



# Drón vezérlés kommunikációs követelményeinek optimalizálása

ERICSSON - BME 5G KÖZÖS KUTATÁSI ÉS FEJLESZTÉSI PROJEKT  
2018-1.3.1-VKE-2018-00005

2020 január

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
1.1. Előzmények . . . . .	3
1.2. A feladat célja . . . . .	3
1.3. Feladat indokoltsága . . . . .	4
1.3.1. Gyártóipar . . . . .	4
1.3.2. Szállítás . . . . .	5
1.3.3. Mezőgazdaság . . . . .	5
1.3.4. Szórakoztatóipar . . . . .	6
1.4. Használt kifejezések . . . . .	6
<b>2. Felhőrendszer kiválasztása</b>	<b>8</b>
2.1. Virtualizációs alapok . . . . .	8
2.1.1. Hypervisor . . . . .	8
2.1.2. Konténer . . . . .	8
2.1.3. Docker . . . . .	9
2.2. Docker Compose . . . . .	10
2.3. Docker Swarm . . . . .	10
2.4. Mesos . . . . .	11
2.5. Kubernetes . . . . .	11
2.5.1. Architektúra . . . . .	11
2.5.2. Komponensek . . . . .	11
2.6. Keretrendszer meghatározása . . . . .	12
<b>3. Robotvezérlés környezete</b>	<b>14</b>
3.1. Robotirányítás . . . . .	14
3.2. Robot operációs rendszer - ROS . . . . .	14
3.3. Kommunikáció - Mavros, Mavlink . . . . .	15
3.4. Vezérlő - PX4 . . . . .	15
3.5. Szimulációs környezet - Gazebo . . . . .	16
3.6. Együttes működés . . . . .	17
<b>4. Virtuális gépen kialakított tesztrendszerek</b>	<b>19</b>
4.1. Azonos konténerrel egy drónvezérlés . . . . .	19
4.2. Két VM-en több drón szimulációja és vezérlése . . . . .	20
<b>5. QoS sztohasztikus becslése</b>	<b>21</b>
5.1. Tömegkiszolgálási modell . . . . .	21
5.2. Várakozási idő várható értéke . . . . .	22
<b>6. Távoli Robotvezérlés Optimalizálása</b>	<b>23</b>

Összegzés	24
Irodalomjegyzék	25

# 1. fejezet

## Bevezetés

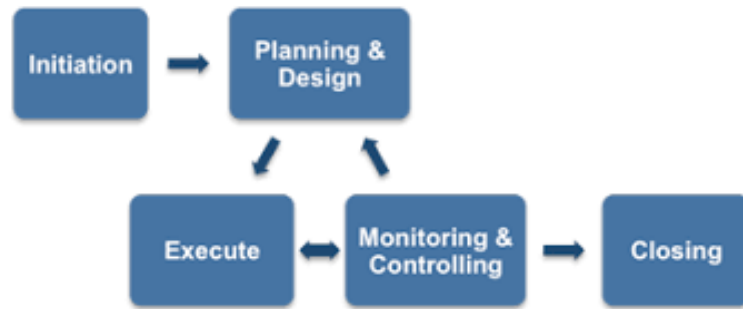
Ebben a fejezetben betekintést nyújtok az olvasó számára a feladatról, mint problémáról, hogy az miért is indokolt és milyen iparágakat érinthet, mi a pontos célja a feladatnak és mik a főbb kritériumok. Tisztázni szeretném a képet az olvasóban, hogy mik is a projekt előzményei amely alapján kialakult a feladat-specifikáció és mi is az a motiváció ami pontosan ezt a projektet eredményezte.

### 1.1. Előzmények

A témám címe felhő alapú drónvezérlés, aminek tükrében többféle ágazatból is belecsöppenhettem ennek a projektnek a feladatköreibe. Ugyanis a felhőtechnológiák, a robotvezérlés és az irányítástechnika spektrumot is lefedi a feladat, azonban én a felhőtechnológiák felől közelítem meg. Villamosmérnök BSC 5. félévében témalabor keretei között csatlakoztam a Távközlési és Médiainformatika Tanszék (TMIT) Felhő alapú hálózatok szakmai műhelyéhez, ahol azóta is minden félévben végzek projektfeladatot. Ehhez a motivációm egyszerű volt, mivel már az alapképzés eleje óta érdekelnek azon architektúrák, ahol erőforrás menedzselés és központi irányítás alatt, például master-slave módon működnek. A felhőalkalmazások ipara is egy ilyen terület, ahol bizonyos szolgáltatásokat minél szélesebb körben szeretnénk értékesíteni igény szerint, azonban valamilyen központi vezérlés működteti automatikusan az erőforráskiosztást, hozzáférést. A műhelyben végzett feladataim között felmerült a Kubernetes elosztott konténerkezelő rendszer, a Kubless serverless architektúra, felhő alapú beszédfelismerő rendszerek és az OpenStack virtuális gép menedzselő rendszer, amihez egy archiválási megoldást fejlesztettem szakdolgozatként. A felhő alapú drónirányítás téma a diplomatervem kezdetén került szóba, előtte nem foglalkoztam mélyebben robotirányítással és vezérléssel, így remélhetőleg ezekből a területekből is nagy tapasztalatot fogok szerezni a feladat befejeztén.

### 1.2. A feladat célja

A feladat sokféleképpen általánosítható, hiszen egy felhőrendszerből igen sokféle ipari folyamatot lehet irányítani, csupán a megvalósítást kell kivitelezni. Egy drón irányítása sem különbözik, lehetne a cél robotkar vezérlése, gyártósor ütemezése, automatikus kötőpályás közlekedési eszköz irányítása vagy akár önvezető autók feletti vezérlés. Azonban a témámban nem egy drón irányítását, hanem több, akár tíz-húsz, de akár százat is elérheti az a teszteset amire szeretnék megoldást adni. Egy ilyen rendszert irányító felhő architektúrában rugalmasabb és erőforrás takarékosabb üzemeltetést lehet elérni. A számításkapacitás optimalizáció nem minden amit ebben a helyzetben az ipar megkíván, hanem azokat a kommunikációs kapcsolatokat, amelyek szigorú minőségi elvárásoknak (Quality



**1.1. ábra.** Kritikus mérföldkövek folyamata [5]

-of Service – QoS) kell megfelelniük, ezért bonyolultabb kezelni. A feladat feltérképezni a modern felhőrendszereket, amelyekkel meg lehet valósítani efféle iparban is használható vezérlési technikát QoS feltételek mellett. Tehát a projektnek három kritikus tervezési mérföldköve van. Ezeknek kapcsolata látszik a 1.1. ábrán.

