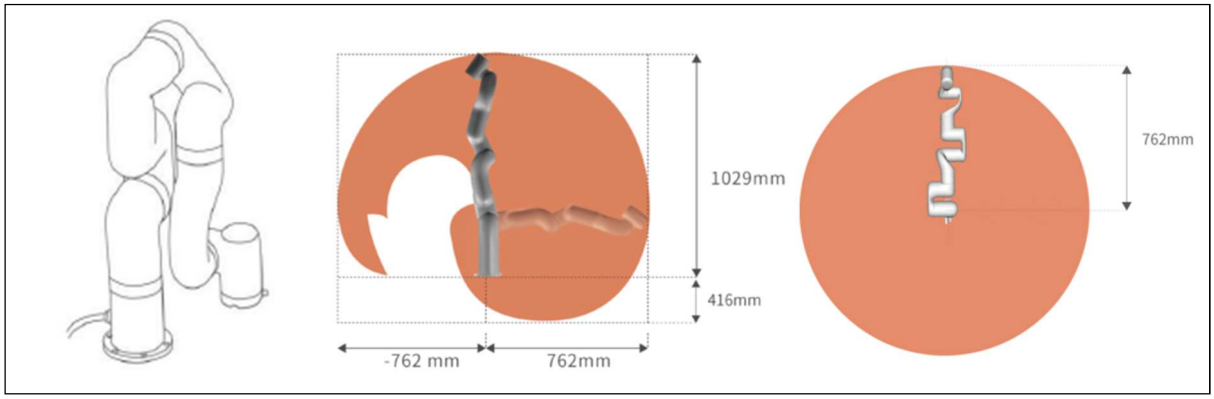
A legyártott mágnesmag termékek szétválogatása, tálcára pakolása mérésük eredményétől függően.

A robotkar állomásához érkező magok mérésének eredményéről jelet küld a mérőállomás az xArm robotkar vezérlőegységének, ami alapján a robotkar a megfelelő helyre rakja a termékeket – rossz mérés esetén a selejt tárolóba, jó mérés esetén, pedig a megfelelő kiosztású tálcára.

## 3.2. A termék ismertetése

Amit az automatizációs folyamathoz használunk és záródolgozatomban bemutatok, a UFactory cég által gyártott xArm 6 típusú robotkar, ami egy fő oszlopból és több forgó csuklóból, szervomotorból áll. Ezeknek az „ízületeknek” köszönhetően, egyfajta szabadságot kap a kar a mozgásban. Meghatározott távolságon belül, szinte bármilyen pozíciót fel tud venni (1. ábra).

*1. ábra: xArm robotkar és hatóköre*




*Forrás: xArm-User-Manual-V2.0.0.*

A robotkar végén a termék felvételére használható megfogó kar vagy akár vákuumos felvételi lehetőség is. Jelen esetben a nálunk gyártott mágnesmag termékek gyűrűs formájából adódóan kézenfekvőbb a karos megfogás (2. ábra) használata. A megfogó nyitásának és zárásának tartománya: -10 és 850 értékek között állítható.

*2. ábra: Megfogó*

*Forrás: xArm-User-Manual-V2.0.0.*

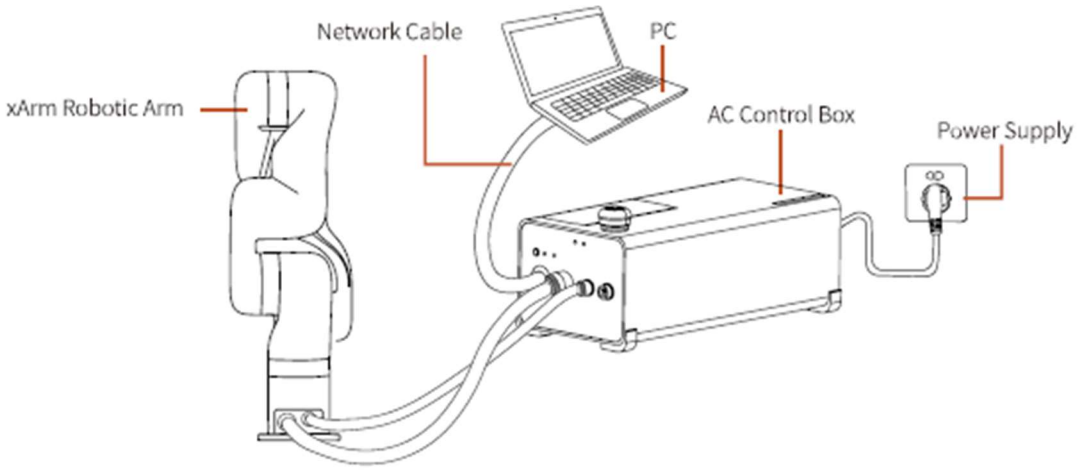
A vezérlést, valamint a kapcsolatot a PC és a robotkar között az AC Contol Box valósítja meg (3. ábra).

