SIMULASI KINERJA SISTEM KENDALI SCOUT DRONE INGENUITY (TAHAP 1: GERAKAN MINIMAL TIGA TITIK)

Disusun untuk Memenuhi Tugas Mata Kuliah Robotica

Dosen Pengampu :

Giga Verian Pratama, S,Si., M.T.

DISUSUN OLEH :

Agung Rambujana	221364002
Khoirul Huda	221364013

TEKNIK OTOMASI INDUSTRI

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

1. Tujuan

- Memahami prinsip kerja rotor dan sistem kendali helikopter Ingenuity
- Mengintegrasikan model aerodinamika dan kontrol penerbangan ke dalam simulasi virtual.
- Mensimulasikan proses lepas landas, melayang, dan pendaratan helikopter secara otonom.
- Menilai performa dan stabilitas helikopter dalam skenario yang diinginkan

2. Dasar Teori

Helikopter Ingenuity dirancang sebagai drone/vehicle/sarana udara pertama yang mampu terbang di atmosfer Mars yang sangat tipis. Untuk menghasilkan gaya angkat, Ingenuity menggunakan dua rotor koaksial berputar berlawanan arah berkecepatan tinggi (hingga 2800 RPM), dengan struktur ringan agar dapat terbang meskipun daya dorong yang dihasilkan terbatas akibat rendahnya kerapatan udara.

Sistem kendali penerbangannya mencakup pengendalian otomatis selama fase penting seperti lepas landas, melayang, dan mendarat. Saat takeoff, helikopter naik dengan cepat sebelum beralih ke kontrol tertutup untuk stabilisasi. Saat landing, kecepatan turun dijaga konstan dan sistem secara cepat mendeteksi kontak dengan permukaan Mars untuk memastikan pendaratan aman.

Untuk mendukung navigasi dan misi ilmiah, Ingenuity dilengkapi kamera (probe camera) beresolusi tinggi yang merekam permukaan Mars dari udara. Kamera ini membantu menghasilkan citra medan, pemetaan digital, dan perencanaan rute rover, sekaligus memberi perspektif visual baru yang tidak dapat diperoleh dari rover atau satelit.

3. Alat dan Bahan

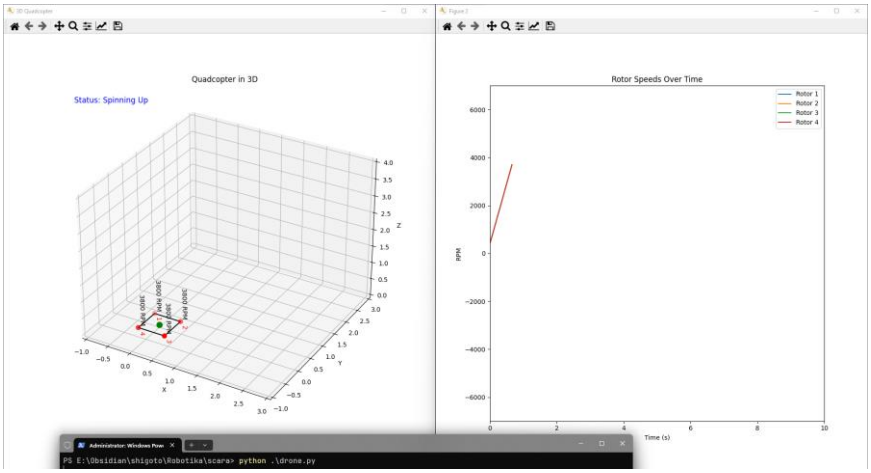
- Laptop (menjalankan simulasi dan visualisasi)
- Mouse (navigasi tampilan selama simulasi)
- Python (bahasa pemrograman utama)
- NumPy (library operasi matematis dan vektor)
- Matplotlib (library visualisasi 2D, 3D, dan animasi)
- mpl_toolkits.mplot3d (membuat grafik 3D)
- matplotlib.animation (membuat animasi pergerakan)
- Logging (modul Python) (mencatat data kecepatan rotor dan waktu waypoint)