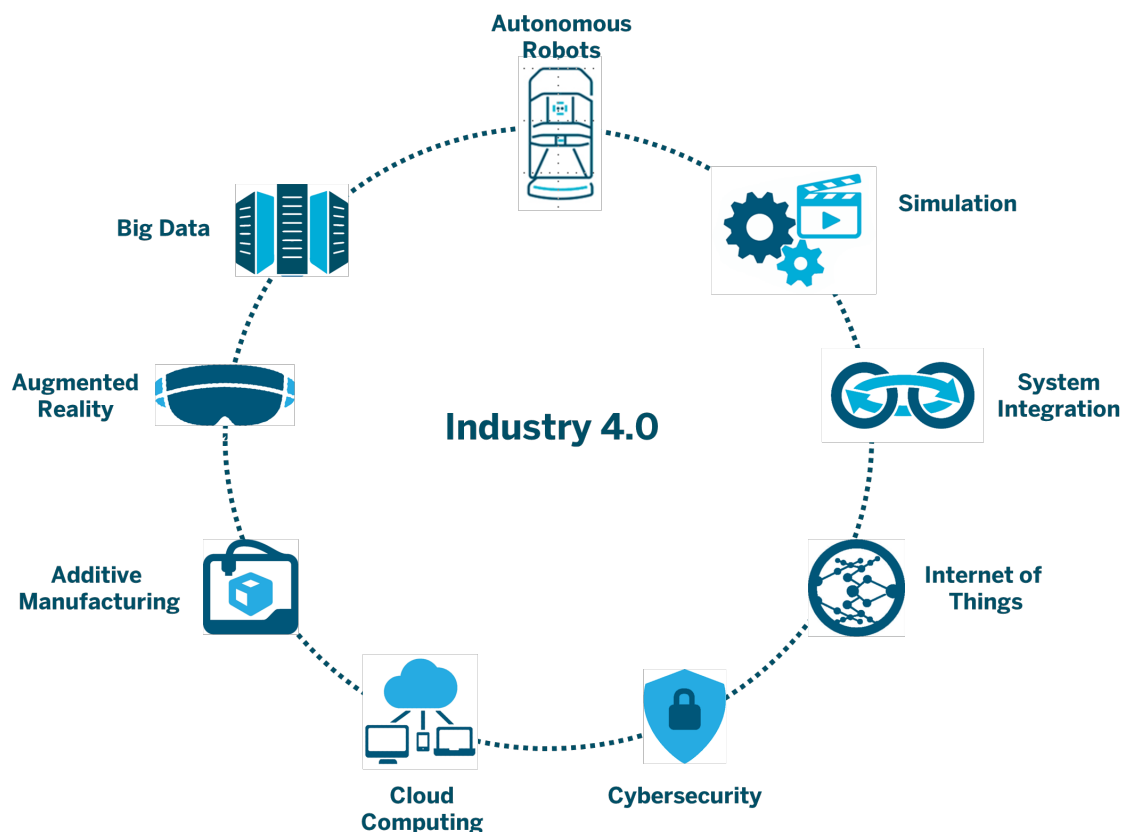
1. Kiválasztani a megfelelő felhőrendszert, amiben megvalósítható  $R \in [1, 100]$  drón, vagy általánosan, valamilyen vezérelhető eszköz (robot) irányítása.
2. Áttekinteni drónvezérlést támogató szoftver megoldásokat, kiválasztani egy olyan környezetet, mellyel távvezérlési funkciók megvalósíthatók. Megtervezni ebben a felhőrendszerben több drón irányítását és jelfeldolgozását, automatikusan és biztosítva, hogy hiba esetén visszaálljon a működő állapot.
3. A meglévő rendszer analízálása, optimalizálása és egy QoS feltétel felállítása.

Tehát a dolgozat célja nem csak a megvalósítás, hanem  $R$  számú tízes-száz-as nagyságrendű robotok irányításának a QoS feltételét és validációját is megadja. Ezt a feltételt valamilyen  $f_{QoS}(R, C) \in N^2$  függvény fogja megadni, melynek paraméterei  $R$  a robotok száma és  $C$  a számítási erőforrás. Ha számításról beszélünk, akkor a felhőszolgáltatások iparában kiválasztunk egy valamilyen optimális  $c = \text{RAM}[\text{GB}]/\text{VCPU}$  arányt mely az alkalmazásunkhoz megfelelő és annak  $n \in N^+$ ,  $C = c \cdot n$  egész számú többszöröse lesz a számítási kapacitás amire méretezünk. Vagyis valamilyen 2 dimenziós tervezési teret adunk meg a kívánt szolgáltatási minőség eléréséhez. Az ilyen jellegű feladatokban a szolgáltatás minősége általában a válaszidőt szokta jelenteni  $n \cdot 10$ -es nagyságrendben. Természetesen ez nem egy túl pontos modell, de egy mérnöki kapacitástervezés számára kellő kiindulási alapot adhat méretezni a felhőrendszert, esetleg skálázhatósági lehetőségeket is figyelembe véve. Ezen kívül mivel nagyszámú eszközről beszélünk nem tekinthetünk el a közegátviteli technológiáról. Azt sejthetjük, hogy a mai elterjedt távközlő rendszerek nem alkalmasak akár száz fölötti felhasználóval, mondjuk robottal folyamatos kommunikációt tartani megfelelő QoS-el. Ezért megnézzük, hogy a mai modern felhőrendszerekkel és az 5G között milyen kapcsolatot lehet alakítani és mik a feltételek egy ilyen rendszerben való üzemeltetéshez.

## 1.3. Feladat indokoltsága

### 1.3.1. Gyártóipar

A nagyszámú eszközök irányítása számtalan szektorban elterjedt és alkalmazható. Akár beszélve a gyártó szektorról, ahol nagyon sok folyamatot robotok látnak el és legyen az bármilyen robot, az valószínűleg irányítható felhőből. Persze sokszor a gyártók egy drágább,



1.2. ábra. Ipar 4.0 komponensei [1]

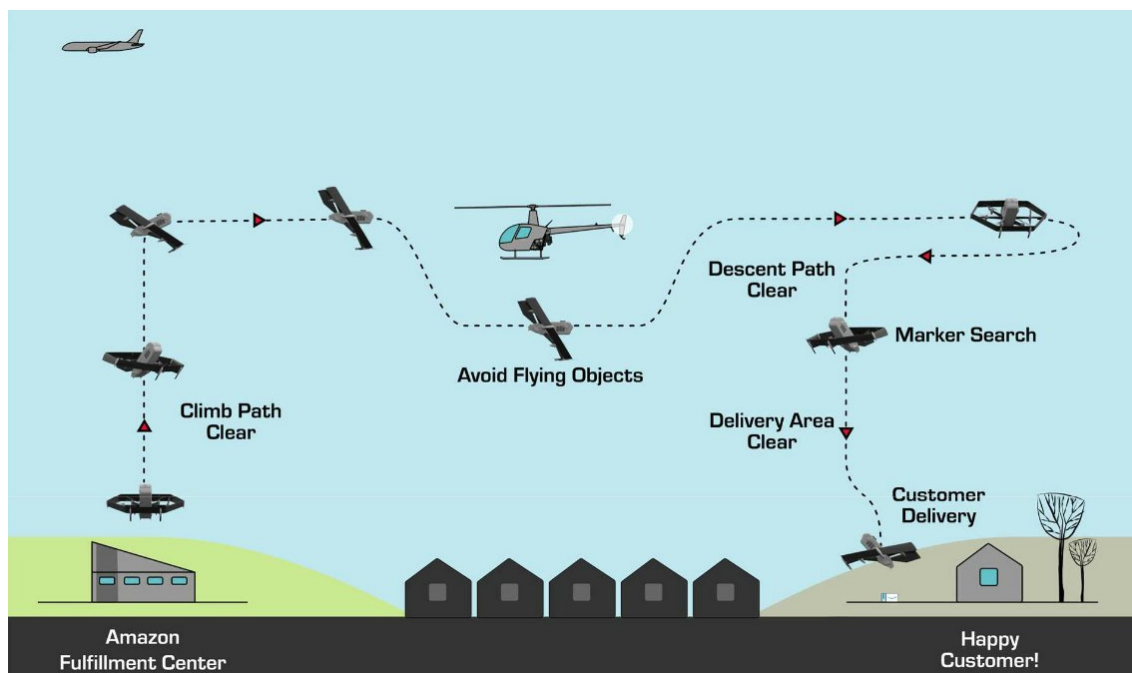
de kapacitásban túlbiztosított lokális rendszerről irányítják a gyártórobotjaikat, mivel a felhős megoldások még annyira nem elterjedtek ebben a célfelhasználásban. Azonban egy ilyen megoldással rengeteg erőforrás megtakarítható. A 1.2. ábrán láthatóak az Ipar 4.0 főbb komponensei és könnyen meggyőződhetünk róla, hogy egy mai innovatív ipari környezetben inkább a felhő alapú megoldásokat választják a skálázhatóság és a biztonságuk miatt. Ebben az esetben felhőről általában számítási kapacitásról beszélünk, azonban később kitérünk a szolgáltatások kategóriáira.

### 1.3.2. Szállítás

Nem csak a gyártóiparban lehet elképzelni sok robot irányítását, hanem például szállítás és utazás terén is. A Műegyetemhez legközelebbi metróvonal már évek óta önvezérlőként működik, ugyan egyelőre a hossza befejezetlensége miatt csak tíz körüli metrószerelvény működik egyszerre, azonban ez is egy olyan példája a robotirányításnak, amit minden nap észlelhetünk. Az Amazon házhozszállító cég, amelynek egyébként az AWS (Amazon Web Services) leányvállalata a világ egyik legnagyobb felhőszolgáltatója, már tesztel drónokat, amelyek betöltik a csomagházhozszállítás szerepét. A levegőben való csomagszállítás egyik problémája a 1.3. ábrán látható.