## 3.3. Eszközök csatlakoztatása

Az eszközök kábelezése:

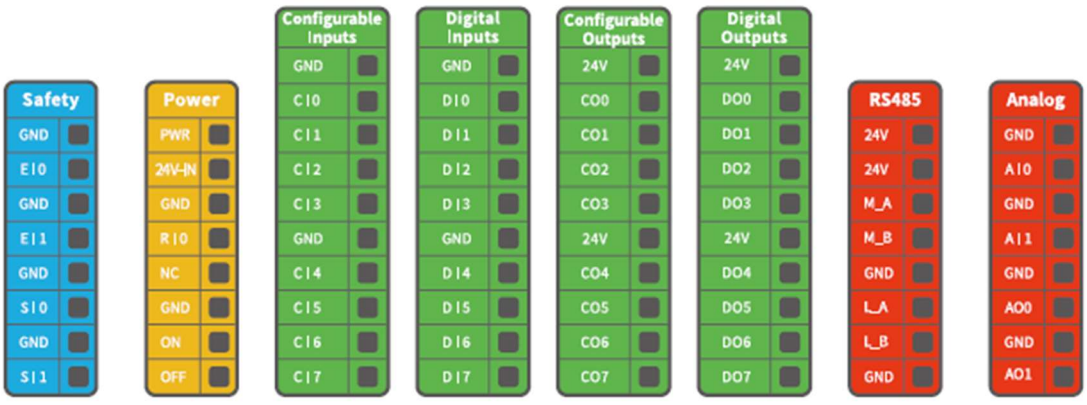
- a vezérlőegység egy tápkábellel csatlakozik az elektromos hálózathoz (230V),
- a vezérlőegység saját 24V-os táp- és jelkábelével csatlakozik a robothoz,
- a PC a vezérlőegység LAN portjára csatlakozik UTP kábelen keresztül (közvetlenül vagy switch-en keresztül),
- a robotkar előtti mérést végző állomás vezérlésének digitális kimenetei, a robotkar vezérlőegységének digitális bemeneteire csatlakoznak többeres jelkábellel (4. ábra).

*3. ábra: AC Control Box*




*Forrás: xArm-User-Manual-V2.0.0.*

*4. ábra: AC Control Box portjai*




*Forrás: xArm-User-Manual-V2.0.0.*

A PC és a robotkar közti kapcsolat létrehozásának további két fontos eleme a megfelelő összekötést követően:

- eszközök azonos IP tartományban legyenek:
    - robotkar: 192.168.250.4
    - PC: 192.168.250.10
- UFactory Studio nevű szoftver segítségével csatlakozzunk a robotkarhoz, az IP címének megadásával (5. ábra).

*5. ábra: UFactory Studio*


*Forrás: UFactory Studio szoftver képernyőkép*

Ezt követően használhatjuk teljeskörűen a gyártó saját szoftverét és érhetjük el pl. azt a felületet is, ahova majd a Python kódot tölthetjük fel, ami alapján fogja majd elvégezni a robotkar az utasításokat.

## 3.4. A forráskód bevezetése

### 3.4.1. Python nyelvről

A Python egy általános célú, magasszintű programozási nyelv, amelynek tervezési filozófiája a kód olvashatóságát hangsúlyozza. A Python szintaxisa lehetővé teszi, hogy kevesebb kódsorban fejezzük ki a fogalmakat, mint amennyi az olyan nyelvekben lehetséges, mint a C. A nyelv olyan konstrukciókat biztosít, amelyek lehetővé teszik a kis és nagy léptékű programok áttekinthetőségét.

A Python többféle programozási paradigmát támogat, beleértve az objektumorientált, imperatív és funkcionális programozási stílusokat. Teljesen dinamikus típusrendszerrel és automatikus memóriakezeléssel rendelkezik, hasonlóan a Scheme, Ruby, Perl és Tclm rendszeréhez, valamint nagy és átfogó szabványkönyvtárral rendelkezik.

Más dinamikus nyelvekhez hasonlóan a Pythont is gyakran használják szkriptnyelvként, de számos nem szkriptes környezetben is használják. Harmadik féltől származó eszközök segítségével a Python kód önálló futtatható programokba csomagolható. A Python interpreterek számos operációs rendszerhez elérhetők. [1]

### 3.4.2. Installáció

Operációs rendszerként a Windows 11 64bites Home verzióját használom, ezalatt telepítem és futtatom a szükséges szoftvereket.

- Elsősorban a nyelv használatához szükség van egy interpreterre, amit a Python hivatalos honlapjáról le lehet tölteni (https://www.python.org/downloads/),
- másodsorban, pedig szükség van egy programozási környezetre, amire én úgy gondolom, hogy az egyik leginkább megfelelő, a PyCharm.

[1] Kelas Karyawan: (Python (programming language)). On-line: http://kelas-karyawan-bali.kurikulum.org/IT/en/2420-2301/Python_3721_kelas-karyawan-bali-kurikulumngetesumum.html

### 3.4.3. A programozási környezet

A PyCharm egy népszerű, integrált fejlesztői környezet (IDE) a Python programozáshoz, amelyet a JetBrains fejleszt. Az IDE számos funkcionalitást és eszközt kínál, hogy hatékonyabban és kényelmesebben dolgozhassunk a projektekkel. Számomra egyik legnagyobb pozitívuma, hogy gyorsan és könnyen elérhetők, listából kiválaszthatók és telepíthetők a Python nyelv „package”-i, melyeket ezt követően, csak importálni kell a kód elején és használatra készek. Elérhető a PyCharm ingyenes verziója, a Community Edition, amely ideális az alapvető fejlesztési feladatokhoz, valamint ezenkívül van egy fizetős változat, a Professional Edition, amely további fejlett funkciókat és támogatást kínál.