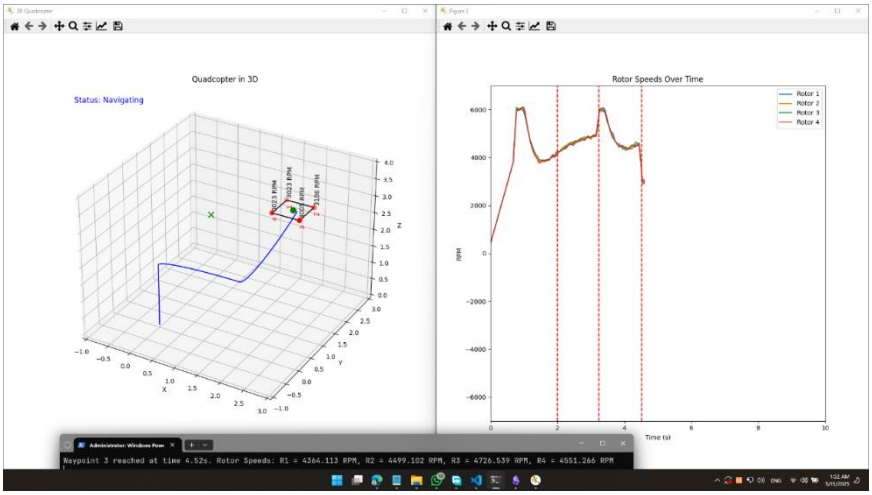
4. Prosedur Kerja

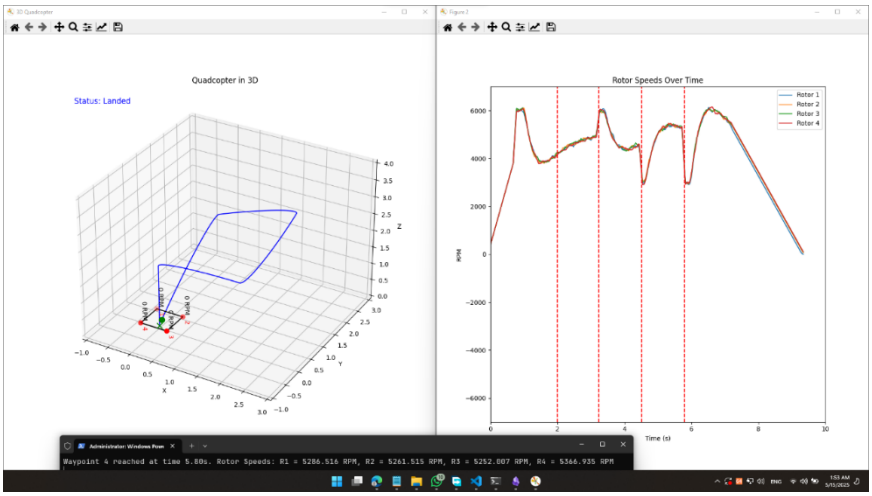
1. Instal Python 3.x di komputer.
2. Install library numpy dan matplotlib dengan pip install numpy matplotlib.
3. Siapkan direktori kerja dan pastikan file drone.py tersedia.
4. Pelajari fungsi utama dalam kode seperti control_input, update_state, dan animate.
5. Pahami cara kerja visualisasi 3D dan grafik rotor dengan matplotlib.
6. Tinjau sistem logging ke file waypoint_rotor_speeds.log.
7. Jalankan simulasi dengan perintah python drone.py.
8. Amati dua jendela: visualisasi 3D dan grafik kecepatan rotor.
9. Periksa isi file log untuk memastikan data tercatat dengan benar.
10. Debug jika ada error dengan membaca pesan di terminal.
11. Ubah waypoint untuk mencoba lintasan baru.
12. Uji pengaruh perubahan parameter seperti kp, kd, dt, atau L.
13. Tambahkan fitur baru seperti log tambahan atau grafik kecepatan.
14. Perbarui README jika ada penambahan fitur atau perubahan besar.
15. Catat hasil pengujian dan eksperimen untuk referensi.
16. Finalisasi dan bersihkan kode sebelum penyimpanan akhir.
17. Backup proyek ke GitHub atau media penyimpanan lain.

