Technical Journal

Pendidikan Robotics Software Control
Aksantara ITB 2026

Nama: [**Muhammad Rafiandhi Suryadinata / Reyy**]
NIM: [**19625129**]
Divisi: **Robotics Software Control**

February 8, 2026

Contents

1 Catatan Harian

1.1 DAY 1 - Kamis, 29 Januari 2025

Hari pertama Pendidikan RSC Aksantara 2026 menjadi fondasi awal untuk memahami arah, budaya, serta ruang lingkup teknis yang akan dijalani sebagai Ca-RSC. Materi yang disampaikan tidak hanya bersifat pengenalan, tetapi juga membangun *big picture* mengenai bagaimana peran software bekerja dalam sistem UAV secara menyeluruh.

Pengenalan Departemen Robotics Software Control (RSC)

Pada sesi awal, kami diperkenalkan dengan Departemen Robotics Software Control (RSC), departemen yang didirikan pada tahun 2020 dan bertanggung jawab atas seluruh aspek **perangkat lunak sistem UAV**. Cakupan kerja RSC meliputi pengembangan **algoritma kendali otonom**, sistem persepsi berbasis **computer vision**, hingga perancangan antarmuka **Ground Control Station (GCS)**.

Saat ini, RSC dipimpin oleh **Kak Z. Nayaka Athadiansyah (IF'23)**. Struktur internal RSC terdiri dari tiga divisi utama yaitu **Divisi Manajemen Pengetahuan**, **Divisi Penelitian dan Pengembangan**, dan **Divisi Manajemen Sumber Daya Manusia**. Selain itu, anggota RSC terbagi ke dalam dua jurusan besar yaitu: **Control and Perception** dan **Ground Control Station** yang kemudian dikelompokkan ke dalam tim UAV seperti **VTOL**, **TD**, **FW**, dan **LELA**.

Peran Divisi

- **Divisi Manajemen Pengetahuan**: bertanggung jawab atas penyusunan materi dan sistem penilaian selama pendidikan, serta mengelola **RSCWiki** sebagai repositori dokumentasi teknis.
- **Divisi Penelitian dan Pengembangan**: membantu proses pendidikan dan mengelola riset serta repositori GitHub.
- **Divisi Manajemen Sumber Daya Manusia**: menangani presensi, technical journal, perizinan, serta sistem internal seperti **RSCBoarding** dan **progress report**.

Jurusan dan Fokus Teknis

- **Control and Perception** berfokus pada pengembangan sistem navigasi otonom berbasis **ROS2** dan sistem persepsi menggunakan **computer vision**.
- **Ground Control Station** berfokus pada pengembangan aplikasi GCS, mulai dari **UI/UX**, aplikasi berbasis web, hingga komunikasi menggunakan **MAVLink**.

Gambaran Tim UAV

TD berfokus pada pengembangan GCS dan computer vision, **VTOL** pada sistem otonom dan presisi misi, **LELA** pada deteksi titik api, serta **FW** pada dukungan pemetaan berbasis **GIS**.

Tata Tertib Budaya komunikasi yang santai namun tetap sopan sangat ditekankan, dengan larangan penggunaan frasa militeristik.

Arsitektur Sistem UAV

Pada materi ini, aku mempelajari arsitektur sistem UAV secara menyeluruh, mulai dari komponen fisik hingga logika operasi dan mekanisme keselamatan. Materi ini membantu membangun pemahaman bahwa UAV bukan sekadar alat terbang, melainkan sebuah sistem kompleks yang terdiri dari banyak subsistem yang saling terintegrasi.