## 3.5. A forráskód bemutatása

### 3.5.1. Bevezetés

Az robotkarhoz számos támogatást nyújt a fejlesztőknek a gyártó.

Elsősorban létezik egy ún. Blockly, amely egy olyan grafikus programozási környezet, amelyet a UFactory fejlesztett ki a robotkarok és oktató robotok programozásának egyszerűsítésére. E platform célja, hogy segítse a robotika és programozás tanulását egy könnyen érthető, grafikus megközelítéssel, melyet leginkább az egyetemen is használt Scratch-hez lehetne hasonlítani. Blockly-val a képernyőn megtekinthetők a programok struktúrái, így könnyen észlelhetők és hibakeresésre kerülnek a hibák, viszont az ilyen vizuális nyelvek korlátozottak lehetnek bizonyos komplex feladatokhoz vagy összetett algoritmusokhoz.

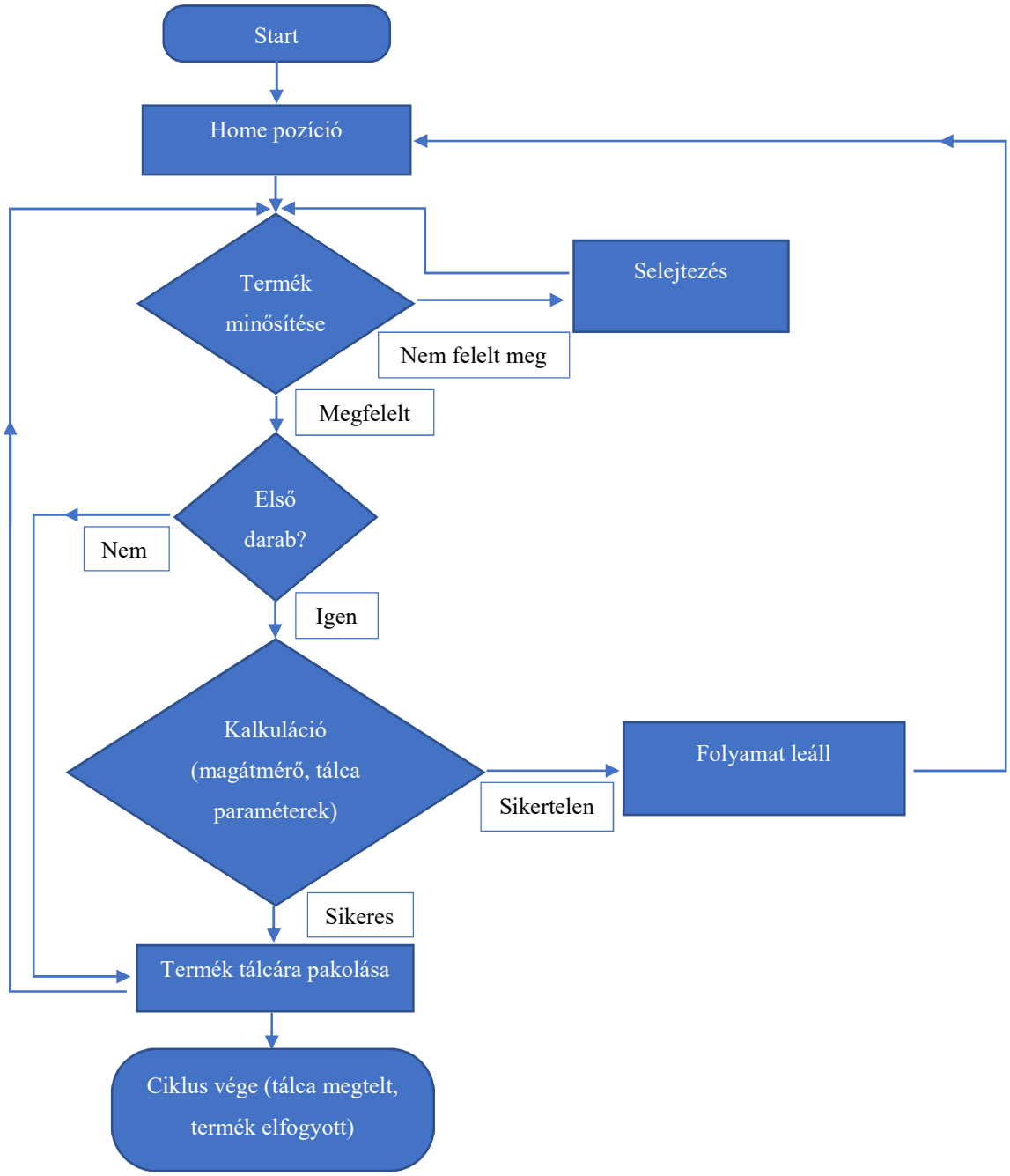
Másodrészt a UFactory GitHub profilján belül megtalálható a cég szoftvercsomagja, eszközkészlete (SDK - Software Development Kit) a robotkarhoz Python és C++ nyelven. Az SDK tartalmaz olyan eszközöket, forrásfájlokat és példakódokat, amelyek lehetővé teszik a kommunikációt a UFactory robotokkal és megkönnyítik a programozói munkát.

