Önálló laboratórium C-témacsoport

# Kamera alapú beltéri navigáció többbrotoros pilóta nélküli repülőgépekkel

Ekart Csaba [ZWPMKP]
Mérnökinformatikus Bsc

**Témavezető:**
dr. Zsedrovits Tamás

2020. május

# Tartalomjegyzék

# 1. Bevezetés

A modern technológia egyik legérdekesebb, és gazdasági szempontból is hatalmas potenciált jelentő újítása a széles körben elérhetővé vált pilóta nélküli repülőgépek, azaz drónok - idegen nevén Unmanned Aerial Vehicles (UAV) - témaköre. Ezek az eszközök az elmúlt években egyre nagyobb szerepet játszanak katonai és nemzetvédelmi felhasználási területeken, de egyéb gazdasági és üzleti alkalmazásuk is folyamatosan növekszik - így használhatóak csomagszállításra, környezeti körülmények elemzésére (pl. időjárás, légszennyezettség stb.), katasztrófavédelmi feladatok ellátására (pl. menekültkeresés). [1–3]

Ezen feladatok kielégítésére, és az új piaci igények teljesítésére a drónoknak jellemzően szüksége van olyan mesterséges intelligenciával támogatott szoftverekre, melyek lehetővé teszik az önálló navigálást, helyzetfelismerést, és egyéb specifikus feladatok elvégzését. [4] Ezekhez, a drónokon jellemzően megtalálható érzékelő szenzorok adatait használhatjuk fel, melyek lehetnek kamerák, légnyomásmérők, LIDAR szenzorok stb.

## 1.1. Feladatom

Az önálló laboratórium során végzett munkám keretében többrotoros pilóta nélküli repülőgépek beltéri navigációs rendszerének fejlesztéséhez szükséges alapvető feltételeket teremtettem meg, illetve - az egyetemi látogatás lehetetlenné válása miatt, és egyéb praktikai okokból - az ehhez szükséges szimulációs környezet összeállításával és megismerésével foglalkoztam.

A munkát két hallgatótársammal - Linzenbold Ádámmal és Benkó Barnabással - közösen, de különböző irányokba haladva végeztem.

# 2. A szimulációs környezet komponensei

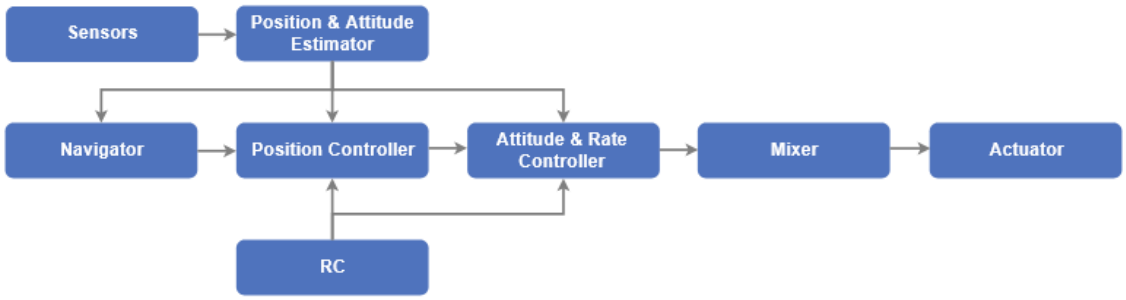
Mindenekelőtt röviden bemutatnám az általam felépített szimulációs rendszer legfontosabb komponenseit, illetve hogy ezek miként kommunikálnak egymással.

## 2.1. PX4

A PX4 egy nyílt forráskódú repülés irányító szoftver drónokhoz, melyet a Dronecode fejleszt. [5] Két fő komponense, a flight stack és a middleware. [6]

### 2.1.1. Flight stack

A flight stack navigációs és vezérlő algoritmusok gyűjteménye, amely többféle eszközön is használható, többek között többrotoros drónokon is. A hivatalos dokumentációban szereplő diagram jól szemlélteti a flight stack felépítését. [6]




**1. ábra.** Flight stack architektúrája [6]

Ennek legfontosabb elemeiről röviden:

- **Estimator:** Az estimator a beérkező szenzor adatok alapján ad számunkra becslést az eszköz állapotáról.
- **Controller:** A controller a navigator által meghatározott állapothoz próbálja eljuttatni az estimatortól kapott állapotot a megfelelő értékek változtatásával.
- **Mixer:** A mixer feladata, hogy a konkrét utasításokat lefordítsa a motor számára értelmezhető parancsokra.