5. Data dan Hasil

5.1. Gambar simulasi







5.2. Coding

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
import logging

# Constants
g, m, dt, L = 9.81, 0.5, 0.02, 0.6
state = np.array([0, 0, 0, 0, 0, 0], dtype=float)
waypoints = [np.array([0, 0, 2]), np.array([2, 0, 2]), np.array([2, 2, 3]), np.array([0, 2, 2]), np.array([0, 0,
0])]
wp_index = 0
rotor_speeds = np.array([0.0] * 4)
speed_history, time_vals = [], []

# Spin-up variables
startup_rpm = 4000
spinup_step = 100  # RPM per frame
spinup_done = False

# Set up logging to both file and console
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# File handler
file_handler = logging.FileHandler('waypoint_rotor_speeds.log')
file_handler.setFormatter(logging.Formatter('%(asctime)s - %(message)s'))
logger.addHandler(file_handler)

# Console handler
console_handler = logging.StreamHandler()
console_handler.setFormatter(logging.Formatter('%(asctime)s - %(message)s'))
logger.addHandler(console_handler)

def control_input(state, target):
    pos, vel = state[:3], state[3:]
    kp, kd = 6.0, 4.0
    acc_des = kp * (target - pos) - kd * vel
    acc_des[2] += g
    thrust_total = m * acc_des[2]
    base_speed = np.clip(thrust_total * 1000, 3000, 6000)
    rotor_speeds[:] = base_speed + np.random.randn(4) * 100
    return m * acc_des

def update_state(state, u):
    pos, vel = state[:3], state[3:]
    acc = u / m
    acc[2] -= g
    vel += acc * dt
    pos += vel * dt
    return np.hstack((pos, vel))

def rotor_positions(center):
    offsets = np.array([[-L/2, L/2, 0], [L/2, L/2, 0], [L/2, -L/2, 0], [-L/2, -L/2, 0]])
    return center + offsets

# --- 3D Visualization Setup ---
fig = plt.figure("3D Quadcopter")
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim(-1, 3)
ax.set_ylim(-1, 3)
ax.set_zlim(0, 4)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("Quadcopter in 3D")

trajectory = []
traj_line, = ax.plot([], [], [], 'b')
rotor_lines = [ax.plot([], [], [], 'k-')[0] for _ in range(4)]
rotor_dots = ax.scatter([], [], [], c='r', s=50)
center_dot = ax.scatter([], [], [], c='green', s=80)
target_dot = ax.scatter([], [], [], c='green', marker='x', s=80)
rpm_texts = [ax.text(0, 0, 0, '') for _ in range(4)]
rotor_number_texts = [ax.text(0, 0, 0, f'{i+1}', color='red') for i in range(4)]
status_text = ax.text2D(0.05, 0.95, "Status: Spinning Up", transform=ax.transAxes, fontsize=12, color='blue')

# --- Rotor Speed Plot ---
fig2, ax2 = plt.subplots()
speed_lines = [ax2.plot([], [], label=f'Rotor {i+1}')[0] for i in range(4)]
```

```python
ax2.set_xlim(0, 10)
ax2.set_ylim(-7000, 7000)
ax2.set_xlabel("Time (s)")
ax2.set_ylabel("RPM")
ax2.set_title("Rotor Speeds Over Time")
ax2.legend()

def moving_average(data, window_size=5):
    if len(data) < window_size:
        return data
    return np.convolve(data, np.ones(window_size)/window_size, mode='valid')