Az SDK a következőket tartalmazza:

- API dokumentáció: Az SDK tartalmazza az UFactory API dokumentációját, amely leírja az elérhető funkciókat, metódusokat és parancsokat a robotok irányításához. Ez a dokumentáció leírja a használati utasításokat és példákat a robotokkal való kommunikációhoz.

- Kommunikációs interfészek: Az SDK tartalmazza a kommunikációs interfészeket és könyvtárakat, amelyek segítenek a robotokkal történő kapcsolat kialakításában. Ezek a könyvtárak lehetővé teszik a robotok állapotának lekérdezését és utasítások küldését.
- Példakódok: Az SDK általában tartalmaz példakódokat különböző feladatokhoz, például mozgatás, szenzorok kezelése vagy egyéb műveletek. Ezek a példakódok segíthetnek megérteni, hogyan használható az SDK egy projektben.

### 3.5.2. Folyamatábra



### 3.5.3. Importálás Blockly-ból

Blockly-ból lehetőség nyílik az összerakott utasításoknak az átvezetésére egy Python környezetbe, így látható, hogy a blokkokból összerakott utasítások milyen kódsorokat alkotnak. Ez az átvezetési folyamat nagyon hasznos a kezdéshez, mert ebben az esetben rögtön legenerálódik egy közel 130 soros Python kódsorozat, ami tartalmazza a kezdéshez szükséges beimportált alap modulokat, metódusokat.

A Python nyelvben lehetőségünk van arra, hogy a kódunkba akár a beépített standard library-ből vagy külső library-kből is importáljunk különböző modulokat, melyek komplex definíciókat, funkciókat és utasításokat tartalmaznak kódunkhoz. Legyen szó akár egy egyszerű „random" modulról, hogy véletlenszámokat lehessen generálni, vagy akár egy külső „MDApp" modulról, hogy mobilra tudjunk alkalmazást készíteni.

Valamint mindemellett a modulok segítik a kód áttekinthetőségét, karbantarthatóságát, mert lehetőséget nyújtanak a program egy adott részletét külön fájlban tárolni, majd azt meghívni és így sokkal rendszerezettebb forráskódot elérni.

Az alábbi modulok importálódnak Blockly-ból Python IDE-be:

- import sys (különböző rendszer specifikus funkciókat tartalmaz és lehetővé teszi a Python programnak, hogy interakcióba lépjen a Python futási környezetével)
- import math (matematikai műveleteket és függvényeket kínál)
- import time (időzítési és időkezelési funkciókat kínál, például időzítőket és időzóna-kezelést)
- import queue (olyan implementációt tartalmaz, amely sorban kezeli az elemeket és hasznos lehet párhuzamos programoknál és szálaknál)
- import datetime (dátum és idő műveleteket, objektumokat kínál a dátumok és idők kezeléséhez)
- import random (véletlen számokat generál)
- import traceback (segítségével hibakezelés során hibajelentések és hibakeresési információk készíthetők)
- import threading (szálak (threads) kezelését teszi lehetővé és lehetővé teszi a párhuzamos programok írását)
- from xarm import version (eszköz-, szoftververzió lekérése)

- from xarm.wrapper import XArmAPI (egy specifikus Python API-t tartalmaz a robotkarral való kommunikációhoz)

A modulok importálását követően a kód egészét egyetlen osztály foglalja magában, ez a „class RobotMain(object)”.

Innentől kezdődően vannak definiálva a kód még átvezetett metódusai. Vannak paraméterek, melyeknek az értéke már előre be van állítva egy alap értékre, valamint vannak olyanok, melyek még csak definiálva vannak, de csak később kapnak értéket:

```
class RobotMain(object):
    def __init__(self, robot, **kwargs):
        self.alive = True
        self._arm = robot
        self._tcp_speed = 600
        self._tcp_acc = 5000
        self._angle_speed = 8000
        self._angle_acc = 1000
        self._variables = {}
        self._robot_init()

    def _robot_init(self):
        self._arm.clean_warn()
        self._arm.clean_error()
        self._arm.motion_enable(True)
        self._arm.set_mode(0)
        self._arm.set_state(0)
        time.sleep(1)
            …
```

Jól látható, hogy vannak üres zárójelek, melyek több dologra is utalhatnak:

- Ha egyenlőségjel után áll a zárójel, akkor még csak az adatszerkezet típusa van meghatározva. Pythonban a tömbök típusának meghatározására többféle zárójelezések használhatók:
    - list [ ]: Rendezett, módosítható, különböző típusok tárolhatók benne, indexekkel lehet az elemeire hivatkozni.
    - tuple ( ): Listához hasonló, de elemei nem változtathatók.
    - set (halmaz) { }: Olyan tömb típus, amelyen nincs rendezés, elemek gyűjteménye. Ebből kifolyólag el lehet belőlük elemeket távolítani, hozzájuk is lehet adni, de az indexhez kapcsolódó műveleteknek nincs értelmük. Különböző halmazműveletek végezhetők velük, például unió, metszet, különbség és szimmetrikus különbség. Ezek a műveletek az union(), intersection(), difference() és symmetric_difference() metódusokkal végezhetők el.
    - dictionary { }: Egy kulcs-érték párokat tartalmazó adatszerkezet. A dictionary nagyon hasznos, amikor gyors hozzáférést szeretnénk biztosítani az értékekhez kulcsok alapján.
- Ha közvetlenül kifejezés után álló kerek zárójel van, akkor pedig metódusról vagy függvényről beszélünk, de a jelenlegi kód esetében túlnyomórészt az xArm-Python-SDK előre meghatározott metódusai kapnak helyet. A gyártó cég GitHub oldalán található meg a metódusok listája, illetve az API dokumentációban útmutató azok argumentumairól.