### 2.1.2. Middleware

A middleware számos különböző feladatot ellát: ez tartalmazza a legfontosabb drivereket az eszközön található szenzorokhoz, valósítja meg a kommunikációt a külvilággal, az uORB üzenetek továbbításáért felel, illetve tartalmaz egy szimulációs réteget is, amely lehetővé teszi az eszközök szimulátoron keresztül történő irányítását, melyre szükségünk is lesz a továbbiakban. [6, 7]

#### 2.1.2.1 uORB üzenetek

Az uORB üzenetek a PX4 különböző moduljainak kommunikációját teszik lehetővé, felépítése a ROS topikokhoz hasonlóan subscribe / publish rendszeren alapul. Ezt a 2.2.2. fejezetben részletesebben is ismertetem. [8]

#### 2.1.2.2 Mavlink

A MAVlink egy egyszerű üzenet protokoll, melyet közvetlenül drónokhoz fejlesztettek, és lehetővé teszi, hogy kommunikáljunk a járművel egy távoli gépen futó szoftver segítségével (pl. ROS vagy QGroundControl). Saját üzeneteket is definiálhatunk. [9, 10]

Mivel a szimulációhoz mi ROS-t használtunk, ezért a MAVLink alapú kommunikáció megvalósításához a MAVROS nevű ROS package-et használtuk. [11]

## 2.2. Robot Operating System - ROS

A drónt a PX4 flight stacken keresztül irányíthatjuk, ehhez azonban olyan szoftver szükséges, amely teljes mértékben az eszközön fut. Amennyiben szeretnénk a drón irányítását távolról vezérelni, szükségünk van egy olyan programra, mely MAVLinken vagy RTPS-en keresztül, a flight stackkel kommunikálva képes erre. E feladat ellátásra alkalmas a Robot Operating System (továbbiakban ROS). [12]

A ROS egy nyílt forráskódú, szabadon használható framework robot programozáshoz, melyet eredetileg a Stanford AI kutatói indítottak útjára, manapság pedig az Open Source Robotics Foundation fejleszti. [13] Működésének rövid bemutatása keretében alapvető felépítését két fontos szempont szerint foglalom össze: a ROS fájlrendszer felépítése, illetve a ROS komputációs gráf működése szempontjából [14], majd röviden bemutatom a későbbiekben a feladatom során is használt URDF formátumot.

### 2.2.1. ROS fájlrendszer

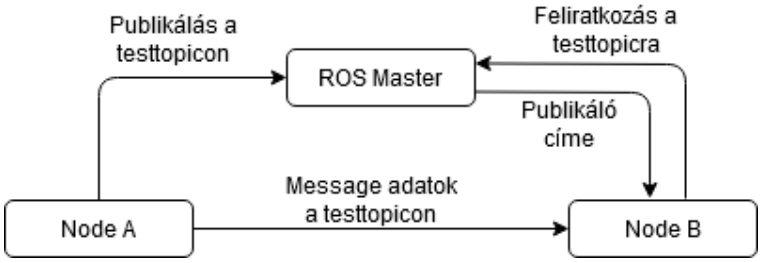
Az ROS rendszer alapvető építő elemei az ún. package-ek, azaz csomagok. Ezek tartalmazzák az összetartozó fájlokat: a ROS node-okat, a kapcsolódó adathalmazokat, konfigurációs fájlokat, stb. Az összefüggő package-eket metapackage-ekbe rendezhetjük. [14] A packagek rendelkeznek egy ún. manifest XML fájllal (`package.xml`), mely tartalmazza a csomag nevét, verzióját, és egyéb információkat.

### 2.2.2. ROS komputációs gráf

A ROS komputációs gráf egy peer to peer hálózat, melyben a ROS folyamatai kommunikálhatnak. Fejlesztés közben arra kell törekedni, hogy a kódot egymástól függetlenül használható, speciális feladatok (pl. motor irányítása) ellátására szakosodott modulokra bontsuk, melyeket node-nak neveznek.