# Add a list to store waypoint reach times
waypoint_reach_times = []
# Add a set to track already plotted waypoint times
plotted_waypoint_times = set()

def animate(i):
    global state, wp_index, rotor_speeds, spinup_done

    t = i * dt

    if not spinup_done:
        # Spin-up phase
        rotor_speeds[:] = np.minimum(rotor_speeds + spinup_step, startup_rpm)
        if np.all(rotor_speeds >= startup_rpm):
            spinup_done = True
            status_text.set_text("Status: Navigating")
        else:
            status_text.set_text("Status: Spinning Up")
    else:
        target = waypoints[wp_index]

        if wp_index == len(waypoints) - 1 and np.linalg.norm(state[:3] - target) < 0.2 and state[2] <= 0.1:
            rotor_speeds[:] = np.maximum(rotor_speeds - 50, 0)
            for j in range(4):
                rpm_texts[j].set_text(f'{rotor_speeds[j]:.0f} RPM')
            if np.all(rotor_speeds == 0):
                status_text.set_text("Status: Landed")
                return
        else:
            u = control_input(state, target)
            state[:] = update_state(state, u)
            trajectory.append(state[:3].copy())

            if np.linalg.norm(state[:3] - target) < 0.2 and wp_index < len(waypoints) - 1:
                wp_index += 1
                waypoint_reach_times.append(t)  # Record the time when the waypoint is reached

                # Format rotor speeds in list style
                formatted_speeds = [f"R{j+1} = {rotor_speeds[j]:.3f} RPM" for j in range(4)]
                formatted_speeds_str = ", ".join(formatted_speeds)

                # Log and print rotor speeds
                log_message = f"Waypoint {wp_index} reached at time {t:.2f}s. Rotor Speeds:
{formatted_speeds_str}"
                logger.info(log_message)
                print(log_message)

    # --- Update 3D elements ---
    if trajectory:
        traj = np.array(trajectory)
        traj_line.set_data(traj[:, 0], traj[:, 1])
        traj_line.set_3d_properties(traj[:, 2])

    center = state[:3]
    rotors = rotor_positions(center)

    for j in range(4):
        p1, p2 = rotors[j], rotors[(j+1)%4]
        rotor_lines[j].set_data([p1[0], p2[0]], [p1[1], p2[1]])
        rotor_lines[j].set_3d_properties([p1[2], p2[2]])

        rpm_texts[j].set_position((rotors[j][0], rotors[j][1]))
        rpm_texts[j].set_3d_properties(rotors[j][2]+0.1)
        rpm_texts[j].set_text(f'{rotor_speeds[j]:.0f} RPM')

        rotor_number_texts[j].set_position((rotors[j][0], rotors[j][1]))
        rotor_number_texts[j].set_3d_properties(rotors[j][2] - 0.2)

    rotor_dots._offsets3d = (rotors[:, 0], rotors[:, 1], rotors[:, 2])
    center_dot._offsets3d = ([center[0]], [center[1]], [center[2]])
    if spinup_done:
        target = waypoints[wp_index]
        target_dot._offsets3d = ([target[0]], [target[1]], [target[2]])
```

```
        # --- Rotor Speeds Plot ---
        time_vals.append(t)
        speed_history.append(rotor_speeds.copy())
        speeds = np.array(speed_history)

        filtered_speeds = np.array([moving_average(speeds[:, j]) for j in range(4)]).T

        if t > ax2.get_xlim()[1]:
            ax2.set_xlim(0, t + 5)
        for j in range(4):
            speed_lines[j].set_data(time_vals[:len(filtered_speeds)], filtered_speeds[:, j])

        # Add vertical lines for waypoint reach times (only once per waypoint)
        for waypoint_time in waypoint_reach_times:
            if waypoint_time not in plotted_waypoint_times:
                ax2.axvline(x=waypoint_time, color='red', linestyle='--', label='Waypoint Reached')
                plotted_waypoint_times.add(waypoint_time)

        ax2.relim()
        ax2.autoscale_view()

        fig2.canvas.draw()
        fig2.canvas.flush_events()