A kapott, legenerált bevezető kód tartalmaz még a leírtakon felül további definiált metódusokat, melyek a regiszterek állapotváltozásairól várnak visszacsatolást (_error_warn_changed_callback, _state_changed_callback, _count_changed_callback), továbbá jelzi a robotkarhoz csatlakozás sikerességét.

A legenerált kód utolsó sorait a „run” metódus (azon belül a kivételkezelés: try, expect, finally), valamint egy speciális Python konstrukció foglalja magában, mely a kód végén olyan programot indít, amely kapcsolódik egy robotvezérlőhöz (XArmAPI), inicializál egy „RobotMain” objektumot, majd elindítja a programot a „run” metódussal.

Az „if __name__ == '__main__':” rész biztosítja, hogy csak akkor fut le, ha a fájlt közvetlenül futtatják és nem fut le, ha csak importálják más Python fájlokból.

```
        def run(self):
                try: …
                expect: …
                finally: …
if __name__ == '__main__':
        RobotMain.pprint('xArm-Python-SDK Version:{}'.format(version.__version__))
        arm = XArmAPI('192.168.250.4', baud_checkset=False)
        robot_main = RobotMain(arm)
        robot_main.run()
```

### 3.5.4. A robotkar mozgásának meghatározása

Az előző címsorban említett „run” metódus „try” ága foglalja magában a kód túlnyomó részét, itt zajlik le a munkafolyamat lényegi része.

Itt kezdésnek a robotkar állapotát állítjuk aktívra, valamint több változónak is kezdeti értéket adunk:

```
        code = self._arm.set_cgpio_digital(8, 1, delay_sec=0)
                if not self._check_code(code, 'set_suction_cup'):
                return


                if self._arm.get_cgpio_digital(12)[1] == 1:
                        z_lerak = 0
                        z_lerak_index = 0
                        first_pickup = 0
                        i = 0
```

Azaz részletesebben, a „self._arm.set_cgpio_digital(8, 1, delay_sec=0):” szakasz beállítja az xArm robotkar 8. csatornáján, hogy bekapcsolt állapotba (1) kerüljön és nincs késleltetés az állapot beállításához (a delay_sec=0). Az „if not self._check_code(code, 'set_suction_cup'):” sor ellenőrzi, hogy az előző sorban lévő állapotbeállítás sikeres volt-e. A „self._check_code” metódus ellenőrzi a robotkar válaszát, hogy megbizonyosodjon arról, hogy a kívánt művelet sikerült. Ha az ellenőrzés kudarcot vall (az állapotbeállítás sikertelen), akkor a kód a következőkben hibát jelez.

Ezt követően az „if self._arm.get_cgpio_digital(12)[1] == 1:” sor ellenőrzi, hogy az xArm robotkar 12. digitális bemenetén észlel-e bejövő jelet. Ha a bemenet állapota 1, akkor a következő változókat állítja be, melyek fontosak lesznek a későbbekben:

- z_lerak = 0
- z_lerak_index = 0
- first_pickup = 0
- i = 0

A folytatásban a tálca méretei vannak meghatározva:

- box_x1 = -40
- box_y1 = -470
- box_x2 = 218
- box_y2 = -240,

valamint:

- „c = 14” változó, ami termékenként egyedi és a magok közti minimum távolságot határozza meg (nálunk ez az M-909 termék),
- „z_lerak = 0” változó, mely a termék lerakási magasságának kezdeti értéke,
- „z_lerak_index = 0” változó, mely egy biztonsági, több szintes lerakást tesz majd lehetővé,
- „z_felvesz = 208” változó, ami a mag felvételi magasságát határozza meg,
- „home =0” változó, melynek a home pozíció felvételében lesz szerepe,
- „first_pickup = 0”, mely az első darab felvételének egy meghatározó változója,
- illetve egy „box_ok = 0” változó, mely a későbbiekben felel majd a paletta helyes méreteiért – kezdeti értéke 0, akkor kezdi el a munkafolyamatot, ha értéke 1.

A következő „while self.is_alive and self._arm.get_cgpio_digital(0)[1]:” ciklus metódusaival vannak lekérdezve a robotkar aktuális pozíciói és ehhez viszonyítva van elvezetve a home (alaphelyzet) pozícióba. A kód, csak akkor fut tovább, ha ez az állapot igaz.