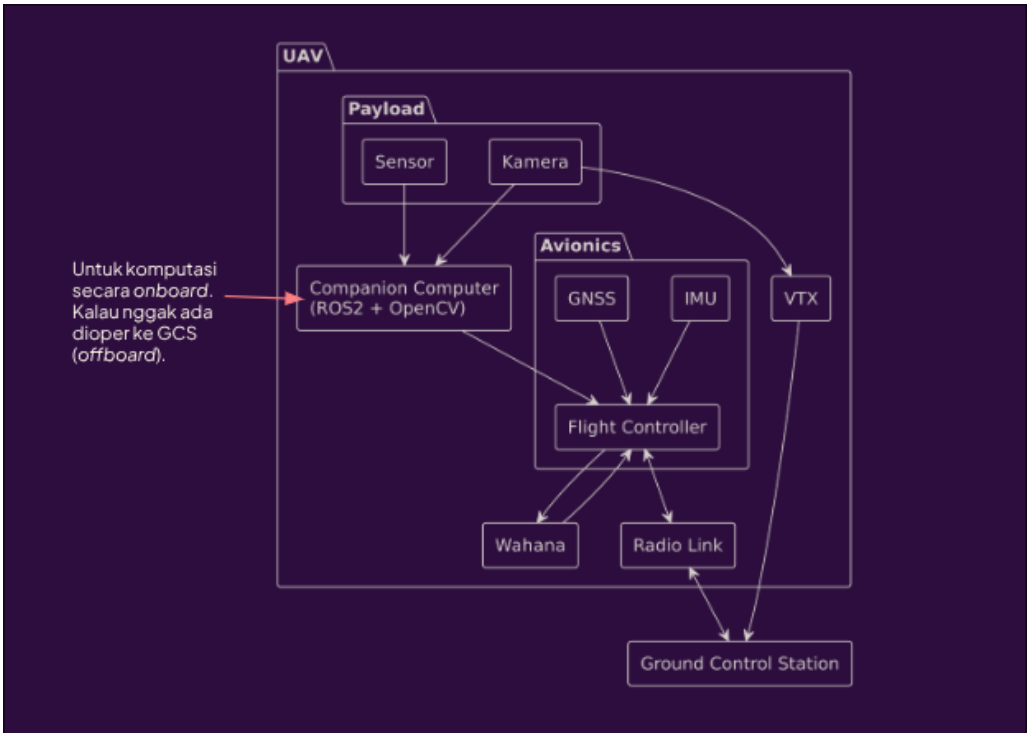

Figure 1: Diagram arsitektur sistem UAV

Gambaran Umum Sistem Secara umum, sistem UAV terdiri dari wahana, avionics, payload, sistem komunikasi, dan companion computer. Seluruh komponen ini bekerja bersama untuk memungkinkan UAV terbang, bernavigasi, berkomunikasi, serta menjalankan misi secara manual maupun otonom. Diagram arsitektur UAV digunakan untuk menunjukkan bagaimana data dan perintah mengalir antar komponen.

Wahana dan Sistem Propulsi

Wahana merupakan struktur fisik utama UAV. Untuk UAV jenis HTOL dan hybrid, wahana terdiri dari fuselage, sayap, dan ekor. Sementara itu, UAV VTOL menggunakan frame atau center plate dengan arm sebagai penopang motor. Sistem propulsi terdiri dari propeller, motor, Electronic Speed Controller (ESC), dan baterai Lithium Polymer. Propeller menghasilkan gaya angkat atau dorong melalui rotasi, motor mengubah energi listrik menjadi energi mekanik, dan ESC mengatur kecepatan serta arah putaran motor berdasarkan perintah dari Flight Controller. Saat ini, ESC yang digunakan merupakan ESC Alcentara, yaitu ESC buatan internal Aksantara.

Avionics dan Flight Controller

Avionics berfungsi sebagai sistem kendali utama UAV, dengan Flight Controller (FC) sebagai pusat pemrosesan. FC dapat dianalogikan sebagai sistem saraf motorik UAV

karena bertugas memproses data sensor dan menjalankan algoritma kontrol penerbangan, baik secara manual maupun otonom. FC yang digunakan antara lain Antares (FC buatan Aksantara) dan Matek F722-HD. Di dalamnya terdapat Microcontroller Unit (MCU) sebagai prosesor utama, barometer untuk estimasi ketinggian, serta Inertial Measurement Unit (IMU) yang terdiri dari accelerometer dan gyroscope untuk menentukan orientasi dan pergerakan UAV. Selain itu, terdapat blackbox yang menyimpan log penerbangan untuk keperluan analisis pasca-terbang. Sistem navigasi dilengkapi dengan modul GNSS yang menerima sinyal dari GPS, GLONASS, Galileo, dan BeiDou. Magnetometer yang biasanya terintegrasi dengan modul GNSS digunakan untuk menentukan arah mata angin, sedangkan airspeed sensor berfungsi mengukur kecepatan relatif UAV terhadap udara menggunakan pitot tube.