A különböző node-ok alapvető kommunikációja úgynevezett topicokon keresztül történik. Egy topic tulajdonképpen egy csatorna, melyen a node-ok speciális struktúrába rendezett adatokat - messageket küldhetnek egymásnak. Ezeket az üzeneteket a megfelelő pacakgeben, `.msg` formátumban tároljuk. A topicokra a különböző node-ok feliratkozhatnak, illetve adatokat publikálhatnak rajtuk. Az a node, amelyik egy topicra fel van iratkozva megkapja az összes üzenetet, amit mások publikálnak rajta. [14, 15]

Ennek a rendszernek fontos eleme a ROS Master, melynek feladata a nodeok közötti peer to peer kommunikáció megteremtése, illetve ezen folyamatok nyilvántartása. A ROS Master címével minden node rendelkezik. Azok a node-ok amelyek információt szeretnének megosztani egy topicon, jeleznek a masternek, hogy melyik topicon szeretnének üzeneteket küldeni, és mi az ő címük. Ezt követően, ha egy node feliratkozik erre a topicra a master elküldi neki, hogy kik azok, akik ezen a topicon publikálnak. Ezt követően közvetlen kapcsolat épül fel az adó és a vevő pontok között. [14–16]




**2. ábra.** A ROS Master szerepe a node-ok topic alapú kommunikációjában. Node A jelzi a ROS Masternek, hogy publikálni szeretne a "testtopic" nevű topicon, aki eltárolja ezt az információt. Ezután Node B szeretne feliratkozni ugyanerre a topicra. A master elküldi neki a címet, és a két végpont között felépül a peer to peer kapcsolat. [15,16]

Habár ez a rendszer rendkívül hatékony, de sajnos nem alkalmas a hagyományos kérés-válasz alapú interakciók megvalósítására. Ezekben az esetekben célszerű a ROS Service-eket használni. A servicet egy kérésből és egy válaszból álló üzenet pár definiál. A service-eket a messagekhez hasonlóan fájlban tároljuk az `.srv` kiterjesztésű fájlokban. [14]

### 2.2.3. URDF

Az URDF (Unified Robot Description Format) egy a ROS által használt speciális XML formátum, melyet robot modellek egyszerű leírására hoztak létre. [17] Leírhatjuk benne az eszköz nevét, a vizuális megjelenését felépítő komponenseket, fizikai tulajdonságait, stb. Az URDF modelleket a Gazebo be tudja olvasni, így a szimulátorban egyszerűen használhatjuk őket. Ezeket az XML fájlokat egy speciális XML makró nyelvben - xacroban írhatjuk. [18]

## 2.3. Gazebo

A PX4 számos különböző szimulátorral használható - ilyen a Gazebo, az AirSim, vagy a jMAVSim. Ezek különböző képességekkel, illetve erőforrás igényekkel rendelkeznek, ezeket mind figyelembe is vettük a megfelelő kiválasztásánál. Az Unreal Engine alapjaira épülő Airsim a legmodernebb és leglátványosabb program, cserébe lényegesen nagyobb erőforrás igénnyel rendelkezik, ezért jóval erősebb hardware szükséges a futtatásához. A jMAVSim pedig hiába egy könnyű és rendkívül gyors program, sajnos nem alkalmas komolyabb robotikai feladatok ellátására. Végül a PX4 fejlesztői által javasolt Gazebot választottam a szimulációs környezetem megvalósításához. [19]

A Gazebo egy jól támogatott, egyszerűen használható, dinamikus 3D szimulátor, ami magas

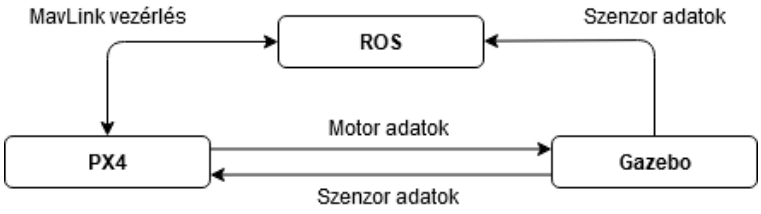
szintű fizikai szimulációt tesz lehetővé, számos szenzor imitálására alkalmas, és beltéri, illetve kültéri környezethez egyaránt felhasználható.