### 1.3.3. Mezőgazdaság

Az agrárvilágban is el lehet képzelni sok robotot kezdve a vetéstől az aratásig, azonban a drónoknak kifejezett szerepe akad a jövőben ebben az iparágban. Használják ma már automatizált drónokat permetezésre, időszakos állomány megfigyelésre vagy akár kombájn útjának a felderítésére is. Ahogyan a gyártósoroknál, itt is optimalizálhatunk több robot



1.3. ábra. Amazon házhozszállítás problémája drónnal [15]

irányítás esetén felhőrendszerrel. A dolgozatban arra keresünk megoldást, hogy ezt milyen eszközrendszerrel érdemes tervezni.

#### 1.3.4. Szórakoztatóipar

Idáig kiderült, hogy rengeteg iparágban alkalmazhatóak a tömegesen irányított robotok, amiket kreativitással könnyű bővíteni. Azonban a szórakoztatóiparban is megjelentek már a tömegesen irányított drónok. Egy jellegzetes példája ennek amerika legnagyobb sport-eseménye a Super Bowl 2017-es döntője, ahol 300 drónt használtak fel az égboltra való fényfestéshez a szünetben lévő koncerthez. Ez az esemény egy pillanatfelvétele a 1.4. ábrán látható és az esemény technikai előkészületeiről a [4] cikkben lehet olvasni. Persze ebben a példában egy pár perces feladatról beszélünk az irányított robotok számára, a felhős megoldással pedig egy hosszútávú optimalizációt szeretnénk megadni.

### 1.4. Használt kifejezések

A hálózati-, virtualizációs- és robotiparban rengeteg rövidítés, mozaikszó és kifejezés létezik, amit főként csak azok ismernek, akik jelentősebben mélyültek el ezen iparág területén. Két csoportra bontom a diplomatervemhez használt kifejezéseket:

1. Azon szakmai kifejezések, amik elterjedtek a mérnöki szakmákban és mondjuk a BME VIK tetszőleges hallgatója, bármely, nem infokommunikációs szakosodás mellett is nagy valószínűséggel ismer és nem kell ismertetnem a tanulmányban. Pár példa a kategóriában, amikre külön nem térek ki, nem oldom fel a szövegben, ilyenek az IPv4, NAT, hálózati réteg, HTTP, CPU, for ciklus.
2. Azokat a kifejezéseket amik pedig a témához, szakterülethez kapcsolódnak, például a kifejezetten hálózati kommunikáció, virtualizáció, robot, irányítás iparágakba tartozó kifejezések, amiket nem általános mérnöki ismeretek, azokat a szövegben az első használatnál kifejtem, továbbá itt összegyűjtöm.



**1.4. ábra.** Lady Gaga 300 drónnal a háttérben a 2017-es Super Bowl döntőjének félideje alatti koncerten [6]

A tanulmányban használt speciális kifejezések:

- QoS (Quality of Service) - A szolgáltatás minőségének a biztosítása
- Virtuális gép (Virtual Machine, VM) - Virtualizált önálló teljes értékű operációs rendszer gazdagépen (host-on)
- KVM (Kernel-based Virtual Machine) - Kernelhez elosztott időben hozzáférő virtuális gép
- Host OS - Hosted virtualizáció esetén a gazdaoperációs rendszer
- Guest OS - Hosted virtualizált operációs rendszer a host OS fölött
- Hypervisor - A hardveren való virtualizációt megvalósító szoftver
- Node - A felhőrendszer egy fizikai eszköze
- Cluster - A felhőrendszerbe csatolt fizikai eszközök kapcsolata
- Volume - Szolgáltatás/VM háttértára
- Vertikális skálázás - Node hozzáadása a cluster-hez
- Horizontális skálázás - Node fejlesztés
- GCS (Ground Control Station) - Földi irányítóállomás



## 2. fejezet

# Felhőrendszer kiválasztása

A számítás alapú felhőrendszereknek két alapja van, a teljes virtualizált önálló operációs-rendszer (VM, mint Virtual Machine) vagy pedig a konténerizált, önálló lebutított virtualizált operációs rendszer szolgáltatás szinten. Egy platform virtualizációja olyan virtuális gép létrehozását jelenti, amely egy valós operációs rendszerű számítógépként működik. A virtuális gépeken végrehajtott szoftverek elkülönülnek az alapul szolgáló hardverforrásoktól.

### 2.1. Virtualizációs alapok

A KVM (Kernel-based Virtual Machine, azaz kernel alapú virtuális gép), egy teljes virtualizációs megoldás amely képes kihasználni az újabb processzorokban rejlő hardveres virtualizáció képességet. Magában foglal egy betölthető kernel modult, alkalmas egy adott operációs rendszeren virtualizált környezetben egy másik operációs rendszert futtatni. Azaz miközben a gazda operációs rendszer fut a számítógépen addig képesek vagyunk egy másik vendégrendszert indítani virtuális környezetben. Ettől ez a megoldás sokkal gyorsabb.

#### 2.1.1. Hypervisor

Szoftverben megvalósított menedzser a virtuális gépek számára. A gazdahardveren fut, lehetővé teszi a vendég gépek (guest OS) futtatását, memória kezelését és processzor ütemezését különböző algoritmusokkal. Két főbb típust különböztetünk meg.

##### Type I: bare matel

Az első típusú hypervisorok közvetlenül a hardverre vannak telepítve. Tartalmazniuk kell saját operációs rendszerüket a rendszerindításhoz, a hardver futtatásához és a hálózathoz való csatlakozáshoz. Ilyen hypervisor pl. a Microsoft Hyper-V.

##### Type II: hosted

A hosztolt hypervisorok olyan operációs rendszereken futnak, amely közvetlenül a hardverre van telepítve. Ebben az esetben szükség van egy gazda operációs rendszerre (host OS).

#### 2.1.2. Konténer

A konténerizáció egy olyan megoldás, ahol a szolgáltatásokat külön-külön letisztult operációs rendszerre telepítjük és csak azokat a szoftvereket ami az adott szolgáltatáshoz

szükséges. A konténerizáció elszeparálása a Linux kernel namespace technológiáján alapul. Működésük alapján hasonlítanak a VM-ekre, úgy is lehet mondani, hogy egy olyan VM optimalizált megoldása, amin egy applikáció fut. Egy rendes operációs rendszeren futó számítógépes program képes megtekinteni az adott rendszer összes erőforrását (csatlakoztatott eszközök, fájlok és mappák, hálózati megosztások, CPU-teljesítmény, számszerűsíthető hardver- képességek). A konténeren belül futó programok azonban csak a konténer tartalmát és eszközeit látják el.

### 2.1.3. Docker

A Docker a legelterjedtebb konténerizáció megoldás, 2013-ban kiadott technológia Windows és Linux operációs rendszerekhez. A Docker elsősorban Linuxra készült, ahol a Linux kernel erőforrás-elkülönítési jellemzőit (cgroup-okat), a névtereket és a fájlrendszert használja. Ezek lehetővé teszik a független konténerek egyetlen Linux példányán működését, elkerülve bármilyen összeférhetlenséget.

#### Hálózata

A docker konténereknek 3 féle hálózati interface-ük lehet,

- Host - kívülről elérhető
- Bridge - a docker konténerek elérik egymást
- None - nincs hálózatra kötve

#### Port forward

Docker konténerek létrehozásánál megadhatjuk a port átirányítást, így azonos alkalmazások sem akadnak össze.

```
$ docker run --name wp1 wordpress
$ docker run --name wp2 -p 80:81 wordpress
```

**2.1. lista.** Példa két WordPress szolgáltatás párhuzamos indítására a 80 és 81-es portokon

#### Volume csatolás

Alapvetően nehézkes hozzáférni a konténer belső tárhelyéhez, azonban gazda OS alatt tudunk csatolni a szolgáltatáshoz tartozó könyvtárat a host OS fájlrendszeréhez.

```
$ docker run -v /usr/local/mywordpress:/wordpress wordpress
```