Payload

Payload merupakan komponen tambahan yang disesuaikan dengan kebutuhan misi. Contoh payload yang digunakan antara lain kamera FPV untuk video real-time, kamera depth untuk pemetaan kedalaman dan titik 3D, kamera tracking untuk estimasi posisi tanpa GPS, LiDAR untuk pemetaan presisi tinggi, serta sensor partikel untuk pemantauan kualitas udara.

Sistem Komunikasi Radio

Sistem Komunikasi Radio memungkinkan interaksi antara UAV dan Ground Control Station (GCS). Komunikasi ini membentuk *link*, yaitu jalur pertukaran data melalui frekuensi radio tertentu. Dalam sistem UAV, terdapat dua link utama. Link pertama adalah *Command and Control (C2) Link*, yang bertanggung jawab atas pengendalian UAV. Link ini terdiri dari uplink untuk mengirim perintah dari GCS ke UAV dan downlink untuk mengirim data telemetri seperti attitude, posisi, dan status baterai ke GCS. Link kedua adalah *Payload Link*, yang digunakan khusus untuk mengirim data payload seperti video dari VTX ke VRX secara real-time.

Companion Computer (CC)

Companion Computer (CC) berperan sebagai unit komputasi tingkat tinggi yang melengkapi Flight Controller. CC digunakan ketika misi membutuhkan pemrosesan lanjutan seperti perencanaan jalur, penghindaran rintangan, dan pemrosesan citra berbasis AI secara langsung di wahana. CC terhubung ke FC melalui komunikasi serial dan berkomunikasi menggunakan protokol MAVLink. Sensor dan payload mengirimkan data ke CC untuk diolah, kemudian CC mengirimkan perintah hasil komputasi ke FC untuk dieksekusi. Contoh perangkat CC yang digunakan adalah NVIDIA Jetson Nano dan Raspberry Pi.

Metode Komunikasi dan Protokol

Pengiriman data dalam sistem UAV dapat dilakukan secara *wired* maupun *wireless*. Komunikasi wired digunakan untuk jarak pendek antarkomponen di dalam UAV dengan keunggulan latensi rendah dan laju transfer tinggi. Sementara itu, komunikasi wireless digunakan untuk jarak jauh, seperti antara UAV dan GCS, dengan mobilitas lebih tinggi namun lebih rentan terhadap interferensi. Protokol komunikasi berfungsi sebagai aturan pengiriman data. Pada level perangkat keras, protokol yang umum digunakan antara lain UART, I2C, dan SPI. UART bersifat asinkron dan sering digunakan untuk

menghubungkan FC dengan modul telemetri, GNSS, dan Companion Computer. I2C dan SPI bersifat sinkron, dengan SPI digunakan untuk komponen berkecepatan tinggi seperti IMU dan blackbox. Pada level perangkat lunak, MAVLink digunakan sebagai protokol komunikasi pesan antar komponen UAV dan GCS. MAVLink bersifat ringan dan efisien, serta lebih sering digunakan di atas UDP untuk meminimalkan latensi. Setiap UAV memiliki System ID dan Component ID sebagai identitas, serta secara berkala mengirimkan pesan *heartbeat* untuk memastikan koneksi tetap aktif.

Logika Operasi dan Failsafe

Materi ini juga membahas logika operasi UAV melalui konsep *system state* dan *flight mode*. Sebelum terbang, UAV melewati tahapan initialization, pre-arm check, disarm, dan arm. Setelah itu, UAV dapat beroperasi dalam berbagai flight mode seperti Stabilize, Alt-Hold, Loiter, RTL, Auto, dan Guided. Untuk mengantisipasi kegagalan sistem, UAV dilengkapi dengan mekanisme failsafe. Pemicu failsafe dapat berupa hilangnya sinyal RC, terputusnya koneksi GCS, kondisi baterai kritis, gangguan sensor navigasi, atau pelanggaran geofence. Dalam kondisi tersebut, FC dapat mengambil tindakan otomatis seperti Return to Launch (RTL), mendarat, atau menghentikan pergerakan UAV.

Materi arsitektur sistem UAV ini sangat membuka wawasan karena menunjukkan betapa kompleksnya integrasi antara hardware, software, dan komunikasi. Walaupun informasinya cukup padat dan sempat terasa overwhelming, materi ini membantu membangun pemahaman fundamental yang penting untuk pengembangan software UAV di RSC.