    ani = animation.FuncAnimation(fig, animate, frames=500, interval=10, blit=False)
    plt.show()
```

6. Analisa

- Drone mampu mencapai 4 dari 5 waypoint yang ditentukan dengan tolenransi yang cukup dekat dengan posisi yang diinginkan.
- Waypoint terakhir tidak dapat dicapai karena deviasi karena respons drone terhadap perintah navigasi mungkin tidak optimal, misalnya karena parameter PID yang belum disetel dengan baik, menyebabkan drone melenceng dari lintasan yang diharapkan.
- Berdasarkan perbandingan dua grafik kecepatan rotor, terlihat bahwa pada simulasi waypoint tracking, rotor menunjukkan empat lonjakan utama kecepatan yang merepresentasikan keberhasilan mencapai empat waypoint. Masing-masing lonjakan diikuti oleh fase stabilisasi. Namun, menjelang akhir misi, terjadi ketidakseimbangan kecepatan antar rotor, yang tampaknya menjadi faktor utama kegagalan drone mencapai waypoint kelima. Sebaliknya, pada grafik penerbangan Flight 4 yang merepresentasikan profil kecepatan vertikal, performa drone terlihat lebih stabil. Kecepatan rotor menunjukkan akselerasi awal yang cepat dan konsisten saat takeoff (climb), kemudian berlanjut dengan fase descent yang stabil dan touchdown yang terkendali tanpa adanya lonjakan berbahaya. Hal ini mengindikasikan bahwa sistem kontrol pada Flight 4 (jurnal) lebih responsif dan presisi dibandingkan simulasi waypoint tracking.

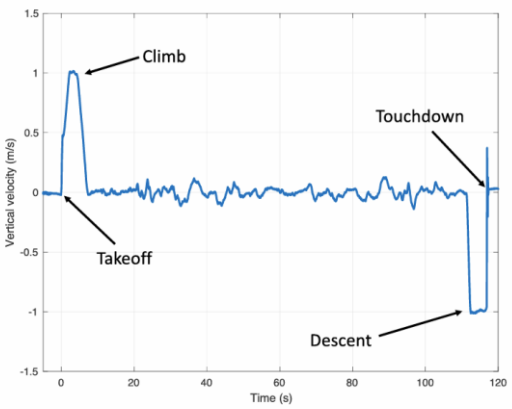
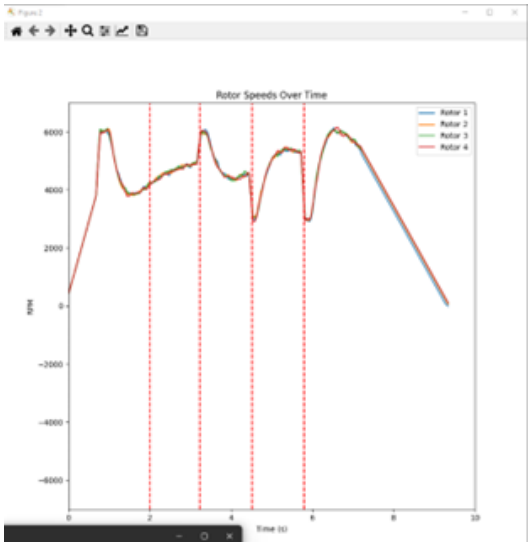


Figure 11 Estimated vertical velocity during Flight 4. Note the rapid acceleration on takeoff and the steady descent speed, both of which are key robustness metrics.



Karakteristik simulasi dari jurnal

Karakteristik simulasi praktek

- Simulasi memerlukan penerapan fungsi dinamika rotasi untuk merepresentasikan perilaku rotasi bodi drone secara akurat, khususnya saat melakukan manuver berbelok. Hal ini dikarenakan dalam kondisi tersebut, sebagian rotor harus berputar lebih cepat atau lebih lambat dibandingkan rotor lainnya untuk menghasilkan momen puntir (torque) yang dibutuhkan. Tanpa model dinamika rotasi yang tepat, simulasi tidak akan mampu menggambarkan perubahan orientasi drone secara realistis, sehingga mengurangi akurasi dalam prediksi trajektori serta respons kontrol selama manuver.
- Log:
    - Waypoint 1 reached at time 2.00s. Rotor Speeds: R1 = 4210.718 RPM, R2 = 4102.003 RPM, R3 = 4259.993 RPM, R4 = 4032.104 RPM
    - Waypoint 2 reached at time 3.24s. Rotor Speeds: R1 = 4916.699 RPM, R2 = 5018.638 RPM, R3 = 5043.915 RPM, R4 = 5017.196 RPM
    - Waypoint 3 reached at time 4.52s. Rotor Speeds: R1 = 4364.113 RPM, R2 = 4499.102 RPM, R3 = 4726.539 RPM, R4 = 4551.266 RPM
    - Waypoint 4 reached at time 5.80s. Rotor Speeds: R1 = 5286.516 RPM, R2 = 5261.515 RPM, R3 = 5252.007 RPM, R4 = 5366.935 RPM