Legfontosabb előnye számomra, hogy könnyen használható ROS-sal, és kifejezetten alkalmas gépi látásra épülő alkalmazások fejlesztéséhez. [20]

## 2.4. Ezen komponensek kommunikációja a szimulációban

Az eddigiekben ismertetett komponensek közösen alkotják az általam használt szimulációs környezetet. A következőkben röviden bemutatom, hogyan kommunikálnak ezek egymással.

A PX4 flight stack valósítja meg magának a drónnak a mozgatását. Mivel a drón vezérlését nem közvetlenül a PX4 flight stackjében, hanem a saját eszközömön akartam megvalósítani, ehhez ROS-t használtam, amely MAVLink üzenetek segítségével kommunikál az eszközzel. A PX4 firmware a végrehajtandó motoros adatokat elküldi a Gazebo szimulátornak, az pedig visszafele a szenzoradatokat közvetíti. A szimulátortól kapott adatokat alapjáraton a PX4 továbbítja a ROS felé a többi állapot információval (pl. akkumulátor szint) egyetemben, de pluginok segítségével közvetlenül is lehetőség van az érzékelők adatait eljuttatni a ROS-nak. [19,21,22]




**3. ábra.** A ROS, PX4, Gazebo szimulációs architektúra [19,22]

Ezzel az architektúrával lehetővé válik, hogy a ROS fejlesztői környezetében, és eszközeit felhasználva az éppen beérkező szenzor adatoktól függően hajtsunk végre utasításokat, melyeket MAVLinken keresztül továbbíthatunk magának a járműnek, így végrehajtva azokat.

# 3. Szimulációs környezet összeállítása és használata

Ebben a fejezetben a szimulációs környezet összeállításához szükséges lépéseket ismertetem, illetve röviden bemutatom ennek egyszerű használatát és hogy pontosan milyen irányban indultam el, valamint merre lehet folytatni majd a témát.

## 3.1. A környezet összeállítása

Először a PX4 Firmware legfrissebb verzióját kellett letölteni a hivatalos repositoryból. [23]

```
$ git clone https://github.com/PX4/Firmware.git --recursive
```

Ezt követően futtattam a fejlesztők által előre elkészített scriptet, mely a szükséges dependenciákat, valamint a jMAVSim és a Gazebo szimulátorokat telepíti. [23]

```
$ bash ./Tools/setup/ubuntu.sh
```

Ezután, a ROS legfrissebb stabil kiadását, a Melodicot telepítjük a PX4 repositoryjában található installáló scripttel. [23]

```
$ wget https://raw.githubusercontent.com/PX4/Devguide/master/build_scripts/
  ubuntu_sim_ros_melodic.sh
$ bash ubuntu_sim_ros_melodic.sh
```

A scriptek elméletileg minden szükséges komponenst telepítenek, de előfordulhat, hogy az alábbiakat külön kell futtatnunk.

```
$ pip install -U future
$ sudo apt install ros-melodic-geographics-msgs libgeographic-dev geographiclib-
  tools
$ sudo geographiclib-get-geoids egm96-5
$ sudo geographiclib-get-gravity egm96
$ sudo geographiclib-get-magnetic emm2015
```

Ezt követően a szimuláció már akadálytalanul futtatható.

## 3.2. A szimuláció indítása

Az alábbi paranccsal elindítjuk a szimulációt és létrehozzuk a kapcsolatot a ROS-sal a MAVROS-on keresztül. [21]

```
$ roslaunch mavros px4.launch fcu_url:="udp://:14540@192.168.12.87:14557"
```

A megadott IP cím a szimulációt futtató számítógépre hivatkozik (a szimuláció futtatható külön eszközön is). Az egyszerűség kedvéért a szimulációt minden alkalommal localhoston futtattam, ilyenkor az IP címet lecserélhetjük a fenti parancsban `127.0.0.1`-re.

