UAV Collision Detection and Path Planning

Απαλλακτική Εργασία

3Δ Υπολογιστική Γεωμετρία και Όραση

Γεώργιος Παπουτσάς

ΑΜ: 1083738

Έτος: 4ο

GitHub repo:
https://github.com/Papiqulos/Project_3D_UAV_2024

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών
Πανεπιστήμιο Πατρών
Ακαδημαϊκό Έτος 2023-2024

Περιεχόμενα

1. Σκοπός

Η παρούσα εργασία έχει ως στόχο την αναπαράσταση πολλαπλών UAVs σε τρισδιάστατο χώρο, τη δημιουργία διαφόρων περιβαλλοντικών όγκων και προβολών σε κάποια επίπεδα, και την ανίχνευση πιθανών συγκρούσεων σε κάθε χρονική στιγμή. Επιπλέον, ασχολείται με τη μοντελοποίηση της κίνησης των UAVs, τον εντοπισμό συγκρούσεων σε συγκεκριμένα χρονικά διαστήματα, και την ανάπτυξη στρατηγικών κίνησης για την αποφυγή αυτών των συγκρούσεων. Τελικός στόχος είναι η δημιουργία πρωτοκόλλων πλοήγησης, που θα επιτρέπουν στα UAVs να αποφεύγουν εμπόδια, όπως η ασφαλής απογείωση και προσγείωση.

2. Επισκόπηση της εφαρμογής

2.1 Δομή κώδικα

Η εφαρμογή έχει αναπτυχθεί με την Python και συγκεκριμένα χρησιμοποιεί τα εξής πακέτα:

- Vvrpywork
- Open3D
- Trimesh
- Random
- NumPy
- Time

Η εφαρμογή αποτελείται από τα αρχεία main.py και το utility.py. Στο utility.py περιέχονται διάφορες βοηθητικές συναρτήσεις και το main αρχείο περιέχει την UavSim κλάση που αποτελεί όλη την εφαρμογή.

Στην βιβλιοθήκη Vvrywork έχουν γίνει μερικές προσθήκες που θα αναλυθούν παρακάτω.

Στο αρχείο shapes.py:

Στην κλάση Cuboid3D προστέθηκε η μεταβλητή self.height για να υπάρχει αποθηκευμένο το ύψος του Cuboid ανά πάσα στιγμή. Επίσης προστέθηκαν οι παρακάτω μέθοδοι:

```
# My adddition
def get_all_points(self, lst=False) -> List[Point3D]|List[NDArray3]:···

# My addition
def get_all_lines(self) -> LineSet3D:···

# My addition
def check_point_in_cuboid(self, point) -> bool:···

# My addition
def check_mesh_in_cuboid(self, mesh:Mesh3D) -> bool:···

# My addition
def get_center(self, lst=True) -> Point3D|NDArray:···

# My addition
def get_face_centers(self, lst=True) -> List[Point3D]|List[NDArray3]:···
```

Η get_all_points() επιστρέφει όλα τα σημεία του Cuboid είτε ως Point3D αντικείμενα είτε ως numpy arrays.

Η get_all_lines() επιστρέφει ένα Lineset3D με όλες τις γραμμές που φτιάχνουν το Cuboid.

Η check_mesh_in_cuboid() ελέγχει αν κάποιο συγκεκριμένο Mesh3D αντικείμενο ανήκει μέσα στα όρια που ορίζει το Cuboid.

Η get_center() επιστρέφει το κέντρο του Cuboid είτε ως Point3D αντικείμενο είτε ως numpy array.

Η get_face_centers() επιστρέφει τα κέντρα των 6 επιπέδων του Cuboid είτε ως Point3D αντικείμενα είτε ως numpy arrays.

Στην κλάση Mesh3D προστέθηκε η μεταβλητή self.path για να υπάρχει αποθηκευμένο το path του 3Δ μοντέλου ανά πάσα στιγμή. Επίσης προστέθηκαν οι παρακάτω μέθοδοι:

```
# My addition
def get_center(self, lst=True) -> Point3D|NDArray: ···

# My addition
def get_copy(self) -> Mesh3D: ···
```

Η get_center() επιστρέφει το κέντρο του Mesh είτε ως Point3D αντικείμενο είτε ως numpy array.

Η get_copy() επιστρέφει ένα αντίγραφο του Mesh.