Dasar Pemrograman (OOP, Design Pattern, dan Concurrency)

Materi ini membahas dasar-dasar pemrograman yang menjadi fondasi pengembangan software di RSC, khususnya untuk sistem UAV dan Ground Control Station (GCS). Fokus utama materi meliputi permasalahan pemrograman prosedural, pemrograman berorientasi objek (OOP), design pattern, error handling, serta pemrograman konkuren.

Permasalahan Pemrograman Prosedural

Pendekatan pemrograman prosedural cenderung menggunakan banyak fungsi global dan struktur data yang saling bergantung. Pada sistem kompleks seperti UAV dan GCS, pendekatan ini sering menimbulkan kode yang sulit dibaca, sulit dirawat, dan rawan error, terutama ketika sistem berkembang dan jumlah fitur bertambah. Kondisi ini sering disebut sebagai *spaghetti code*.

Pemrograman Berorientasi Objek (Object-Oriented Programming)

OOP diperkenalkan sebagai solusi untuk mengelola kompleksitas sistem dengan memodelkan komponen software sebagai objek yang merepresentasikan entitas nyata di sistem UAV.

Konsep utama OOP yang dipelajari meliputi:

- **Class dan Object**. Class berfungsi sebagai blueprint, sedangkan object merupakan instansi dari class tersebut di memori.
- **Encapsulation**. Data dan fungsi dibungkus dalam satu kesatuan untuk mencegah akses langsung yang tidak diinginkan.

- **Abstraction**. Menyembunyikan detail implementasi dan hanya menampilkan antarmuka yang relevan.
- **Inheritance**. Memungkinkan class baru mewarisi atribut dan method dari class lain untuk menghindari duplikasi kode.
- **Polymorphism**. Satu interface dapat memiliki banyak implementasi dengan perilaku yang berbeda.

Contoh Implementasi OOP

Sebagai contoh, koneksi UAV pada sistem GCS dapat dimodelkan sebagai sebuah class.

Listing 1: Contoh class koneksi UAV

```
class DroneConnection:
    def __init__(self, system_id):
        self.system_id = system_id
        self._connected = False

    def connect(self):
        self._connected = True

    def disconnect(self):
        self._connected = False
```

Dengan pendekatan ini, setiap UAV dapat direpresentasikan sebagai object yang terpisah dan dikelola secara modular.

Design Pattern

Design Pattern diperkenalkan sebagai solusi umum yang sudah teruji untuk permasalahan desain software yang sering muncul. Design pattern membantu engineer menulis kode yang lebih rapi, scalable, dan mudah dipahami oleh orang lain.

Contoh design pattern yang relevan dengan sistem UAV dan GCS antara lain:

- **Singleton**, untuk memastikan hanya ada satu instance, misalnya pada modul koneksi atau logger.
- **Observer**, untuk memantau perubahan data seperti telemetri UAV dan memperbarui UI secara otomatis.
- **Factory**, untuk membuat object UAV atau payload tanpa bergantung pada implementasi spesifik.

Error Handling (Try-Catch / Exception)

Materi ini juga membahas error handling menggunakan mekanisme exception. Pendekatan ini memungkinkan program menangani error secara terstruktur tanpa menghentikan seluruh sistem.

Listing 2: Contoh error handling menggunakan try-except

```
try:
    drone.connect()
except Exception as e:
    print(f"Connection␣error:␣{e}")
```

Error handling sangat penting pada sistem UAV karena kegagalan komunikasi, sensor, atau hardware dapat terjadi sewaktu-waktu.

Pemrograman Konkuren (Multithreading)

Pemrograman konkuren memungkinkan program menjalankan beberapa task secara bersamaan. Dalam sistem GCS, concurrency diperlukan untuk menangani telemetri, input pengguna, dan pengiriman perintah secara paralel tanpa saling menghambat.

Listing 3: Contoh multithreading sederhana

```
import threading

def telemetry_task():
    pass

telemetry_thread = threading.Thread(target=telemetry_task)
telemetry_thread.start()
```

Race Condition

Race condition terjadi ketika beberapa thread mengakses dan memodifikasi resource yang sama secara bersamaan tanpa sinkronisasi yang tepat. Hal ini dapat menyebabkan data menjadi tidak konsisten dan sulit dideteksi.

Mutex dan Sinkronisasi

Untuk mencegah race condition, digunakan mekanisme mutex atau lock. Mutex memastikan bahwa hanya satu thread yang dapat mengakses resource tertentu pada satu waktu.