Aktuális pozíció:

```
if home == 0:
        actual_pos = arm.get_position()
        actual_pos_xyz= actual_pos[1]
        actual_pos_x=float(actual_pos_xyz[0])
        actual_pos_y=float(actual_pos_xyz[1])
        actual_pos_z=float(actual_pos_xyz[2])
```

home pozíció:

```
        safe_position_x =float(144.20000)
        safe_position_y =float(334.70000)
        safe_position_z =float(292.40000)
        print("safe positions", safe_position_x, safe_position_y, safe_position_z)
        print(actual_pos_x, actual_pos_y,actual_pos_z).
```

Fontos része a kódnak a megfelelő alaphelyzetbe állítás és itt a hangsúly a „megfelelő”-n van, mert nem mindegy milyen pozícióból, milyen útvonalon áll be alaphelyzetbe a robotkar. Kalkulálni kell az esetleges akadályokkal, hogy ne ütközzön semminek. Ezt a következő paraméterek vizsgálatával tudjuk megtenni:

- Ellenőrzi, hogy az aktuális pozíció magassága (z-koordináta) alacsonyabb-e, mint a biztonságos, home pozíció magassága. Ha igen, a robotot feljebb emeli.
    - if actual_pos_z < safe_position_z :
        print("z is lower")

- Valamint további feltétel vizsgálatokat végez, majd mindegyik után lekéri az aktuális pozíciót és vizsgálatok eredményeitől függően elvégzi az alaphelyzetbe állítást.
    - if actual_pos_y != safe_position_y :
        print("y is not equal")
    - if actual_pos_y > 0 :
        print("y is more than 0")
    - else:
        print("y is less than 0")
    - if actual_pos_z < 400:
        print("z is less than 400")
    - if actual_pos_x != safe_position_x :
        print("x is not equal")

A termék felvételi és lerakási pozíciójának területei minden esetben egyvonalban helyezkednek el. Ebben az esetben fontos megmondani a robotkarnak, hogy a megközelítőleg 180 fokos visszafordulást melyik irányban kezdje meg. Ez úgy lett megvalósítva, hogy a megfelelő fordulási irányba plusz lépésként meg lett határozva egy pozíció, amin keresztül haladva jut el a kívánt helyre.

A folytatásban a robotkar ellenőrzi, hogy a megfogó zárt állapotban van-e, feltételezve, hogy termék van a megfogóban van:

```
gripper_pos = arm.get_gripper_position()
gripper_width = gripper_pos[1]
if gripper_width < 700:
    print("Gripper is closed", gripper_width)
```

Amennyiben a megfogó zárt, a következő mozgási műveleteket hajtja végre a robotkarral:

- Először mozgatja a robotkart jobbra a [261.3, 284.3, 65, -177.9, 2.2, 90] pozícióba.
- Ezután megnyitja a megfogót (set_gripper_position(700)), várva a művelet befejezéséig (wait=True).

- Vár egy másodpercet (set_pause_time(1)), majd a robotkarral középre ([91.6, 334.7, 333.1, -178, -0.8, 90]) mozog.

Az összes ezen művelet végrehajtása során ellenőrzi a „self._check_code” függvénnyel, hogy az elvégzett műveletek sikeresek voltak-e.

A kód következő szakasza felkészíti a robotkart a termék felvételére a meghatározott pozícióban és állapotban:

- „set_cgpio_digital(9, 0, delay_sec=0):” a metódus beállítja a robotkar 9-es digitális bemenetét "0"-ra, ezáltal a körasztal, amin a termékek helyezkednek el foroghat, valamint nincs késleltetés a művelet során.
- „set_gripper_position(700, wait=True, speed=5000, auto_enable=True):” ezt követően a kódrészlet beállítja a robotkar megfogójának állását. A „set_gripper_position(700)” paranccsal a fogó 700 egységnyi szélességűre nyit ki. Az "auto_enable=True" azt jelzi, hogy a megfogó automatikusan be van kapcsolva. Az "auto_enable" segíthet abban, hogy a megfogó automatikusan bekapcsoljon, amikor eléri a kívánt pozíciót. Az "auto_enable" beállításától függően a megfogó automatikusan be- vagy kikapcsol a kívánt pozíció elérésekor. A "speed" paraméter meghatározza a fogó mozgásának sebességét.
- „set_position(*[141.2, 536.7, 292.4, -178.7, 0.8, 93.3], speed=self._tcp_speed, mvacc=self._tcp_acc, radius=5.0, wait=False):” sor beállítja a robotkar pozícióját. A megadott koordináták (X, Y, Z) és Euler szögek (YAW, PITCH, ROLL) segítségével a robotkar a kívánt helyre mozog. A "speed" paraméter meghatározza a mozgás sebességét, a "mvacc" a mozgásgyorsulást, a "radius" pedig az elmozdulás milyen vonalon történik. Az „wait=False” azt jelenti, hogy nem vár a mozgás befejezésére, hanem aszinkron módon végzi el.
- „home = 1:” végül a "home" változót beállítjuk „1”-re, amely azt jelenti, hogy a home pozícióba történő mozgás végrehajtódott és nincs szükség újra végrehajtani.