**2.2. lista.** Példa volume csatolásához

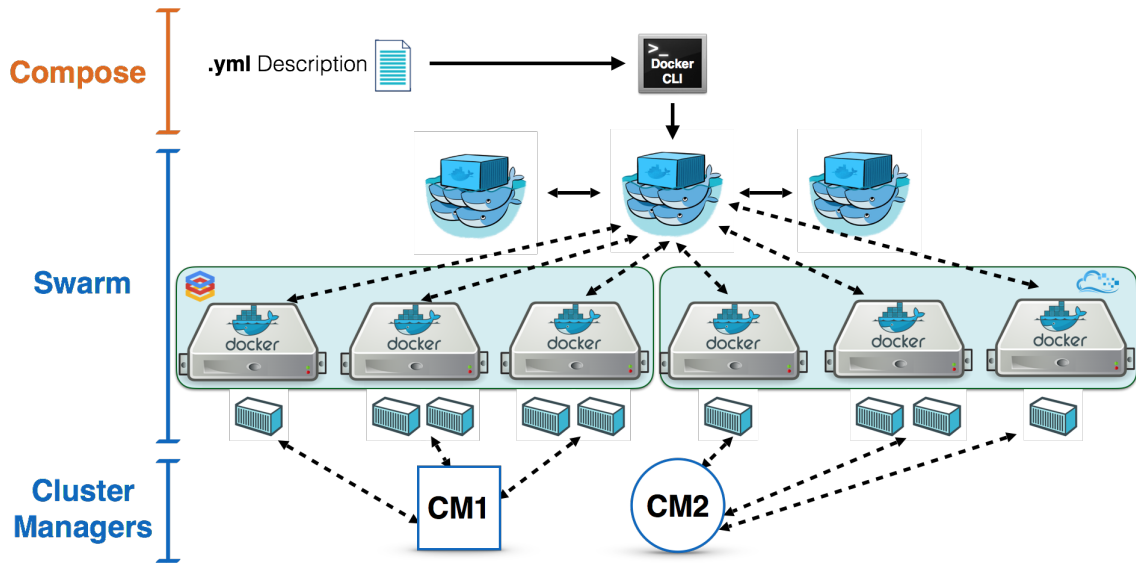
#### Példa docker konténer robotoperációsrendszerhez

A robotoperációs rendszerekről később lesz szó, azonban megadok egy példát ilusztrálva, hogyan alakítható egyszerű szolgáltatás docker konténerre, csupán egy Dockerfile nevű fájlra van szükség az applikáció főkönyvtárában (2.3. számú lista).

```
FROM ros:melodic-ros-base

RUN apt-get update && apt-get install locales -y
RUN locale-gen en_US.UTF-8
ENV LANG en_US.UTF-8

COPY . /catkin_ws/src/
WORKDIR /catkin_ws
RUN /bin/bash -c '. /opt/ros/kinetic/setup.bash; catkin_make'
RUN /bin/bash -c '. /opt/ros/kinetic/setup.bash; source devel/setup.bash'
```



2.1. ábra. Docker Swarm architektúra [11]

```
CMD ["roslaunch", "bebop_gazebo bebop_moving_helipad.launch"]
```

2.3. lista. Példa alap robotoperációsrendszer konténerizációjához

A *FROM* paranccsal a szülő konténert adom meg, amit DockerHub-ról automatikusan letölt, a *RUN* parancsokkal környezeti programokat telepítek, *COPY* az applikációhoz tartozó fájlokat másolja a konténerbe, *ENV* paranccsal környezeti változót állítok, *WORKDIR* parancs a munkakönyvtár beállítása és a *CMD* utasítás a konténerizáltan futtatandó applikáció megadása.

## 2.2. Docker Compose

A Compose egy multi-konténer Docker eszköz különböző alkalmazások együttes definiálásához és működéséhez. A Compose használatával YAML fájlban definiálni lehet különböző konténereket más-más paraméterekkel szolgáltatásainak konfigurálására. Ezután egyetlen paranccsal elindíthatóak a docker konténerek, az összes együtt működő szolgáltatás a konfigurációból. [3] A feladathoz biztosan kell használni, mivel a drónirányítás több különböző funkcióból valósul meg, így a konténer definíciója szerint érdemes minden egyes applikációt elkülöníteni és Compose-al összekötni a működésüket.

## 2.3. Docker Swarm

A Docker Swarm olyan fizikai vagy virtuális gépek (node-ok) csoportja, amelyek futtatják a Docker alkalmazást, és amelyeket együttesen egy swarm-t alkotnak, hogy összekapcsolódjanak egy clusterként. Miután egy csoport node-ot összekapcsoltak, továbbra is futtható rajtuk a Docker vezérlőparancsok, ezeket most a worker node-ok hajtják végre. A klaszter tevékenységét egy swarm manager irányítja, és a klaszterhez csatlakozó gépeken osztja ki a feladatokat. A docker swarm manager és worker node-ból áll és alkalmazható a swarm-ra a docker compose is (2.1. ábra).

## 2.4. Mesos

A Mesos az Apache szoftvere, mely egy absztrakt környezetet hoz létre CPU, RAM és háttértárral, akár 10.000 node-ig. A Mesos egy nyílt forrású cluster kezelő szoftver. Alkalmazásokat kínál API-kat az erőforrás-kezelésre és az ütemezésre a cluster-en keresztül. A Mesos rugalmasságot és az elhaló alkalmazások újraindítását biztosítja. Lehetővé teszi a framework-ök számára, hogy ütemezzék és végrehajtják a feladatokat API-n keresztül. A Mesos-architektúra egy master node-ból áll, amely az egyes node-okon futó worker-eket és a worker-eken futó framework-öket kezeli, melyeken a szolgáltatások futnak. Az összes alkalmazásdefiníció egy JSON-fájlban található, amelyet továbbítanak a Mesos / Marathon REST API-hoz.

## 2.5. Kubernetes

Egy docker rendszerben a definiált konténer egy példánya futtatható a *docker run* utasítással. Kuberneteset használva a kube control CLI-on keresztül akár ezer példány is futtatható ugyanannak az alkalmazásnak.

```
$ kubectl run --replicas=100 wordpress
```

**2.4. lista.** Példa 100 alkalmazás indítására

A futtatott alkalmazásokat egy másik paranccsal felskálázhatjuk.

```
$ kubectl scale --replicas=200 wordpress
```

**2.5. lista.** Példa alkalmazás skálázására

Tehát egy kihasználtsági emelkedő esetén egyszerűen lefoglalhatunk több erőforrást a felhasználás tekintetében. A Kubernetes a Docker host programot használja az egyes node-okon alkalmazások futtatásához. Alapvetően konténereket támogat, azonban nem csak a Docker-t, hanem pl. a Rocket-et és a Cryo-t is.

### 2.5.1. Architektúra

Egy Kubernetes cluster architektúrája fizikai vagy VM node-okból áll és minden node egy worker (2.2. ábra). Egy node meghibásodása esetén a rajta futó service-t átveszi a többi node. A node-ok között van egy kitüntetett master node, amelyik tárolja a cluster információkat és végzi a manager service-ek processzeit.

### 2.5.2. Komponensek

#### API server

A Kubernetes frontend-jét web UI-t és API-t szolgáltat. Ezzel kommunikál a felhasználó menedzser, CLI-t és a többi UI eszköz is.

#### etcd

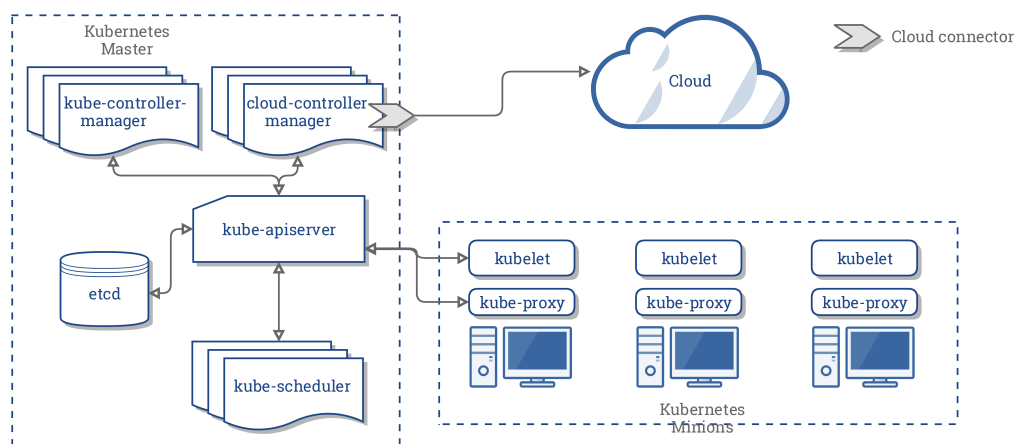
Elosztott kulcsérték tároló, a cluster menedzseléséhez szolgáló információknak.