Στο αρχείο scene.py:

Στην Scene3D κλάση προστέθηκε η change_camera() μέθοδος που μετακινεί την κάμερα σε ένα προκαθορισμένο σημείο.

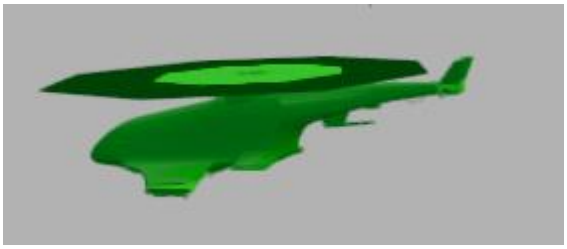
```
# My addition
def change_camera(self, new_center): ···
```

2.2 3Δ Μοντέλα

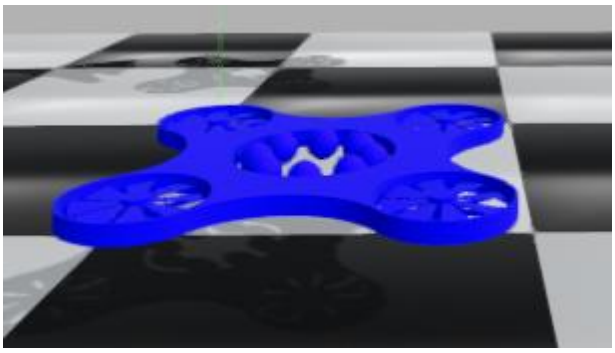
Προτείνεται να χρησιμοποιούνται μόνο τα πρώτα δύο μοντέλα επειδή έχουν μικρότερη γεωμετρική πολυπλοκότητα

- F52.obj:

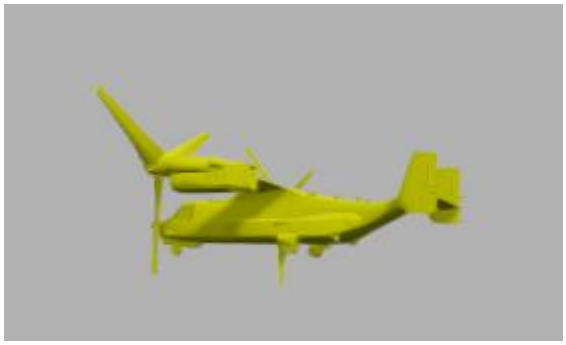
- Helicopter.obj:



- quadcopter_scifi.obj:



- v22_osprey:



3. Ερωτήματα Μέρος Α

3.1 Ερώτημα 1 (Οπτικοποίηση)

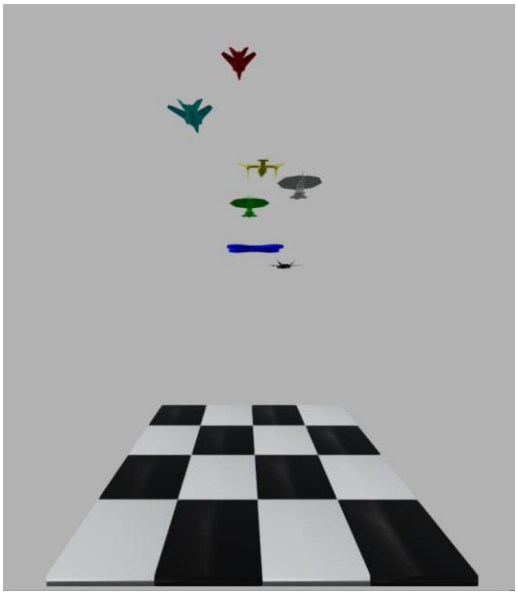
Για το ερώτημα αυτό αναπτύχθηκαν οι εξής μέθοδοι:

```
# SETTING UP THE SCENE
def landing_pad(self, size:float, height:float = 0.2) -> None:···

def show_drones(self, num_drones:int = 10, rand_rot:bool = True,
                singular:bool = False, label:bool = False) -> None:···

def randomize_mesh(self, mesh: Mesh3D, drone_id:int, trans_thresold:float = 2.0,
                   label:bool = False,
                   rand_rot:bool = True) -> Mesh3D:···
```