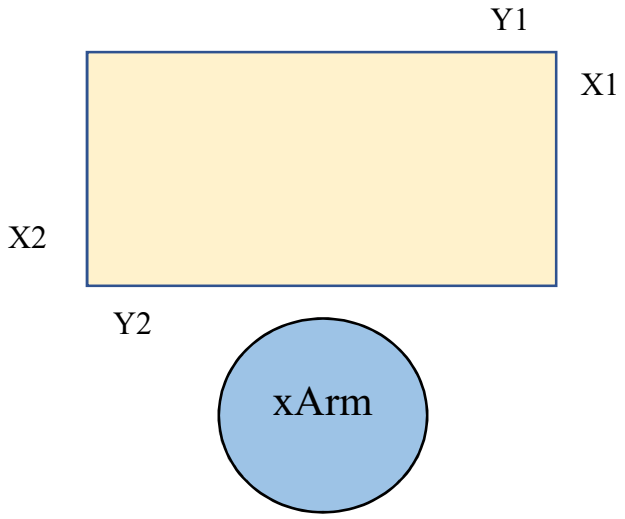
A kód következő soraiban van meghatározva a jó és a rossz termékek válogatása. Itt a robotkar két különböző bemenetére jelzések érkeznek, ez alapján tudja eldönteni hova rakja a termékeket:

- „if self._arm.get_cgpio_digital(11)[1] == 1:” – hibás termék esetén
- „if self._arm.get_cgpio_digital(10)[1] == 1:” – jó termék esetén.

Hibás termék esetén jel érkezik a robotkar vezérlőjének 11-es bemenetére és a „set_position” paraméterezéseivel elmozgatjuk a kart a selejt láda fölé, majd a megfogó nyitásával („set_gripper_position(700, wait=True, speed=5000, auto_enable=True)”) elengedjük ott a terméket.

Jó termék esetén már jóval összetettebb a folyamat:

- jel érkezik a robotkar vezérlőjének 10-es bemenetére,
- a robotkar a felvételi pozícióba lép, megfogja az első magot,
- a korábban bevezetett „first_pickup” változó értéke 0-ről 1-re változik, jelezve, hogy megtörtént az első darab termék felvétele és a folytatásban szereplő kalkulációkat már legközelebb nem kell elvégeznie,
- a megfogó az összezárás értékéből megállapítja a mag átmérőjét és az így létrejött értéket egy „d” változóban letárolja, ami majd felhasználásra kerül a további számításokkor.




A továbbiakban az X1, X2 koordináták egymáshoz viszonyított helyzetét kell definiálni, figyelembe véve a számok lehetséges előjeleit:

```
if box_x1 >= 0 and box_x2 >= 0:
        x_len = box_x2 - box_x1
elif box_x1 < 0 and box_x2 >= 0:
        x_len = box_x2 + abs(box_x1)
elif box_x1 < 0 and box_x2 <= 0:
        x_len = abs(box_x1) - abs(box_x2)
if x_len == 0:
        print ("Box x is zero")
if x_len != 0:
        print("x_len",abs(x_len))
col_number = (abs(x_len)+ c) // (d + c)
        print ("col_number",col_number)
```

Majd ugyanúgy az y1, y2 koordináták egymáshoz viszonyított helyzetét is meg kell határozni:

```
if box_y1 >= 0 and box_y2 >= 0:
        y_len = box_y2 - box_y1
elif box_y1 < 0 and box_y2 >= 0:
        y_len = box_y2 + abs(box_y1)
elif box_y1 < 0 and box_y2 <= 0:
        y_len = abs(box_y1) - abs(box_y2)
f y_len == 0:
        print ("Box y is zero")
```

```
if y_len != 0:
        print("y_len",abs(y_len))
        row_number = (abs(y_len) + c) // (d + c)
        print ("row_number",row_number)
```

Az „x_len” és „y_len” változók meghatározását követően szükség van még egy ún. „clearance”, azaz egy plusz engedélyezett távolság meghatározására a termékek között, mely első körben az első lerakott termék és a tálca széle közti távolságot szabja meg, valamint az első és második lerakott termék köztit, majd így tovább. Ez fontos szempont, mivel muszáj adott távolságot hagyni a magok között. A kapott értékeket a „round” függvénnyel, 3 tizedesjegyre kerekítve kapjuk meg.

```
if col_number == 1:
        clear_x = round(((abs(x_len) - (col_number * d)) / 2), 3)
if col_number == 2:
        clear_x = round((abs(x_len) - (col_number * d)), 3)
if col_number > 2:
        clear_x = round(((abs(x_len) - (col_number * d)) / (col_number - 1)), 3)
print("clear_x",clear_x)
if row_number == 1:
        clear_y = round(((abs(y_len) - (row_number * d)) / 2), 3)
if row_number == 2:
        clear_y = round((abs(y_len) - (row_number * d)), 3)
if row_number > 2:
        clear_y = round(((abs(y_len) - (row_number * d)) / (row_number - 1)), 3)
print("clear_y",clear_y)
```

Sikeres kalkulációt követően a folytatásban a „box_ok = 1” értékre állítjuk és meghatározható a tálca kiosztása. Sikertelen számolt adatok esetén a folyamat leáll.