#### Scheduler




Az elosztott működést valósítja meg a node-okon, minden új feladatot eloszt egy node-hoz.

#### Controller

Figyeli a végpontokat és beavatkozik ha valami tönkremegy.



2.2. ábra. Kubernetes architektúrája [8]

Konténer orkesztráció rendszerek			
			
Konténerre optimalizált	✓	✗	✓
Egyszerű telepítés	✓	✗	✗
Szolgáltatás skálázhatóság	✗	✗	✓
Horizontális skálázhatóság	✓	✓	✓
Vertikális skálázhatóság	✗	✓	✓
Multi-konténer deploy	✓	✗	✓
Minimum node	1	3 masters + slaves	master + 2 slaves

2.1. táblázat. Konténer orkesztrációs rendszerek összehasonlítása

### Container runtime

A konténerek keretszoftvere, leggyakoribb esetben a Docker.

### Kubelet

Ez a service minden node-on fut a cluster-en, a feladata, hogy a node-hoz kiszervezett konténerek fussanak az elvárt módon.

### Kube control

CLI, amin keresztül az adminisztrátor tud szolgáltatásokat indítani és cluster menedzsment feladatokat ellátni.

## 2.6. Keretrendszer meghatározása

A nagyszámú robotvezérlés feladatkörében hasonlítom össze a három legelterjedtebb orkesztrációs megoldást.

A vezérlési rendszerre legoptimálisabb választás a **Kubernetes**.

**Indoklás**

Egy QoS minőségbiztosított alkalmazás számára nagyon fontos a skálázhatóság és a perzisztencia. Ha kiesik egy node, akkor is fontos, hogy egy kis lassulással is, de stabil maradjon a szolgáltatás. Nagyon jól kezel multi-konténer alkalmazásokat, könnyű a szolgáltatás működése közben új konténereket indítani. A minőségbiztosítási feltételeket a Kubernetes biztosítja a legjobban.

## 3. fejezet

# Robotvezérlés környezete

### 3.1. Robotirányítás

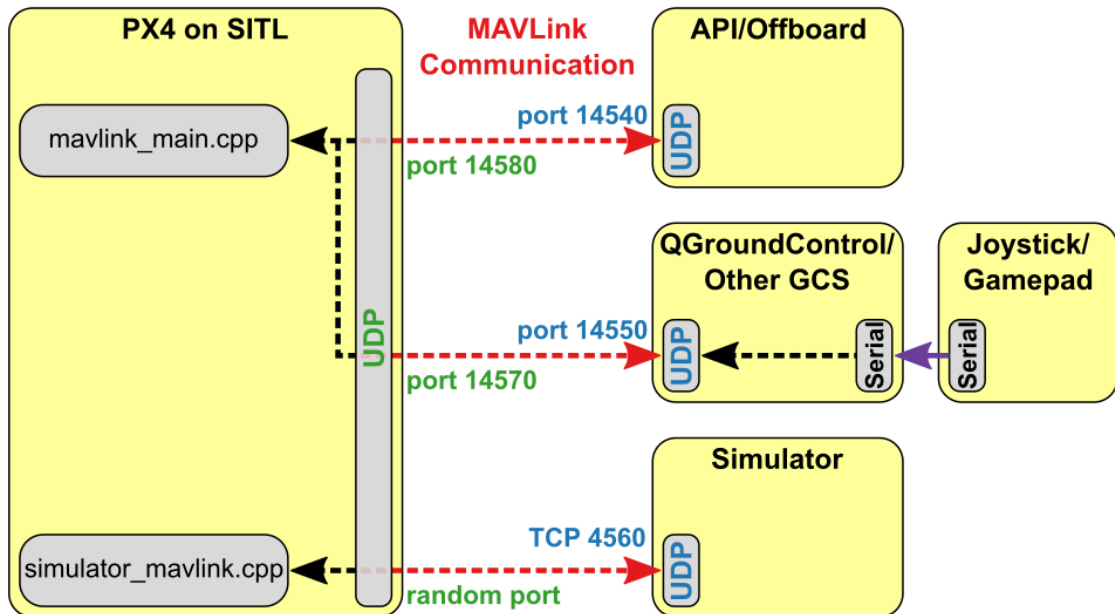
Robotirányításhoz szükséges, hogy legyen egy mechanikai rendszer, ez a robot azon részrendszere, amely az akciót valósítja meg. Az akció során szükség lehet a robot mozgatására a környezetben, ezt a helyváltoztató berendezés végzi. Motorok, különböző mechanikai tagok teszik lehetővé a helyváltoztatást. A szenzoros rendszer belső állapota maga a mechanikai rendszer állapota, míg a külső állapot a környezet állapotát jellemzi. Sokféle külső környezeti állapot létezik. Ahhoz, hogy a különböző környezeti tényezőket, például hőmérséklet, fényerősség, mágnesesség, láthatóvá és érzékelhetővé tegyük a robotunk számára, fel kell szerelni a megfelelő szenzorokkal. [2] A dolgozat során egy Mantis Q drónt (3.1. ábra) használunk robotként, a többit szimuláljuk.

### 3.2. Robot operációs rendszer - ROS

Bármilyen robot rendelkezik különféle eszközökkel amivel érzékelik a világot és mozognak benne. A Robot operációs rendszer egy nyílt forráskódú könyvtárakat és eszközöket kínál a szoftverfejlesztés segítségére robotot, mint hardvert irányító alkalmazások létrehozásához. Hardver absztrakciót, driver-eket, könyvtárakat, megjelenítőket, üzenetek továbbítását, csomagkezelést és más szolgáltatást is nyújt. [14] A ROS node-ok kommunikálni tudnak egymással, a szolgáltatások, hogy kérést küldjenek és választ kapjanak a node-ok között. A *rosservice* szolgáltatással kapcsolódhatunk a ROS szerveréhez. A *roslaunch* utasítás egy *.launch* kiterjesztésű fájl alapján indít egy robotot, a megadott paraméterekkel.



3.1. ábra. Yuneec Mantis Q [16]



3.2. ábra. PX4 kommunikációja Mavlink protokollal [12]

### 3.3. Kommunikáció - Mavros, Mavlink

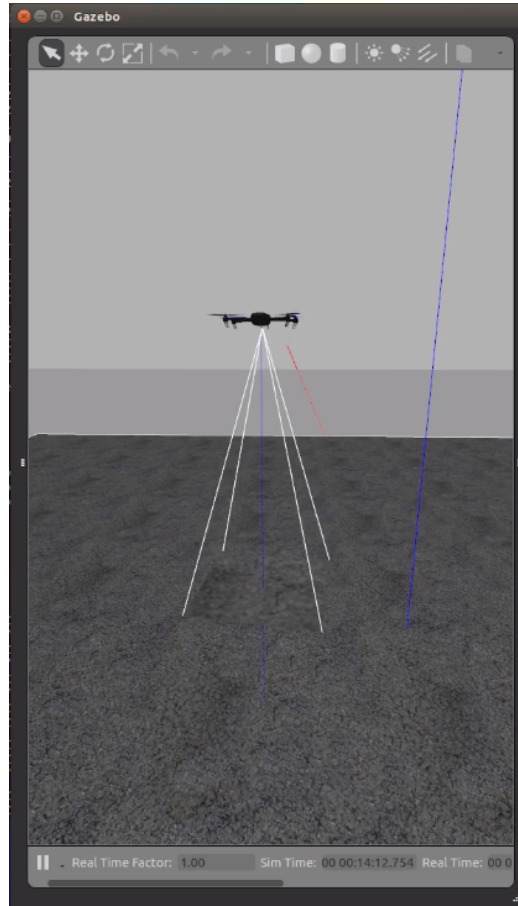
A MAVLink egy egyszerű üzenetküldési protokoll a drónokkal (és a fedélzeti drónkomponensek között) történő kommunikációhoz. A MAVLink hibrid publish-subscribe és point-to-point tervezési mintát követi. Az adatfolyamok témákként kerülnek elküldésre / közzétételre, míg a konfigurációs alprotokollok, például a missziók vagy a paraméterek point-to-point közötti újraküldéssel. Az üzeneteket az XML fájlok határozzák meg. Minden XML fájl meghatározza az adott MAVLink rendszer által támogatott üzenetkészletet. [10] A MAVLink nagyon hatékony, extrém kicsi az overhead, így QoS célra ideális.