- **landing_pad()**: Δημιουργεί μια επιφάνεια προσγείωσης με NxN θέσεις
- **randomize_mesh()**: Περικλείνει το mesh στην μοναδιαία σφαίρα και αλλάζει τυχαία την θέση ή/και τον προσανατολισμό του
- **show_drones()**: Οπτικοποιεί ένα συγκεκριμένο αριθμό drones σε τυχαίες θέσεις ή/και με προσανατολισμούς

Η **landing_pad()** τρέχει στον constructor της κλάσης **UavSim** και αναλόγως αν πατήσουμε S ή P εκτελείται η **show_drones()** και οπτικοποιούνται τα drones σε τυχαίες θέσεις ή/και με τυχαίους προσανατολισμούς αντίστοιχα

Εικόνα 2: Οπτικοποίηση drones σε τυχαίες θέσεις και με τυχαίους προσανατολισμούς



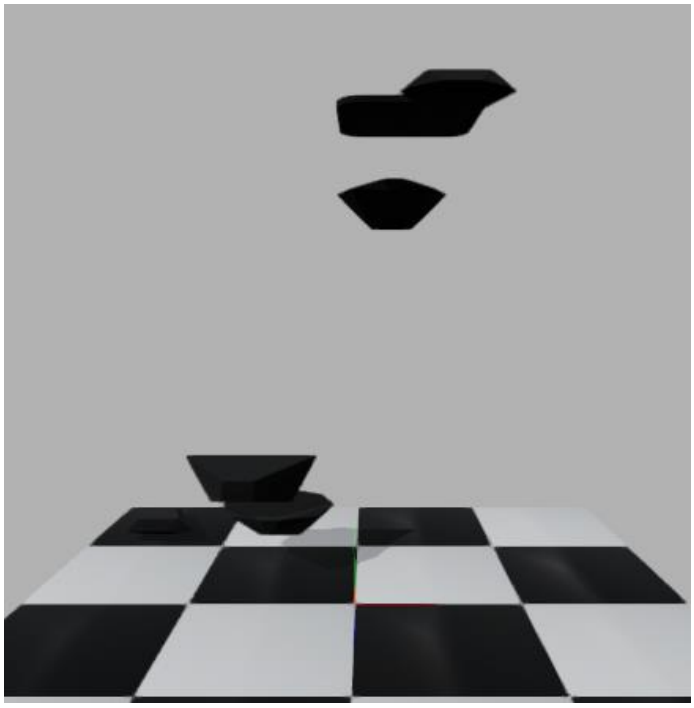
Εικόνα 1: Οπτικοποίηση drones σε τυχαίες θέσεις

3.2 Ερώτημα 2 (Δημιουργία AABB, Convex Hull, 14-DOP)

Για το ερώτημα αυτό αναπτύχθηκαν οι εξής μέθοδοι:

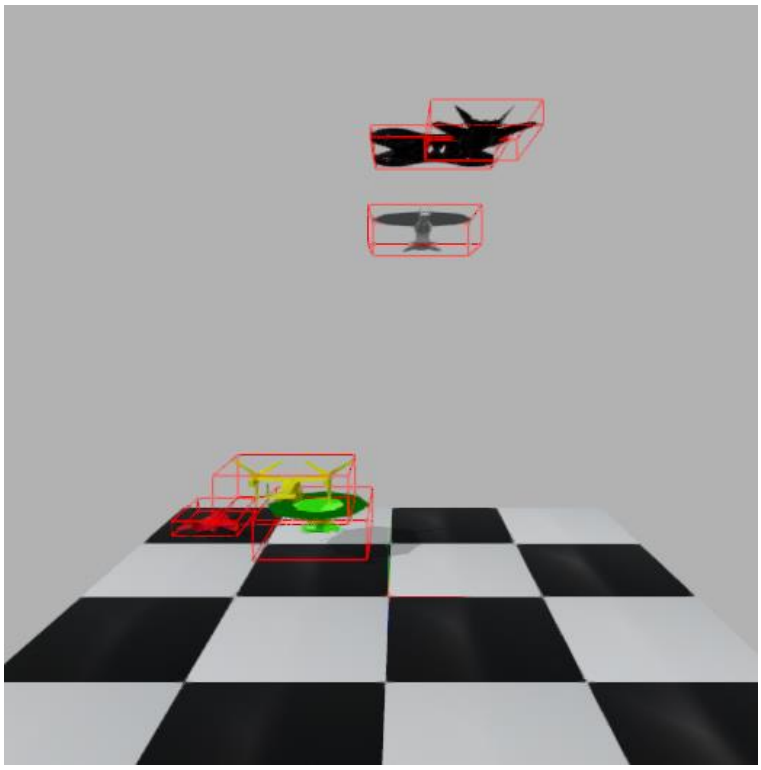
```
# VOLUMES' FUNCTIONS(Convex Hulls, AABBs, 14-DOPs)
def get_convex_hull(self, mesh:Mesh3D, mesh_name:str) -> Mesh3D:

def show_convex_hull(self, mesh:Mesh3D, mesh_name:str) -> None:···

def get_aabb(self, mesh:Mesh3D, mesh_name:str) -> Cuboid3D:···

def show_aabb(self, mesh:Mesh3D, mesh_name:str) -> None:···

def get_14dop(self, mesh:Mesh3D, mesh_name:str) -> Mesh3D:···

def show_14dop(self, mesh:Mesh3D, mesh_name:str) -> None:···
```