Listing 4: Contoh penggunaan mutex untuk mencegah race condition

```
import threading

lock = threading.Lock()
shared_data = {}

def update_data(data):
    with lock:
        shared_data.update(data)
```

Relevansi dengan Sistem UAV dan GCS

Seluruh konsep pada materi ini sangat relevan dengan pengembangan software UAV dan GCS. OOP dan design pattern membantu membangun sistem yang terstruktur dan mudah dikembangkan, sedangkan concurrency memungkinkan sistem real-time berjalan secara responsif dan stabil.

Materi dasar pemrograman ini menunjukkan bahwa tantangan utama dalam pengembangan software UAV bukan hanya pada algoritma, tetapi juga pada desain sistem dan pengelolaan kompleksitas. Walaupun beberapa konsep seperti OOP cukup familiar, topik concurrency dan design pattern menuntut pemahaman yang lebih dalam karena dampaknya sangat besar terhadap stabilitas dan maintainability sistem.

Hari pertama ini cukup melelahkan namun membuka banyak perspektif baru. Selain memahami struktur RSC dan gambaran besar sistem UAV, aku juga belajar bahwa dokumentasi teknis adalah bagian penting dari proses engineering. Walaupun sempat mengalami kesulitan saat belajar LaTeX, proses tersebut menjadi latihan kemandirian dalam mencari solusi dan memahami tools yang akan sering digunakan ke depannya.

1.2 DAY 2 - Jumat, 30 Januari 2026

Hari kedua di Pendidikan RSC Aksantara 2026 melanjutkan materi yang dipelajari untuk mantepin materi kita. Untuk day 2 ini mempelajari tentang **Protokol MAVLink dan SITL**.

MAVLink (Micro Air Vehicle Link)

Pada sesi ini, kita mempelajari tentang MAVLink atau Micro Air Vehicle Link yaitu protokol komunikasi pesan open source yang sangat ringan karena hanya beberapa puluh byte. Ini penting karena jaringan nirkabel punya bandwidth yang terbatas. MAVLink ini sudah menjadi standar untuk di riset dan industri, dipakai oleh sistem autopilot besar seperti PX4 dan ArduPilot. Selain itu, MAVLink mendukung komunikasi dua arah yang efisien dan bisa mengintegrasikan wahana UAV ke dalam jaringan internet.

System ID dan Component ID

Setiap entitas yang bisa mengirim atau menerima pesan MAVLink harus punya identitas unik agar tidak tertukar. Suatu sistem MAVLink (Wahana/GCS) bisa mempunyai banyak komponen.

System ID

- Merujuk pada identitas unik untuk satu sistem UAV.
- Rentang nilai 1 sampai 255.
- System ID 255 dialokasikan untuk GCS.
- Semua komponen dalam wahana yang sama mempunyai System ID yang sama.

Component ID

- Merujuk pada identitas spesifik untuk setiap komponen atau subsistem di dalam suatu wahana.
- Sudah ada penomoran bawaan dari MAVLINK:
  - FC: 1
  - Kamera: 100-105
  - Gimbal: 154
  - CC: 191-194

Messages

MAVLink punya berbagai jenis pesan untuk pertukaran data antarkomponen dalam sistem UAV. Tiap pesan punya ID unik yakni **Message ID**. Untuk MAVLink v1 ada 255, untuk MAVLink v2 hingga 16 juta. **Dialect** kumpulan definisi pesan yang dapat disesuaikan dengan kebutuhan sistem tertentu.

Table 1: Daftar MAVLink Messages

Message Name	MSG ID	Tipe	Fungsi Utama
HEARTBEAT	0	STATE	Menandakan sistem aktif (dikirim setiap 1 detik).
SYS STATUS	1	STATE	Status kesehatan sensor dan daya baterai.
ATTITUDE	30	STATE	Mengirimkan data kemiringan wahana (Roll, Pitch, Yaw).
GLOBAL POSITION INT	33	STATE	Posisi global (Lintang, Bujur, Ketinggian).
COMMAND LONG	76	COMMAND	Mengirim perintah aksi seperti ARM atau TAKEOFF.
MISSION ITEM	39	COMMAND	Mengirim titik koordinat rute terbang (waypoint).