A mavros egy ROS kiegészítő, amely megvalósítja a MAVLink kommunikációját a ROS-t futtató számítógépek, a MAVLink-kompatibilis autopilotok és a MAVLink-kompatibilis Ground Control Station (GCS) között.

### 3.4. Vezérlő - PX4

A PX4 tag körben elterjedt mint robot vezérlő eszköz, akár egyéni, akár ipari felhasználásra. A nyílt forráskódú PX4 használható drón, de akár tengeralattjáró vagy hajó vezérlésére is, sőt könnyedén testreszabható eszközöket lehet vele készíteni, illetve megosztani a közösségi platformjukon. [13] A ROS használható PX4-el és a Gazebo szimulátorral. A MAVROS MAVLink node-on keresztül használja a PX4-el való kommunikációhoz. A ROS és Gazebo integrált rendszer a 3.2 ábrán látható módon végzi a kommunikációt egy generikus PX4 szimulációs környezetben. A PX4 a szimulátorral (például Gazebo-val) kommunikál, ahonnan megkapja a szenzor adatot, esetünkben a kamera adatát a szimulált világból, illetve elküldi a motor és rotor vezérlési paramétereit. A PX4-el közvetlen fizikai eszközöket mozgatunk, ebben a környezetben a fejlesztő feladata, hogy megvalósítsa, hogy például egy méteres magasságba felszálláshoz a drónnak milyen eszközeivel mit kell csinálni. A PX4 továbbá kommunikál a GCS-el és a fedélzeti API-val, ami esetünkben a ROS, ahova parancsokat tud küldeni. [12]





**3.3. ábra.** Gazebo kamerás drón modell

## 3.5. Szimulációs környezet - Gazebo

Mivel a dolgozat egy nagy számú eszközkiszolgálást szeretne bemutatni és a tanszéknek egy Mantis Q drónja van, ezért a többinek a kiszolgálását szimulálni fogjuk, erre pedig kell egy szimulációs környezet, ami a Gazebo lesz. A Gazebo szimulátor lehetővé teszi az algoritmusok gyors tesztelését, a robotok tervezését, a regressziós tesztek elvégzését és akár AI rendszer kialakítását. Továbbá lehetőséget nyújt a robotok pontos és hatékony szimulálására komplex beltéri és kültéri környezetben. Kézhez kapunk egy 3D-s felületet is, amivel figyelni tudjuk a szimulációt, továbbá felhő alapú támogatást is biztosít, ami a témában még jól fog jönni. A Gazebo különböző modelleket támogat, mint például a SITL optical flow, ami pont egy Mantis Q jellegű kamerás drónt szimulál bejövő és feldolgozható optikai adatcsomaggal (3.3. ábra). Az ábrán látható drónból kijövő fénykúp által vetődő keret lesz a kamera képe. Sőt egy szimulált világba akármennyi modellt képesek vagyunk szimulálni a következőképpen.

```
cd src/Firmware
DONT_RUN=1 make px4_sitl_default gazebo
Tools/gazebo_multi_vehicle.sh -m sitl_optical_flow -n 10
```

**3.1. lista.** 10 optikai adatfolyamos drón szimulálása Gazebo-val

Az 3.1. számú listázásban futtatott script egyépként a szimulációs fájlrendszerben *xacro* fájlokat módosítva éri el, hogy létrejöjjenek a kívánt modellek. Ezen a scripten könnyen lehet módosítani saját tetszésünk szerint. A modellek elérésének az UDP portjai 14560-tól,

a TCP portjai 4560-tól inkrementálódnak, továbbá az összes a 14550 porton broadcast-el.  
[7]

A Gazebo felbontható szerverre és kliensre és a következő számításokat végzik:

#### Szerver

- Fizika kiszámítása
- Szenzorok szimulálása
- Engine API

#### Kliens

- Renderelés
- GUI

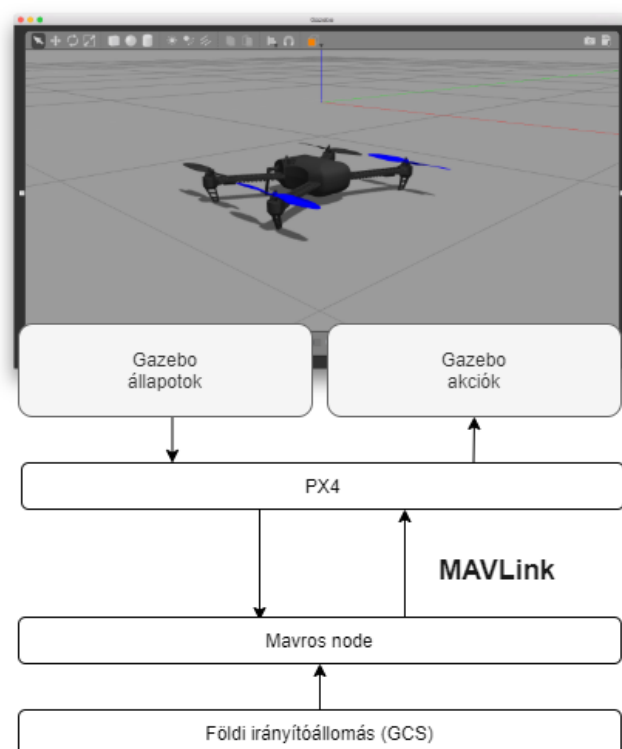
### 3.6. Együttes működés

A következő roslaunch utasítás egy lokális szimulációt indít a ROS-t összekötve MAVROS-al a 3.2. ábrán látható módon keresztül.

```
roslaunch mavros px4.launch fcu_url="udp://:14540@127.0.0.1:14557"
```

**3.2. lista.** Lokális PX4 szimuláció ROS-on keresztül Mavlink-el összekötve

A szimulált világban való videót a QGroundControl alkalmazással lehet megfigyelni, továbbá akármennyi modell manuális irányítását is ezzel az eszközzel teszteltem. Persze a későbbi tömeges szimulációhoz, majd valamilyen automatizmusra lesz szükség. Az együttes működéshez szükséges egyfajta proxy, a mavlinken keresztül való kommunikációhoz, ami a Mavros (3.4. ábra).



**3.4. ábra.** Az együttes működés kommunikációja Mavros node-on keresztül

## 4. fejezet

# Virtuális gépen kialakított tesztrendszer

A felhőrendszerbe való integrálás előtt két tesztet végzek Ubuntu VM-eken, amik a működés szempontjából fontosak lesznek.

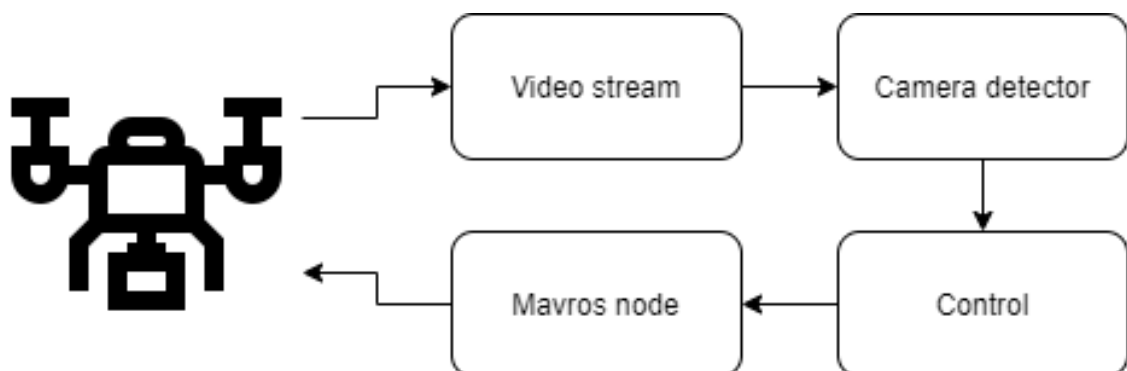
### 4.1. Azonos konténerrel egy drónvezérlés

Az első ilyen teszt egy a TMIT tanszék egyik előre készített konténerével lesz, ez a DockerHub-on megtalálható *bmehsnlab/aruco\_detect\_image\_v2* konténer. Ebbe a konténerbe bele van építve a videó stream, a kép feldolgozása, ami aruco kódokat detektál, a Mavros node működése és az irányító program. Ezen konténerek indítása előtt ki kell adnunk az *xhost* parancsot, mivel az X szerveren keresztül kommunikálnak egymással. Az X szerver kommunikációjához a konténereknek felcsatoljuk a */tmp/.X11-unix* fájlt.