- **get_convex_hull()**: Επιστρέφει το convex hull του Mesh χρησιμοποιώντας την Open3D βιβλιοθήκη
- **show_convex_hull()**: Απεικονίζει και αποθηκεύει σε ένα λεξικό το convex hull



Εικόνα 3: Δημιουργία Convex Hulls

- **get_aabb()**: Επιστρέφει το AABB του Mesh βρίσκοντας τα vertices με τις ελάχιστες και μέγιστες συντεταγμένες σε κάθε άξονα και χρησιμοποιώντας ένα Cuboid3D δημιουργείται το AABB.
- **show_aabb()**: Απεικονίζει και αποθηκεύει σε ένα λεξικό το AABB



Εικόνα 4: Δημιουργία AABBs

- **get_14dop()**: Επιστρέφει το 14-DOP ενός mesh το οποίο υπολογίζεται με τον εξής τρόπο:

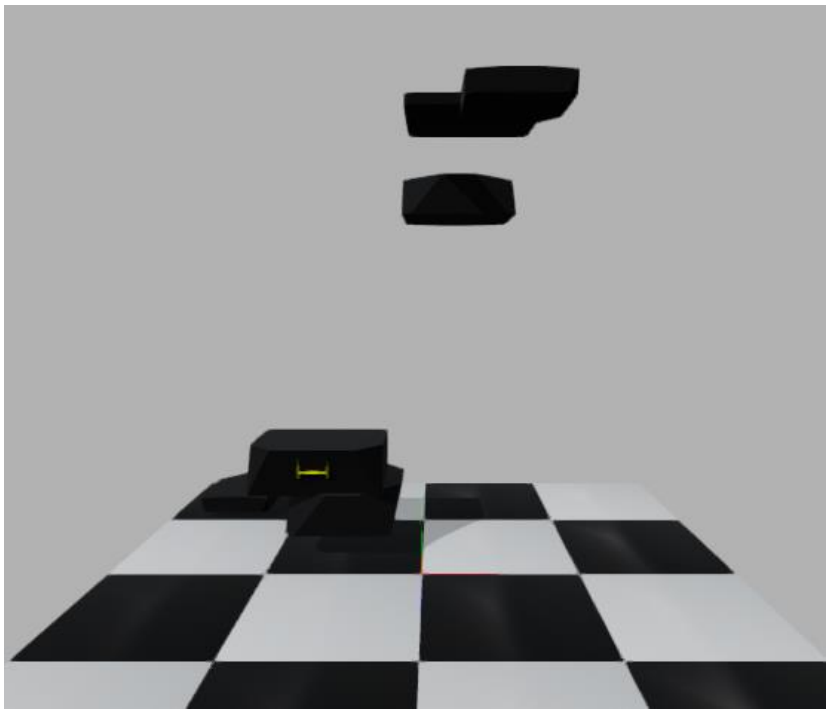
  Αρχικά βρίσκουμε τα vertices με τις μέγιστες και ελάχιστες συντεταγμένες σε κάθε μια από αυτές 14 κατευθύνσεις:

```
# 14 directions for the 14-DOP
DIRECTIONS = np.array([
    [1, 0, 0],  # x-axis
    [0, 1, 0],  # y-axis
    [0, 0, 1],  # z-axis

    [-1, 0, 0], # -x-axis
    [0, -1, 0], # -y-axis
    [0, 0, -1], # -z-axis

    [1, 1, 1],  # diagonals
    [-1, 1, 1],
    [-1, 1, -1],
    [1, 1, -1],

    [-1, -1, -1],
    [1, -1, -1],
    [1, -1, 1],
    [-1, -1, 1]

])
```