Pola Interaksi dengan MAVLink

Streaming (Telemetry)

- Dipakai untuk data yang harus dipantau terus-menerus
- Bersifat broadcast (satu arah) secara periodik dari wahana ke GCS
- Mengandalkan Message ID untuk menentukan jenis data tanpa alamat tujuan spesifik.
- Use cases: untuk pesan bertipe State Messages
  - HEARTBEAT
  - ATTITUDE
  - GLOBAL POSITION INT

Microservices

- Dipakai untuk perintah kritis yang butuh konfirmasi balasan.
- Bersifat point-to-point (dua arah), mirip client-service.
- Menggunakan System ID dan Component ID agar perintah tidak salah sasaran
- Use cases: untuk pesan bertipe Command Messages
  - Upload/Download waypoints untuk misi
  - Membaca atau mengubah pengaturan internal (parameter) pada FC
  - Instruksi seperti ARM/DISARM, TAKEOFF, atau LAND.

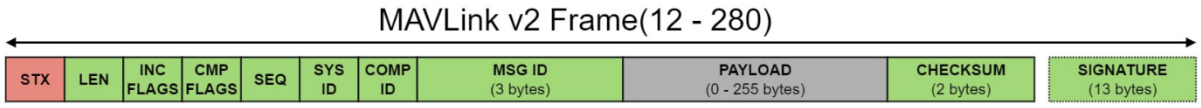

Figure 2: Anatomi Paket MAVLink

Anatomi Paket MAVLink

- Setiap paket dimulai dengan STX (0XFD) sebagai penanda bahwa paket tersebut menggunakan protokol MAVLink.
- SYS ID (System ID) dan COMP ID (Component ID) untuk identitas.
- Message ID untuk jenis message yang dikirim
- Payload membungkus konten data utama
- Checksum memastikan paket tidak corrupt saat diterima.
  - Menggunakan algoritma CRC-16 (ITU X.25) untuk menghasilkan kode verifikasi 2 byte (CKA dan CKB) menggunakan data dari LEN sampai PAYLOAD.
- Signature, seperti namanya, adalah tanda tangan digital (SHA-256) untuk memastikan bahwa paket MAVLink yang diterima berasal dari pihak terpercaya.
  - Mencegah orang orangan janggal memfabrikasi paket MAVLink dan menghack wahana.

MAVLink Interfaces

- MAVProxy: CLI-Based GCS, sangat ringan dan stabil.
- MAVSDK: SDK (software development kit) bagi developer untuk membangun aplikasi berbasis MAVLink dengan bahasa tingkat tinggi (C++, Python, dsb)
- MAVROS: SDK (software development kit) bagi developer untuk membangun aplikasi berbasis MAVLink dengan bahasa tingkat tinggi (C++, Python, dsb)

Software In The Loop (SITL)

Software In The Loop adalah metode untuk menjalankan dan menguji sistem autopilot wahana UAV tanpa menggunakan perangkat keras fisik. FC dan lingkungan (hukum fisika) disimulasikan oleh perangkat lunak. Flight stack seperti ArduPilot/PX4 dijalankan seperti program biasa di komputer. Sehingga SITL ini Sangat aman dan cocok untuk eksperimen awal dan memeriksa ketepatan implementasi logika navigasi karena tidak ada risiko kerusakan fisik akibat crash.