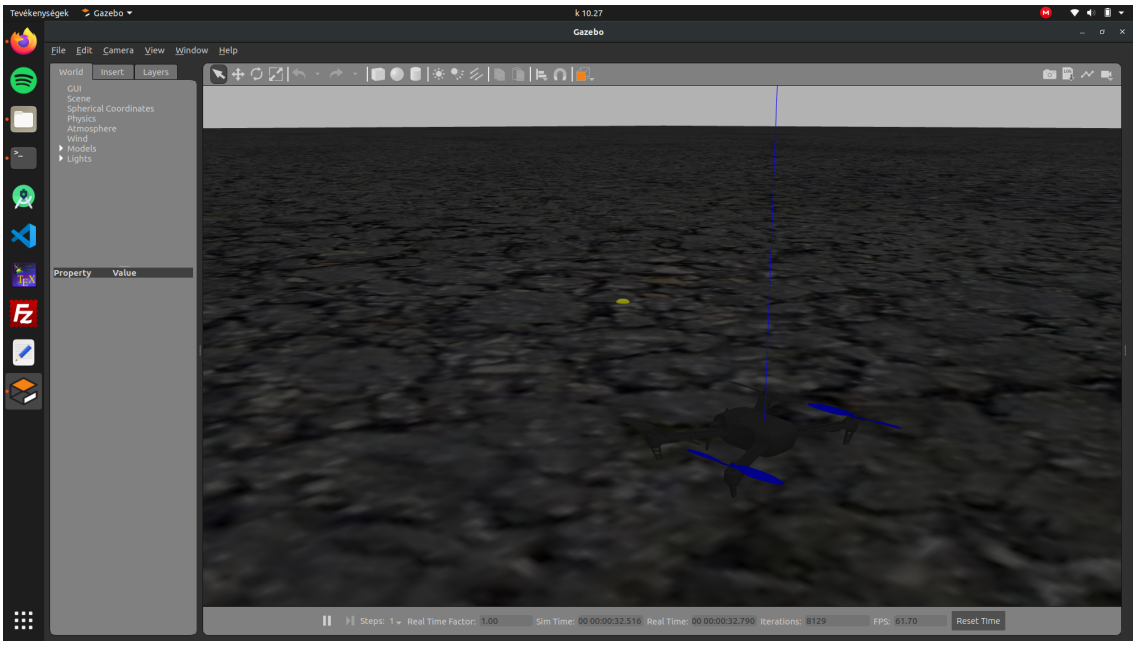
Ezt követően egy másik terminálból elindíthatjuk a Gazebo szimulátort amelyhez először a leklónozott Firmware mappájába kell navigálnunk. Ahhoz hogy maga a szimulátor lépes legyen a szenzor adatokat közvetlenül a ROS felé kommunikálni a megfelelő topicocon, az ezekhez kapcsolódó pluginokra van szükségünk. Ehhez az alábbiak szerint kell indítanunk a szimulátort: [21]

```
DONT_RUN=1 make px4_sitl_default gazebo
source ~/catkin_ws/devel/setup.bash    # (optional)
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
roslaunch px4 posix_sitl.launch
```

Az első terminálban megtett lépéseket az egyszerűség kedvéért kihagyhatjuk, és magát a MAVROS-t is indíthatjuk ugyanabból a terminálból, csak az utolsó sorban lévő parancsot cseréljük le az alábbi szerint.

```
roslaunch px4 mavros_posix_sitl.launch
```

A szimuláció sikeres indítása után a Gazeboban meg is jelenik az Iris kvadkopter.




**4. ábra.** A Gazebo környezet indítás után, benne az Iris kvadkopterrel

## 3.3. A drón mozgatása

A szimuláció sikeres indítása után az első feladatom a drón megmozdítása volt. A drón felemeléséhez a szimulációt futtató terminálba beírtam az alábbi parancsot. [20]

```
pxh> commander takeoff
```

A drón mozgatását először a QGroundControl nevű program segítségével valósítottam meg, amely egy egyszerűen használható multiplatform földi vezérlő állomás (Ground Control Station - GCS), mely segítségével - a ROS-hoz hasonlóan - MAVLinken keresztül küldhetünk közvetlen parancsokat a PX4 felé. [24,25] A számítógéphez virtuális, vagy valódi kontrollert csatlakoztatva, közvetlen parancsokat adhattunk a drónnak, illetve repülés terveket hajtottunk végre vele. [19]

A cél azonban az volt, hogy olyan programot hozzunk létre, amellyel esélyünk van a drón távoli vezérlésére, illetve a kamera adatok lekérésére. A ROS két fejlesztői könyvtárat is rendelkezésünkre bocsát ehhez: a rospy-t, amely Pythonnal használható, illetve a C++-hoz készült rocpp-t. A későbbiekben tervezett gépi tanulásos irány, illetve a látszólag bővebb közösség miatt a rospy irányában indultunk el.

