```
import math
import matplotlib.pyplot as plt
from typing import List, Tuple, Dict, Union



Waypoint = Dict[str, Union[float, int]]  # {"x": float, "y": float, "z": float, "t": int}
DroneMission = List[Waypoint]



def euclidean_distance(p1: Waypoint, p2: Waypoint, use_3d: bool = False) -> float:
    if use_3d:
        return math.sqrt((p1['x'] - p2['x']) ** 2 + (p1['y'] - p2['y']) ** 2 + (p1['z'] - p2['z']) ** 2)
    return math.sqrt((p1['x'] - p2['x']) ** 2 + (p1['y'] - p2['y']) ** 2)

def time_overlap(t1: Tuple[int, int], t2: Tuple[int, int]) -> bool:
    return max(t1[0], t2[0]) < min(t1[1], t2[1])



def check_conflict(primary: DroneMission, others: List[DroneMission], buffer: float = 5.0, use_3d: bool = False):
    conflicts = []
    for i, p_wp in enumerate(primary[:-1]):
        p_next = primary[i + 1]
        p_time = (p_wp['t'], p_next['t'])

        for drone_id, sim in enumerate(others):
            for j, s_wp in enumerate(sim[:-1]):
                s_next = sim[j + 1]
                s_time = (s_wp['t'], s_next['t'])

                if time_overlap(p_time, s_time):
                    dist = euclidean_distance(p_wp, s_wp, use_3d)
                    if dist < buffer:
                        conflicts.append({
                            "time": (max(p_time[0], s_time[0]), min(p_time[1], s_time[1])),
                            "location": ((p_wp['x'] + s_wp['x']) / 2, (p_wp['y'] + s_wp['y']) / 2),
                            "drone": drone_id
                        })
    return conflicts


def check_mission_safety(primary: DroneMission, others: List[DroneMission], use_3d: bool = False):
    conflicts = check_conflict(primary, others, use_3d=use_3d)
    if conflicts:
        return "conflict detected", conflicts
    return "clear", None



def visualize(primary: DroneMission, others: List[DroneMission], conflicts):
    plt.figure(figsize=(10, 8))

    # Primary drone
    px = [wp['x'] for wp in primary]
    py = [wp['y'] for wp in primary]
    plt.plot(px, py, 'bo-', label='Primary Drone')

    # Other drones
    for idx, sim in enumerate(others):
        sx = [wp['x'] for wp in sim]
        sy = [wp['y'] for wp in sim]
        plt.plot(sx, sy, '--', label=f'Simulated Drone {idx}')

    # Conflicts
    if conflicts:
        for c in conflicts:
            plt.plot(c['location'][0], c['location'][1], 'rx')
            plt.text(c['location'][0], c['location'][1], f"D{c['drone']}@{c['time']}", fontsize=8)

    plt.legend()
    plt.title("Drone Trajectory and Conflict Zones")
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    plt.grid(True)
    plt.show()


if __name__ == '__main__':
    primary_mission = [
```

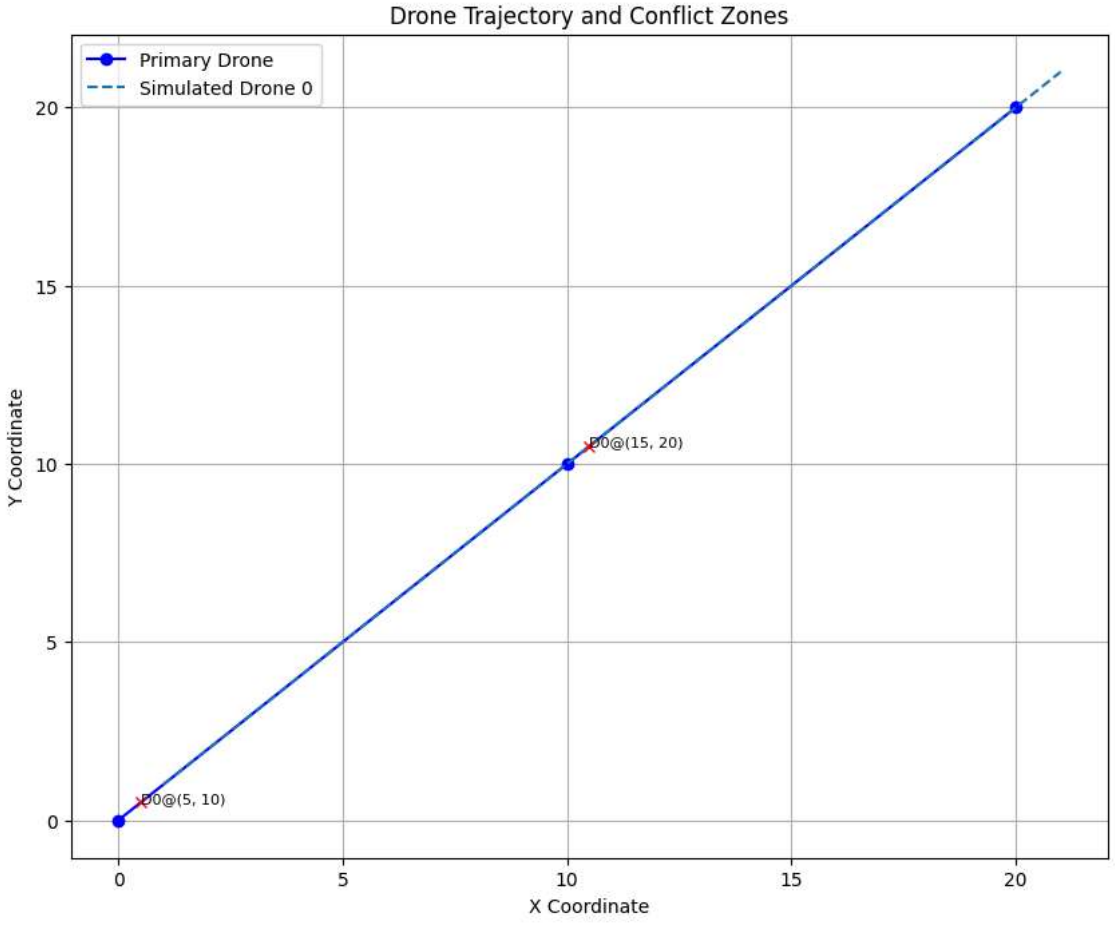
```
        {"x": 0, "y": 0, "z": 10, "t": 0},
        {"x": 10, "y": 10, "z": 10, "t": 10},
        {"x": 20, "y": 20, "z": 10, "t": 20},
    ]

    other_drones = [[
        {"x": 1, "y": 1, "z": 10, "t": 5},
        {"x": 11, "y": 11, "z": 10, "t": 15},
        {"x": 21, "y": 21, "z": 10, "t": 25},
    ]]

    status, details = check_mission_safety(primary_mission, other_drones, use_3d=True)
    print("Status:", status)
    if details:
        for d in details:
            print(d)

    visualize(primary_mission, other_drones, details)
```

```
Status: conflict detected
{'time': (5, 10), 'location': (0.5, 0.5), 'drone': 0}
{'time': (15, 20), 'location': (10.5, 10.5), 'drone': 0}
```




```
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from typing import List, Tuple, Dict, Union