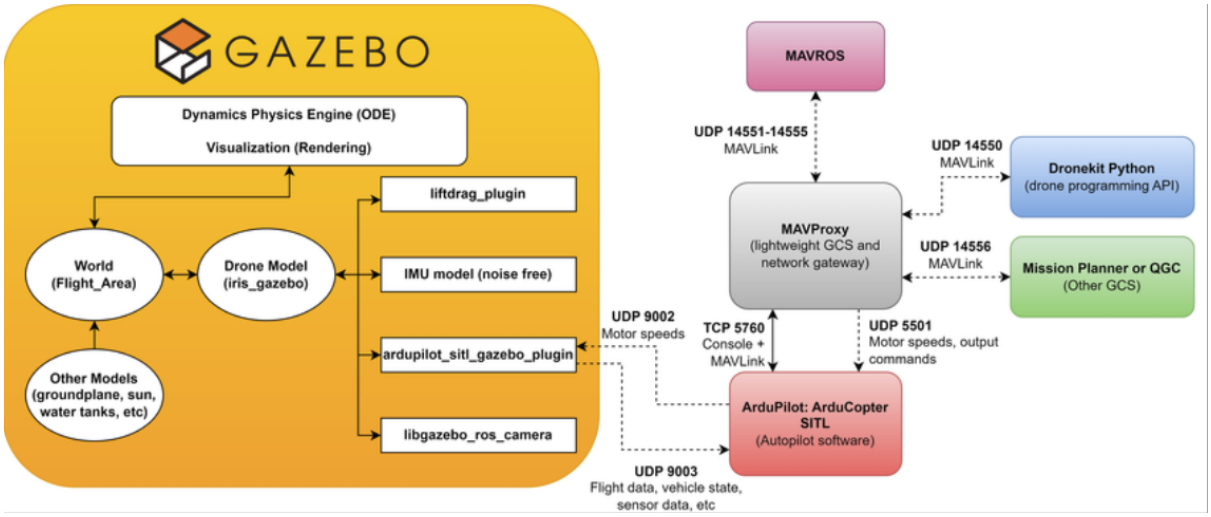

Figure 3: Arsitektur Sistem SITL

Arsitektur Sistem SITL

Tools untuk SITL

- ArduPilot: Flight stack untuk mensimulasikan Flight Controller.
- Gazebo Harmonic: Simulasi fisika dan lingkungan serta dinamika wahana yang realistis.
- MAVProxy: Jembatan komunikasi antara pengguna dan flight stack.

Untuk materi Day 2 ini aku cukup menguras tenaga dan mental, tapi aku dapet ilmu yang daging banget. Trus ada Handson yang bikin aku belajar ekstra banget tapi seru, karena aku bisa langsung coba dan praktek untuk ngeliat simulasi drone mulai dari persiapannya, gimana mereka dikontrol untuk mapping, sampe gimana cara landingnya. Jujur di materi ini aku harus baca materinya berkali kali, sambil cari referensi dari dokumentasi atau web web lain buat belajar.

1.3 Hands-On 2: MAVLink dan SITL

Pendahuluan

Hands-On 2 menjadi salah satu pengalaman yang cukup seru dan ilmu banget buat aku seputar UAV, khususnya dalam penggunaan ArduPilot SITL, Gazebo Harmonic, MAVProxy, dan MAVSDK. Pada awalnya, tugas ini terlihat cukup sederhana yaitu menerbangkan drone untuk takeoff, membentuk lintasan angka delapan, lalu melakukan landing. Namun dalam praktiknya, proses yang aku laluin jauh lebih kompleks dari yang aku bayangin.

Awal Pengerjaan

Aku mulai dengan melakukan setup environment menggunakan Distrobox Ubuntu di dalam Arch Linux. Proses instalasi mengikuti handout yang diberikan, dan secara umum berjalan dengan lancar. Setelah semua dependensi terpasang, aku coba buat run SITL

dan Gazebo buat pertama kalinya. Pas simulator berhasil kebuka, aku cukup percaya diri. Namun ternyata, itu baru permulaan.

Masalah Pertama: Drone Tidak Bergerak

Ketika aku berhasil melakukan arm dan takeoff melalui MAVProxy, indikator ketinggian menunjukkan bahwa drone sudah berada di udara. Anehnya, di Gazebo drone tetap diam di landasan. Kondisi ini janggal dan bikin otak mikir banget karena secara sistem drone harusnya udah keliatan terbang, tapi secara visual tidak terjadi apa apa. Setelah melakukan searching, coba baca baca di dokumentasi, coba trouble shoot, dan tanya AI(kalo udah mentok), aku menemukan bahwa masalah tersebut berasal dari koneksi antara ArduPilot dan plugin Gazebo yang belum sinkron. Beberapa warning seperti:

```
Incorrect protocol magic 0 should be 18458
```

ini jadi petunjuk bahwa komunikasi simulator belum berjalan dengan benar.

Setelah restart SITL dan Gazebo beberapa kali serta memastikan world yang digunakan sudah sesuai, akhirnya drone mulai merespons. Momen itu cukup bikin lega karena untuk pertama kalinya aku liat dronenya beneran lepas landas di simulator.

Masalah Kedua: Gagal Landing

Setelah berhasil terbang manual, coba coba buat landing. Nah bukannya turun, malah muncul error:

```
AP: Unable to start landing sequence
AP: Mode change to AUTO RTL failed: No landing sequence found
```

aku sempet ngira ada kesalahan besar di konfigurasi. Ternyata penyebabnya cukup simple yaitu mode penerbangan yang digunakan gak sesuai. Drone harus beradax pada mode yang mendukung perintah landing seperti GUIDED.