A rospy client API lehetővé teszi számunkra, hogy könnyen kezelhessük a korábban említett ROS topicokat, serviceket, illetve paramétereket, lehetőséget teremtve az egyszerű kommunikációhoz a PX4-gyel. A hivatalos dokumentációt, illetve egy a GitHub-on talált rövid példaprogram [26] segítségével megvalósítottuk a drón irányítására alkalmas scriptet. Ez egy egyszerű Service alapú kommunikációt valósít meg a PX4-gyel a MAVROS csomag segítségével.

## 3.4. Kamera szenzor adatainak lekérése

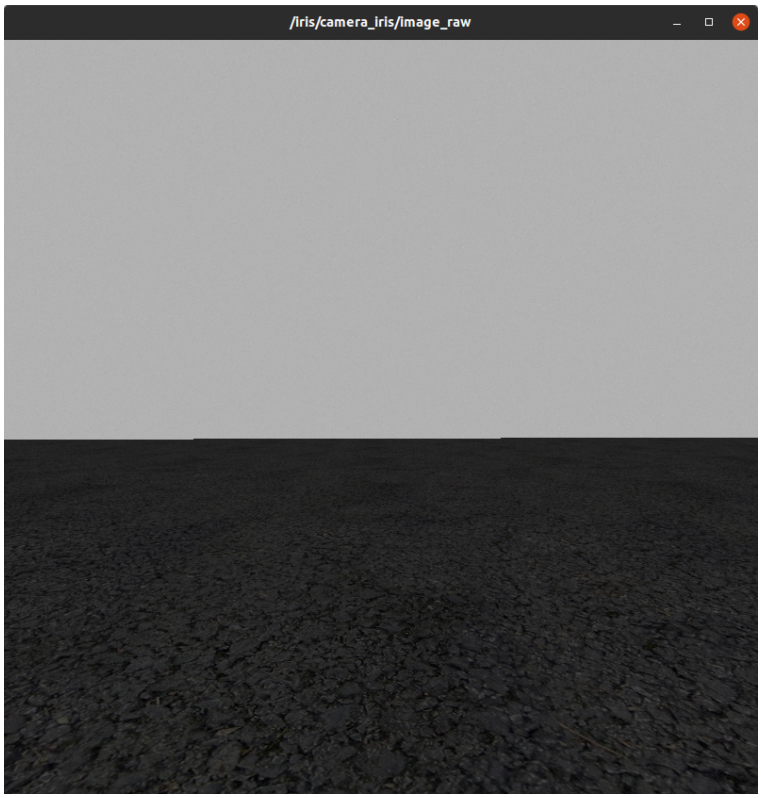
Ezt követően azt vizsgáltam, hogy milyen lehetőségeink vannak a kamera adatok lekérésére, mivel a későbbiekben a szakdolgozatom keretében, ezen adatok vizsgálata alapján tervezek objektum felismerésre alkalmas programot létrehozni.

Mivel az Irisen alapbeállítások szerint nem szerepel kamera modul, a ROS fórumokon talált információk szerint ki egészítettem a modellhez tartozó `iris_base.xacro` URDF fájlt, melyben definiáljuk a kamera dőlés szögét, az FPS számot, a kép méretet, és további értékeket. [27]

Ezt követően újrabuildeltem a PX4-et, hogy tartalmazza a változtatásokat, és elindítottam a szimulációt. A drónon meg is jelent a hozzáadott kamera modul. A `rostopic list` parancs segítségével kilistáztam az aktív topicokat, és láthatóvá is vált, hogy a drón kamerájának képe az `/iris/camera_iris/image_raw` kerül megosztásra. Ezen adatok megjelenítésére az image_view

nevű ros packaget használtam, mellyel az alábbi egysoros kóddal sikerült is megjeleníteni a kamera élő képét. [28]

```
$ rosrun image_view image_view image:=/iris/camera_irris/image_raw
```




**5. ábra.** A kamera által érzékelt kép megjelenítése a ROS image_view segítségével

# 4. Összefoglalás

A félév során megismerkedtünk a pilóta nélküli repülőgépek fejlesztésének különböző lehetőségeivel az általunk épített szimulációs környezetben. Sikerült megvalósítanunk egy egyszerű programot, mellyel meg tudjuk valósítani a drón mozgatását, illetve le tudtuk kérni és megjeleníteni a kamera szenzor adatait. Az általunk felépített rendszer alapos megismerése, és használata után a témát tovább tervezzük folytatni a jövőre készülő szakdolgozatunk keretében.