- **Waypoint 1 (2.00s):** Rotor menunjukkan kecepatan bervariasi antara 4032 hingga 4259 RPM. Variasi kecil ini menunjukkan manuver awal untuk lepas landas dan menuju waypoint pertama, dengan perbedaan kecepatan antar rotor menciptakan kestabilan awal dan sedikit rotasi horizontal.
- **Waypoint 2 (3.24s):** Terdapat peningkatan kecepatan rotor secara signifikan, dengan kisaran sekitar 4916–5043 RPM. Ini menandakan drone mempercepat laju dan ketinggian untuk mencapai waypoint kedua yang kemungkinan berada lebih tinggi atau membutuhkan kecepatan translasi yang lebih besar.
- **Waypoint 3 (4.52s):** Kecepatan rotor sedikit turun dari sebelumnya, tetapi tetap cukup tinggi (sekitar 4364–4726 RPM). Ini bisa disebabkan oleh kebutuhan untuk menstabilkan posisi atau menghadapi perubahan arah, terlihat dari ketidakseimbangan kecil antar rotor.

- **Waypoint 4 (5.80s):** Terjadi lonjakan kecepatan rotor secara keseluruhan, dengan R4 menyentuh 5366 RPM, tertinggi dari semua data. Ini menunjukkan manuver signifikan, kemungkinan berupa perubahan arah tajam atau pendakian vertikal untuk menyesuaikan posisi ke waypoint 4.
- **Kegagalan Mencapai Waypoint 5:** Setelah waypoint 4, kecepatan rotor mulai menurun dan menunjukkan adanya ketidakstabilan atau ketidaksesuaian daya dorong. Hal ini bisa mengindikasikan kegagalan mencapai waypoint 5 akibat deviasi lintasan yang besar atau kurangnya kompensasi kontrol, misalnya karena kurang akuratnya model dinamika rotasi atau keterbatasan respons aktuator.

7. Kesimpulan

Berdasarkan hasil simulasi dan analisis grafik rotor, dapat disimpulkan bahwa sistem navigasi drone mampu mengarahkan drone hingga mencapai 4 dari 5 waypoint yang ditentukan dengan tingkat akurasi yang cukup baik. Keberhasilan ini tercermin dari pola lonjakan kecepatan rotor yang konsisten setiap kali drone mencapai sebuah waypoint, diikuti oleh fase stabilisasi. Namun, kegagalan dalam mencapai waypoint kelima menunjukkan adanya keterbatasan pada sistem kontrol, kemungkinan besar disebabkan oleh parameter PID yang belum optimal atau kurang lengkapnya model dinamika rotasi dalam simulasi. Ketidakseimbangan kecepatan antar rotor pada fase akhir memperkuat dugaan bahwa simulasi belum sepenuhnya mampu merepresentasikan dinamika rotasi bodi drone saat manuver kompleks.

Jika dibandingkan dengan simulasi dalam jurnal (Flight 4), performa sistem kontrol pada Flight 4 lebih stabil dan presisi, terutama dalam menjaga kecepatan vertikal yang konstan saat takeoff dan descent. Ini menunjukkan bahwa penerapan model kontrol dan dinamika yang lebih matang, termasuk pengaruh rotasi, sangat berperan dalam meningkatkan keandalan drone selama penerbangan. Oleh karena itu, untuk meningkatkan akurasi dan keberhasilan misi navigasi secara keseluruhan, diperlukan penyempurnaan model dinamika rotasi serta penyetelan parameter kontrol yang lebih cermat.