Dari sini saya belajar bahwa memahami flight mode bukan sekadar teori — pemilihan mode sangat menentukan apakah sebuah command dapat dijalankan atau tidak.

Transisi ke Metode Otonom

Jika metode manual sudah cukup menantang, metode otonom memberikan tingkat kesulitan berikutnya.

Saat pertama kali menjalankan script Python menggunakan MAVSDK, terminal hanya menampilkan:

```
Script started
```

lalu tidak terjadi apa-apa.

Tidak ada error. Tidak ada pergerakan drone. Hanya diam.

Saya sempat mengira program mengalami crash secara silent, tetapi ternyata script sedang menunggu koneksi ke sistem. Setelah memastikan SITL sudah berjalan dan port yang digunakan benar (`udpin://0.0.0.0:14540`), barulah drone terdeteksi.

Pelajaran penting dari tahap ini adalah: dalam sistem asynchronous, program yang terlihat ”diam” belum tentu bermasalah — bisa jadi ia sedang menunggu event tertentu.

Masalah Ketiga: Lintasan Tidak Membentuk Angka Delapan

Bagian paling memakan waktu justru terjadi ketika mencoba membentuk lintasan angka delapan.

Awalnya saya menggunakan beberapa waypoint sederhana, tetapi hasilnya tidak menyerupai angka delapan sama sekali. Drone bergerak terlalu kaku dan patah-patah karena perpindahan antar titik bersifat diskrit.

Saya mencoba:

- Memperbesar area waypoint
- Menambah jumlah titik
- Mengatur ulang koordinat NED
- Memberi jeda hover

Namun hasilnya masih belum memuaskan.

Akhirnya saya memahami bahwa pola seperti angka delapan lebih cocok dibentuk menggunakan pendekatan parametrik (misalnya sinus dan cosinus) agar lintasan menjadi halus dan kontinu. Setelah mengubah strategi tersebut, pergerakan drone terlihat jauh lebih natural di udara.

Momen ketika lintasan angka delapan akhirnya terbentuk dengan jelas menjadi salah satu titik paling memuaskan dalam pengerjaan tugas ini.

Error yang Sempat Mengkhawatirkan

Di tengah proses, saya juga sempat menemui pesan:

```
Critical failure 0x100000
flow_of_ctrl
```

Pesan ini terdengar cukup serius. Setelah investigasi, penyebabnya kemungkinan besar adalah konflik Offboard command akibat pengiriman setpoint yang terlalu cepat atau kondisi sistem yang belum stabil.

Solusinya adalah melakukan restart simulator dan memastikan urutan eksekusi Offboard benar: setpoint dikirim terlebih dahulu, baru mode Offboard diaktifkan.

Dari sini saya belajar bahwa pada autonomous control, urutan eksekusi sangat krusial.

Refleksi

Hands-On ini mengajarkan saya bahwa bekerja dengan sistem robotika bukan hanya soal menulis kode, tetapi juga tentang memahami bagaimana berbagai komponen berinteraksi.

Beberapa hal penting yang saya pelajari:

- Simulator bisa terlihat berjalan normal meskipun koneksi backend bermasalah.
- Flight mode menentukan valid atau tidaknya sebuah command.
- Autonomous flight membutuhkan logika yang runtut dan stabil.
- Warning tidak selalu berbahaya, tetapi tetap perlu dipahami.

Lebih dari itu, saya juga belajar untuk tidak langsung menyerah ketika sistem tidak bekerja sesuai ekspektasi. Sebagian besar solusi justru muncul setelah mencoba menganalisis masalah dengan lebih tenang.

Kesimpulan

Meskipun tugas ini terlihat sederhana, proses yang saya lalui penuh dengan trial and error. Dari drone yang tidak bergerak, gagal landing, script yang tampak membeku, hingga lintasan yang tidak terbentuk — semuanya menjadi bagian dari pembelajaran.

Pada akhirnya, keberhasilan menerbangkan drone secara manual dan otonom serta melihatnya membentuk angka delapan memberikan rasa pencapaian tersendiri.

Pengalaman ini membuat saya semakin memahami bahwa dalam bidang UAV dan robotika, kesabaran dan ketelitian sering kali sama pentingnya dengan kemampuan teknis.