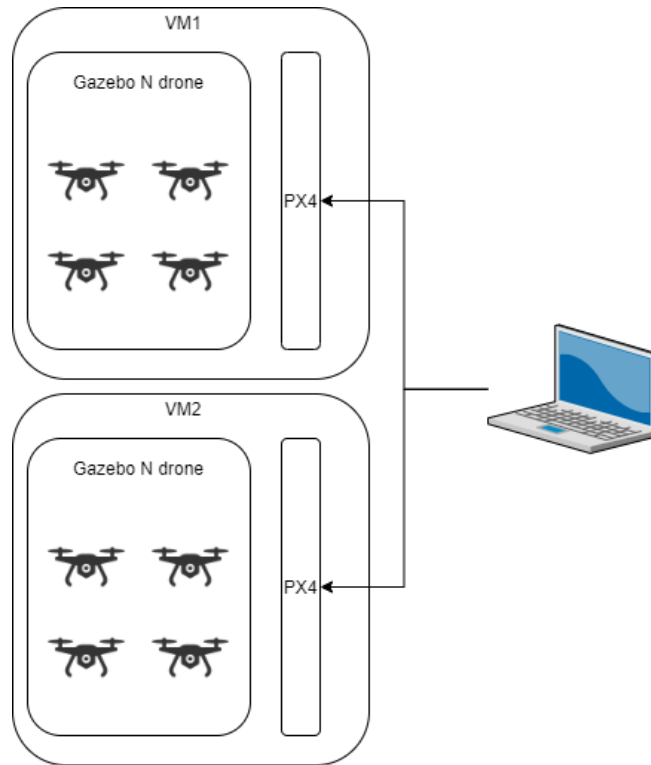
```
xhost +  
docker run --net=host -v /tmp/.X11-unix:/tmp/.X11-unix --name stream bmehsnlab/aruco_detect_image_v2  
docker run --net=host -v /tmp/.X11-unix:/tmp/.X11-unix --name dtctor bmehsnlab/aruco_detect_image_v2  
docker run --net=host -v /tmp/.X11-unix:/tmp/.X11-unix --name mavros bmehsnlab/aruco_detect_image_v2  
docker run --net=host -v /tmp/.X11-unix:/tmp/.X11-unix --name cntrol bmehsnlab/aruco_detect_image_v2
```

**4.1. lista.** Azonos konténerek indítása négy különböző feladattal és az X szerveren való kommunikációt megvalósítva

A teszt architektúrája a 4.1. képen látható. A teszt sikeresnek mondható, mivel a drónt sikerült irányításra bírni, illetve az optikai adatfolyamot fel tudta dolgozni az aruco kód-feldolgozó, habár aruco kódok nem voltak a szimulált világban.



4.1. ábra. Négy azonos konténerrel drónirányítás



**4.2. ábra.** Két VM-en több drón irányítása

## 4.2. Két VM-en több drón szimulációja és vezérlése

A következő teszten egy host OS-ből indított két VM-en teszteltem a több drón irányítását manuálisan. A teszthez telepítettem a VM-eken a fejezetben már felsorolt szoftvereket és környezeteket és a 3.1. listázásban bemutatott módon több drónt szimulációt is indítottam egy VM-en. Majd ezeket a földi irányítóállomás szimulátorával manuálisan vezéreltem. A teszt architektúrája a 4.2. ábrán látható.

## 5. fejezet

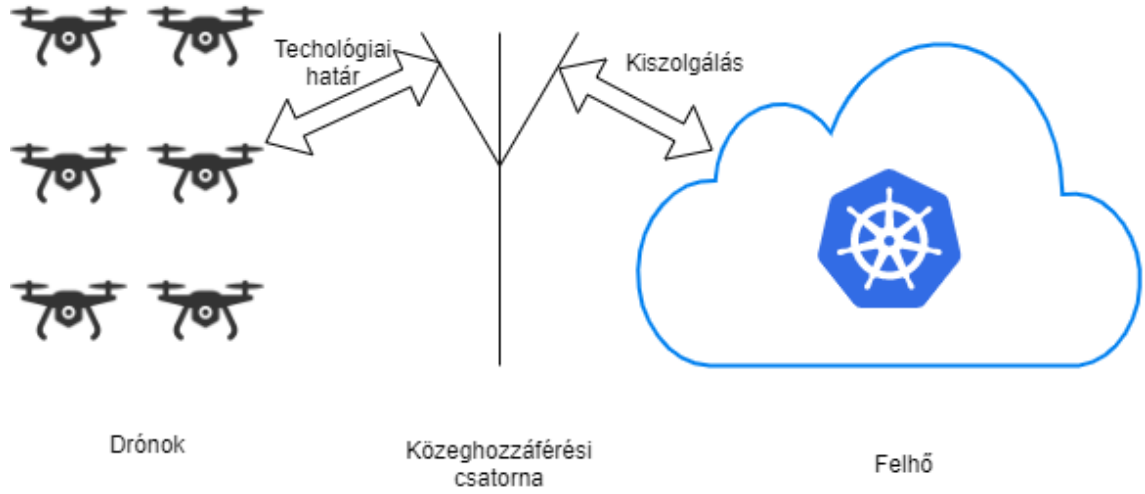
# QoS sztohasztikus becslése

A diplomaterv feladatai közé tartozik a Quality of Service (QoS), azaz a szolgáltatás minőségének a biztosításának megvalósítása, továbbá az 5G feltételrendszerével a feladat áttekintése. Egy durva modellben (5.1. ábrán látható), a robotok felhővel való kiszolgálásának késleltetését két helyen vizsgálhatjuk:

1. Van egy technológiai határ, amit a nagy eszközszám kommunikációja határoz meg, ez annak a határa, hogy hány eszköz tud stabilan kommunikálni egy hálózaton, például az egyetem hálózatán. Tudjuk, hogy a Bluetooth, IEEE 802.11 különböző változatai és az LTE is más-más eszközmennyiséget tud kiszolgálni egyszerre egy antennán keresztül. Ebben a fejezetben ennek a limitációnak a kibővítéséről nem fog szó esni, de későbbiekben megnézzük, hogy mit tehet ennek a kibővítésnek az érdekében az 5G technológia.
2. A másik a felhő kihasználtsága és feladatvégzési ideje. Itt a felhő feladatokat kap sorban valamilyen intenzitással és ezeknek a feladatoknak van egy elvégzési ideje, nyilván a Kubernetes cluster kapacitásának függvényében. Nyilvánvaló, hogy a felhő kihasználtsága valamilyen függőségben lesz a kiszolgálás idejével, ami a QoS-t meghatározza. Ebben a fejezetben azt a határt nézzük meg, hogy hogyan tudjuk méretezni a felhőnket, hogy valamilyen  $t_0$  felsőbecsléssel élhessünk egy drón kiszolgálásának idejére, természetesen miliszecundumban.