A téma robusztus mivoltából fakadóan több irányba tervezünk elindulni hallgatótársaimmal. Drónok beltéri alkalmazása esetén rendszerint nincs esélyünk külső referenciákat alkalmazó pozíció meghatározó rendszerek használatára, ezért rendkívül fontos a drón szenzorai segítségével a környezet megfelelő felmérése. A továbbiakban az eszköz kamerájának képe alapján, gépi látás segítségével szeretnék megvalósítani olyan programot, mely alkalmas a különböző objektumok felismerésére, így lehetővé téve a megfelelő navigációs döntések (például kitérést, vagy épp közelítést) meghozatalát.

A munka nagy része természetesen még hátra van, de bízom benne, hogy tovább haladva az elkezdett úton sikerülhet olyan programot fejlesztenünk mely alkalmas lehet drónok beltéri navigációjának levezénylésére, és különböző akadálypályák sikeres teljesítésére.

# Hivatkozások


[1] *Types of Drones – Explore the Different Models of UAV's.* http://www.circuitstoday.com/types-of-drones

[2] Francesco Castellano *Commercial Drones Are Revolutionizing Business Operations.* https://www.toptal.com/finance/market-research-analysts/drone-market

[3] Goldman Sachs https://www.goldmansachs.com/insights/technology-driving-innovation/drones/

[4] Esat Dedezade *Jobs of the future: how self-piloted AI drones are creating exciting new opportunities* (2019.09.12). https://news.microsoft.com/europe/features/jobs-of-the-future-how-self-piloted-ai-drones-are-creating-exciting-new-opportunities/

[5] PX4 Development Guide, PX4 - Software Overview https://px4.io/software/software-overview/

[6] PX4 Development Guide, PX4 Architectural Overview https://dev.px4.io/master/en/concept/architecture.html

[7] PX4 Development Guide, Middleware https://dev.px4.io/master/en/middleware/

[8] PX4 Development Guide, uORB Messaging https://dev.px4.io/master/en/middleware/uorb.html

[9] PX4 Development Guide, MAVLink Messaging https://dev.px4.io/master/en/middleware/mavlink.html

[10] MAVLink Developer Guide https://mavlink.io/en/

[11] PX4 Development Guide, MAVROS https://dev.px4.io/v1.10/en/ros/mavros_installation.html

[12] PX4 Development Guide, Robotics using ROS https://dev.px4.io/master/en/ros/

[13] ROS wiki, About ROS https://www.ros.org/about-ros/

[14] ROS wiki, Concepts http://wiki.ros.org/ROS/Concepts

[15] Justin Huang *ROS tutorial #2: Publishers and subscribers* (2016.03.21). https://www.youtube.com/watch?v=bJB9tv4ThV4

[16] ROS wiki, Master http://wiki.ros.org/Master

[17] ROS wiki, URDF http://wiki.ros.org/urdf

[18] ROS wiki, xacro http://wiki.ros.org/xacro

[19] PX4 Development Guide, Simulation https://dev.px4.io/master/en/simulation/

[20] PX4 Development Guide, Gazebo Simulation https://dev.px4.io/master/en/simulation/gazebo.html

[21] PX4 Development Guide, ROS with Gazebo Simulation https://dev.px4.io/master/en/simulation/ros_interface.html

[22] CE-Bench *Basic Overview of ROS / Gazebo / PX4 Simulation Architecture* (2018.02.11). https://www.youtube.com/watch?v=AmhbRmqjYXY

[23] PX4 Development Guide, Development Environment on Ubuntu LTS / Debian Linux https://dev.px4.io/master/en/setup/dev_env_linux_ubuntu.html

[24] QGroundControl http://qgroundcontrol.com/

[25] PX4 Development Guide, QGroundControl https://dev.px4.io/master/en/qgc/

[26] https://github.com/aniskoubaa/ros_dronemap

[27] https://discuss.px4.io/t/how-to-add-a-ros-camera-to-iris-for-gazebo-simulation/5118/2

[28] ROS wiki, image_view http://wiki.ros.org/image_view