Waypoint = Dict[str, Union[float, int]]  # {"x": float, "y": float, "z": float, "t": int}
DroneMission = List[Waypoint]



def euclidean_distance(p1: Waypoint, p2: Waypoint, use_3d: bool = False) -> float:
    if use_3d:
        return math.sqrt((p1['x'] - p2['x']) ** 2 + (p1['y'] - p2['y']) ** 2 + (p1['z'] - p2['z']) ** 2)
    return math.sqrt((p1['x'] - p2['x']) ** 2 + (p1['y'] - p2['y']) ** 2)

def time_overlap(t1: Tuple[int, int], t2: Tuple[int, int]) -> bool:
    return max(t1[0], t2[0]) < min(t1[1], t2[1])
```

```
def check_conflict(primary: DroneMission, others: List[DroneMission], buffer: float = 5.0, use_3d: bool = False):
    conflicts = []
    for i, p_wp in enumerate(primary[:-1]):
        p_next = primary[i + 1]
        p_time = (p_wp['t'], p_next['t'])

        for drone_id, sim in enumerate(others):
            for j, s_wp in enumerate(sim[:-1]):
                s_next = sim[j + 1]
                s_time = (s_wp['t'], s_next['t'])

                if time_overlap(p_time, s_time):
                    dist = euclidean_distance(p_wp, s_wp, use_3d)
                    if dist < buffer:
                        conflicts.append({
                            "time": (max(p_time[0], s_time[0]), min(p_time[1], s_time[1])),
                            "location": ((p_wp['x'] + s_wp['x']) / 2, (p_wp['y'] + s_wp['y']) / 2, (p_wp['z'] + s_wp['z']) / 2 if use_3d
                            "drone": drone_id
                        })
    return conflicts


def check_mission_safety(primary: DroneMission, others: List[DroneMission], use_3d: bool = False):
    conflicts = check_conflict(primary, others, use_3d=use_3d)
    if conflicts:
        return "conflict detected", conflicts
    return "clear", None



def visualize_3d(primary: DroneMission, others: List[DroneMission], conflicts):
    fig = plt.figure(figsize=(12, 9))
    ax = fig.add_subplot(111, projection='3d')

    # Primary drone
    px = [wp['x'] for wp in primary]
    py = [wp['y'] for wp in primary]
    pz = [wp['z'] for wp in primary]
    ax.plot(px, py, pz, 'bo-', label='Primary Drone')

    # Other drones
    for idx, sim in enumerate(others):
        sx = [wp['x'] for wp in sim]
        sy = [wp['y'] for wp in sim]
        sz = [wp['z'] for wp in sim]
        ax.plot(sx, sy, sz, '--', label=f'Simulated Drone {idx}')

    # Conflicts
    if conflicts:
        for c in conflicts:
            x, y, z = c['location']
            ax.scatter(x, y, z, c='r', marker='x')
            ax.text(x, y, z, f"D{c['drone']}@{c['time']}", fontsize=8)

    ax.set_title("3D Drone Trajectories and Conflict Zones")
    ax.set_xlabel("X Coordinate")
    ax.set_ylabel("Y Coordinate")
    ax.set_zlabel("Z Coordinate")
    ax.legend()
    plt.show()



if __name__ == '__main__':
    primary_mission = [
        {"x": 0, "y": 0, "z": 10, "t": 0},
        {"x": 10, "y": 10, "z": 15, "t": 10},
        {"x": 20, "y": 20, "z": 20, "t": 20},
    ]

    other_drones = [[
        {"x": 1, "y": 1, "z": 10, "t": 5},
        {"x": 11, "y": 11, "z": 15, "t": 15},
        {"x": 21, "y": 21, "z": 20, "t": 25},
    ]]

    status, details = check_mission_safety(primary_mission, other_drones, use_3d=True)
    print("Status:", status)
    if details:
        for d in details:
            print(d)
```

```
visualize_3d(primary_mission, other_drones, details)
```

```
Status: conflict detected
{'time': (5, 10), 'location': (0.5, 0.5, 10.0), 'drone': 0}
{'time': (15, 20), 'location': (10.5, 10.5, 15.0), 'drone': 0}
```