A sikeres fázistól elkezdődik a robotkar tálca fölé mozgatása a „set_position” és „set_servo_angle” metódusok paraméterezéseivel, mint pl.:

- self._arm.set_position(*[141.2 , 536.7, 292.4, -178.7, 0.8, 93.3], speed=self._tcp_speed, mvacc=self._tcp_acc, radius=10.0, wait=False)
- self._arm.set_servo_angle(angle=[0.0, -42.8, -35.6, 0.3, 81.4, 0], speed=self._angle_speed, mvacc=self._angle_acc, wait=False, radius= 10)

A tálcára pakolás fontos biztonsági intézkedése, hogy bár a pakolás elméletileg egy szinten történik, létre lett hozva egy „z_lerak_index”, mely az alap pozíció magasságán még két ütemben emel 18-as értékkel, elkerülve az esetleges törést.

```
if  i == j:
        i = 0
        j = col_number * row_number
        row = 1
        column = 1
        z_lerak_index += 1

        if z_lerak_index < 3:
                z_lerak = z_lerak + 18
                print("Actual z iteration, level:", z_lerak_index, z_lerak)
        else:
                z_lerak = 0
                z_lerak_index = 0
```

Innentől már elkezdődik a lerakási pozíciók kiszámítása:

```
        while i<j:
            if row < row_number + 1:
                if row == 1:
                    move_y = box_y1 + d/2
                else:
                    move_y = box_y1 + d/2 + d * (row - 1) + clear_y * (row - 1)
            if column < col_number + 1:
                if column == 1:
                    move_x = box_x1 + d/2
                    column += 1
                else:
                    move_x = box_x1 + d/2 + d *(column - 1) + clear_x*(column- 1)
                    column += 1
```

Ezt követően a meghatározott értékek alapján elkezdődik a tálcára pakolás a „set_position” és „set_gripper_position” metódusokkal, valamint az „i = i+1” gondoskodik a léptetésről és a ciklust egy „brake” utasítással zárjuk, mivel lehet, hogy a következő termék nem jó és ne pakolja folyamatosan. Ezek után visszalép ismét a megfelelő útvonalon, félkörívben a következő termékért mindaddig, amíg a korábban már említett módon jelet nem kap az előző munkaállomás PLC-jétől, hogy a kívánt darabszám a tálcán van, valamint a körasztalnál lévő szenzor érzékel-e terméket.

Mint ezen címsor elején említettem, a kód majdnem egésze a „run” metódus „try” ágában fut le, de ezen ág végeztével a kivételkezelés másik két ágával záródik a program fő metódusa, ami az „except” és a „finally”, ahol állapotváltozásokról kapunk visszajelzést, illetve a 8-as digitális port 0-ra állításával, offline állapotba rakjuk a robotkart:

```
    except Exception as e:
        self.pprint('MainException: {}'.format(e))
    finally:
        self._arm.release_error_warn_changed_callback(self._error_warn_changed_callback)
        self._arm.release_state_changed_callback(self._state_changed_callback)
        code = self._arm.set_cgpio_digital(8, 0, delay_sec=0)
        if not self._check_code(code, 'set_suction_cup'):
                return
```

# 4. Összegzés

Záródolgozatom célja az volt, hogy bemutassa a Python programozási nyelv és az xArm robotkar közötti hatékony együttműködést az ipari automatizáció és robotika terén. A Python nyelv kiváló választásnak bizonyult a robotprogramozás szempontjából, mivel könnyen tanulható, széles körben használt és sokoldalú. Mindezek alapján úgy gondolom, a Python és az xArm robotkarok együttműködése sok ígéretes alkalmazási területet nyit meg a jövőben, valamint ezek kombinációja hozzájárulhat a gyártás hatékonyságának növeléséhez és az automatizált folyamatok fejlesztéséhez az ipar más ágazatában is.

A folyamatot még tovább fejlesztené, segítené egy kamera applikálása a robotkarra, mert a kamera segíthetne pl. munkaterületet és terméket azonosítani, pozíciókat meghatározni. Ennek a megvalósítása még nem kezdődött le, de tervek között szerepel.

# 5. Internetes irodalom

1. Kelas Karyawan: (Python (programming language)). On-line: http://kelas-karyawan-bali.kurikulum.org/IT/en/2420-2301/Python_3721_kelas-karyawan-bali-kurikulumngetesumum.html

# 6. Ábrajegyzék

1. ábra: xArm robotkar és hatóköre ..... 6
2. ábra: Megfogó ..... 6
3. ábra: AC Control Box ..... 7
4. ábra: AC Control Box portjai ..... 7
5. ábra: UFactory Studio ..... 8

# NYILATKOZAT

## a záródolgozat nyilvános hozzáféréséről és eredetiségéről

| | |
|---|---|
| A hallgató neve: | Czene Attila |
| A Hallgató Neptun kódja: | NRFI26 |
| A dolgozat címe: | Ipari automatizálás Python nyelv használatával |
| A megjelenés éve: | 2023 |
| A konzulens intézetének neve: | Műszaki Intézet |
| A konzulens tanszékének a neve: | Alkalmazott Informatikai Tanszék |

Kijelentem, hogy az általam benyújtott záródolgozat egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, és az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe. Tudomásul veszem, hogy a megvédett és

- nem titkosított dolgozat a védést követően
- titkosításra engedélyezett dolgozat a benyújtásától számított 5 év eltelte után

nyilvánosan elérhető és kereshető lesz az Egyetem könyvári repozitori rendszerében.

Kelt: Gyöngyös 2023.11.05.

Hallgató aláírása

# NYILATKOZAT

Czene Attila (név) (hallgató Neptun azonosítója: NRFI26) konzulenseként nyilatkozom arról, hogy a záródolgozatot[1] áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A záródolgozatot/szakdolgozatot/diplomadolgozatot/portfóliót a záróvizsgán történő védésre **javaslom / nem javaslom**[2].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[3]

Kelt: 2023. év november hó 06. nap

belső konzulens

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő aláhúzandó.
[3] A megfelelő aláhúzandó.