### 5.1. Tömegkiszolgálási modell

Valamennyi  $R$  drón valamilyen  $\lambda$  [kérés/s] intenzitással fordul a felhőhöz, hogy az számítást végezzen a kameraképen és megmondja mit kell csinálni. Erre a drón valamennyi idő múlva választ kap és ennek az időnek nem szabad elszállni. A megbecsüléséhez majd meg kell mérnünk, hogy átlagosan egy ilyen kérést a felhő mennyi idő alatt végez el, ez legyen  $\mu$ . Feltételezhetjük, hogy nagy  $R$  drónszám esetén függetlenül érkeznek a kérések, így modellezhetünk Poisson folyamattal a beérkező kéréseket nézve. Tehát ha egy drónnak a kérés intenzitása valamilyen feladatra  $\frac{\lambda}{R}$ , akkor az összes drónnak  $\lambda$ . Ezt tekinthetjük egy Markov folyamatnak, ahol az infinitezimális generátorból tudunk majd valamilyen sorhossz várhatóértékeket kiszámítani. Mivel nem vesznek el kérések, csupán torlódik a sor, ezért



**5.1. ábra.** Több robot kiszolgálásának felhőből a durva modellje

ez egy M/M/1 kiszolgálási modell. [9] Az infinitezimális generátor definíció szerint

$$G = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & 0 & \dots \\ \mu & -(\lambda + \mu) & \lambda & 0 & 0 & 0 & \dots \\ 0 & \mu & -(\lambda + \mu) & \lambda & 0 & 0 & \dots \\ 0 & 0 & \mu & -(\lambda + \mu) & \lambda & 0 & \dots \\ 0 & 0 & 0 & \mu & -(\lambda + \mu) & \lambda & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Ami annyit mond, hogy a várakozó eszközök sorhossza  $\lambda$  intenzitással növekedik egyet, amikor érkezik egy kérés és  $\mu$  intenzitással csökken egyet, amikor elvégződik egy kérés.

## 5.2. Várakozási idő várható értéke

Ha a tömegkiszolgálási modellünk jó és tényleg tekinthetjük a kérések eloszlását egy véletlennek, amit nem nagyon specifikus drónirányítás esetben megtehetünk, akkor egy M/M/1-es rendszer kiszolgálási késleltetését kell megnéznünk. A [9] jegyzet 4.4-es fejezet bebizonyítja, ennek a modellnek a késleltetését, ami a drón kérésére visszakapott válasz a felhő és az antenna között a  $D$  késleltetéssel arányos. A jegyzet bizonyítása alapján:

$$D = \frac{1}{\mu - \lambda}$$

## 6. fejezet

# Távoli Robotvezérlés Optimalizálása

Ebben a fejezetben a távoli robotvezérléshez használt algoritmust ismertetem. Az algoritmus célja, hogy a drónok vezérléséhez biztosítsa a QoS feltételeit, amit jelen esetben két paraméterben határozunk meg: sávszélesség és válaszidő. Ennek a két paraméternek a kontrollálásával biztosítható a videó folyam minősége a drón és a Kubernetes Node-ok között. A Kubernetes klaszter worker node-jai (rendelkezésre álló szerverek) erőforrás készletének függvényében a videófeldolgozás és a drón irányítása áthelyezhető (váltható, migrálható) az optimális teljesítmény érdekében. Az algoritmus rendszeresen méri az elérhető erőforrásokat, proaktívan vált a szolgáltatás zökkenőmentes fenntartása érdekében.

Tehát van valamennyi Node-unk amik között a drónok vezérlését áthelyezhetjük, az algoritmus úgy vált időközönként, hogy a késleltetést optimalizálja és biztosítja a sávszélességet is. Ha a sávszélességet nem lehet biztosítani, akkor egy ALARM állapotba fut, ekkor a drón biztonságosan leszáll és azonnal értesíti a felhasználót.

Az algoritmus bemenete tehát a Node-ok IP címének halmaza, amin keresztül kapcsolódik a drón a központhoz, a minimum sávszélesség a videó folyamra felé (vbw), a minimum sávszélesség a kontroll kommunikációra (cbw), ellenőrzések között eltelt időintervallum (T) és a jobb válaszidő esetén az az arány ami esetén vált (R). Az algoritmus a teljes üzemidő alatt fut T időközönként, a kimenete pedig a videó folyam vezérlése és a virtuális funkciók lokációjának a meghatározása, hálózati erőforrás vezérlése.

```
ALARM := false
while(!ALARM):
    migrationTrigger := false
    actLat = checkActualLat()
    (minLat, minNode) = MIN{checkAllNodeLat()}
    actBW = checkActualBW()
    moreBandwith = isThereNodeWithBetterBW()
    IF (minLat / actLat > R AND morBandWith):
        migrationTrigger = true
        nextNode = minNode
    criticalState = IF (cbw + vbw > actBW)
    IF (migrationTrigger OR criticalState):
        bestNode = findBestNode()
        bBW = checkActualBW(bestNode)
        IF (bBW < cbw + vbw):
            ALARM = true
        ELSE:
            swithTo(bestNode)
    sleep(T)
```

6.1. lista. Az optimalizáló algoritmus



Az algoritmusban használt migrationTrigger boolean változó-t igazra áll, hogyha a az algoritmus váltana egy jobb node-ra, az ALARM változó pedig igazra áll, ha baj van és le kell állítani biztonságosan a drónt. A minLat változóba tárolom a késleltetés minimumát, az actLat-ba pedig az aktuális node késleltetését. Hasonlóan az actBW az aktuális sávszélesség, a bestNode pedig az a node amelyikre vált az algoritmus.

# Összegzés

A tanulmányban megnéztük mire hasznának ma tömeges robotos irányítást, specifikusabban kitekintettünk, hogy mely iparágakban használhatóak a drónok, mire és hogyan használják. Megnéztük mire jó a konténerizáció, elmerültünk a konténer alapú felhőrendszerek világában, összehasonlítottuk a Docker Swarm-ot, a Mesos-t és a Kuberneteset. Átnéztük a robot- és drónirányítással kapcsolatos szoftvereket, hogy mik a lehetőségek, hogyan és minek szükséges jól együttműködni egy fizikai vagy egy szimulált drón irányításához. Megnéztük, hogyan lehet nagymennyiségű drónt szimulálni. Megnézhettünk két tesztet, konténerekkel való drónirányítás és jelfeldolgozás tesztjét és különböző VM-ekről drónszimulálás tesztet. Továbbá megnéztük, hogyan tudjuk megbecsülni nagyszámú drónkiszolgálásnak a késleltetését. Végül pedig javasoltunk egy algoritmust, amely a távoli drónvezérlés során szükséges videó folyamatok és vezérlési adatok együttes minőségbiztosítását végzi.

# Irodalomjegyzék

- [1] Aethon: Industry 4.0 components (2020. május 23.). <https://aethon.com/mobile-robots-and-industry4-0/>.
- [2] Juhász Ádám: Robotvezérlés telekommunikációs eszközökkel, 2008.
- [3] Docker docs: Overview of docker compose (2020. május 24.). <https://docs.docker.com/compose/>.
- [4] Frank Tobe: Lady gaga, 300 intel drones, and the super bowl (2020. május 23.). <https://www.therobotreport.com/lady-gaga-300-intel-drones-and-the-super-bowl/>.
- [5] Free Management Books: Project execution process (2020. május 23.). <http://www.free-management-ebooks.com/faqpm/processes-04.htm>.
- [6] Jacob Brogen: How intel lit up the super bowl with drones—and why (2020. május 23.). <https://slate.com/technology/2017/02/how-the-intel-drones-at-the-lady-gaga-super-bowl-halftime-show-worked.html>.
- [7] Jaeyoung Lim: Multi-vehicle drone simulation in gazebo (2020. május 27.). <https://auterion.com/multi-vehicle-drone-simulation-in-gazebo/>.
- [8] Kubernetes: Concepts underlying the cloud controller manager (2020. május 25.). <https://v1-17.docs.kubernetes.io/docs/concepts/architecture/cloud-controller/>.
- [9] Dr. Györfi László – Györi Sándor – Dr. Pintér Márta: *Tömegkiszolgálás*. 2002, Műegyetemi Kiadó.
- [10] Mavlink: Mavlink developer guide (2020. május 25.). <https://mavlink.io/en/>.
- [11] Mohamed Fawzy: Create cluster using docker swarm (2020. május 24.). <https://medium.com/tech-tajawal/create-cluster-using-docker-swarm-94d7b2a10c43>.
- [12] PX4: Ros with gazebo simulation (2020. május 26.). <https://dev.px4.io/v1.9.0/en/simulation/>.
- [13] PX4 developers: Open source autopilot (2020. május 26.). <https://px4.io/>.
- [14] ROS: Wiki (2020. május 25.). <http://wiki.ros.org/>.
- [15] Verband für unbemannte Luftfahrt: Amazon’s new delivery drone has ‘fail-safe logic (2020. május 23.). <https://www.uavdach.org/?p=1294813>.
- [16] Yuneec: Mantis q (2020. május 25.). [https://www.yuneec.com/en\\_GB/camera-drones/mantis-q/overview.html](https://www.yuneec.com/en_GB/camera-drones/mantis-q/overview.html).