Για κάθε μία από τις κατευθύνσεις, προβάλουμε όλα τα vertices του 3Δ mesh στην κατεύθυνση αυτού του διανύσματος

Για κάθε τέτοια κατεύθυνση βρίσκουμε την μέγιστη και ελάχιστη τιμή αυτών των προβολών.
Αυτές οι τιμές καθορίζουν τις θέσεις των planes που θα περικλείσουν το mesh. Αφού βρούμε τις εξισώσεις αυτών των planes, έπειτα βρίσκουμε τις τομές των planes με τις διαγώνιες κατευθύνσεις με τις γραμμές του AABB του mesh. Παίρνοντας το convex hull αυτών των σημείων τομής υπολογίζουμε το 14-DOP. Ο τρόπος αυτός αποτελεί μια καλή προσέγγιση του περιβάλλοντος όγκου του mesh όμως συχνά εμφανίζει λάθη και το 14-DOP που επιστρέφει δεν περικλείει με ικανοποιητικό τρόπο το mesh.

- **show_14dop()**: Απεικονίζει και αποθηκεύει σε ένα λεξικό το 14-DOP



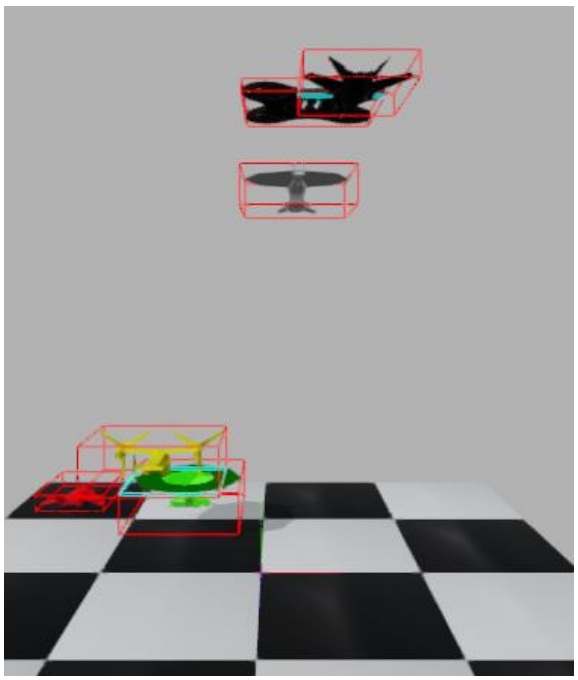
Εικόνα 5: Δημιουργία 14-DOPs

3.3 Ερώτημα 3 (Έλεγχος συγκρούσεων με AABBs και Convex Hulls)

Για το ερώτημα αυτό αναπτύχθηκαν οι εξής μέθοδοι:

```
# COLLISION DETECTION FUNCTIONS
def collision_detection_aabbs(self, mesh1:Mesh3D, mesh_name1:str,
                              mesh2:Mesh3D, mesh_name2:str, vis:bool = True) -> bool:

def collision_detection_chs(self, mesh1:Mesh3D, mesh_name1:str,
                            mesh2:Mesh3D, mesh_name2:str) -> bool: ...
```

- **collision_detection_aabbs()**: Έλεγχος συγκρούσεων βάσει των AABBs και προαιρετική απεικόνιση του AABB τομής. Αν υπάρχουν συγκρούσεις τυπώνονται στο διάλογο της σκηνής. Ο έλεγχος γίνεται με την συνάρτηση **intersect_cuboids()** του utility.py συγκρίνοντας τις μέγιστες και ελάχιστες συντεταγμένες των δύο AABBs**.** Αυτή η μέθοδος μας δίνει τα λιγότερα ακριβή αποτελέσματα όμως με την πιο υψηλή ταχύτητα.
- **collision_detection_chs()**: Έλεγχος συγκρούσεων βάσει των convex_hulls χρησιμοποιώντας την Trimesh βιβλιοθήκη. Αν υπάρχουν συγκρούσεις τυπώνονται στο διάλογο της σκηνής. Αυτή η μέθοδος μας δίνει τα αρκετά ακριβή αποτελέσματα όμως με σχετική αργή ταχύτητα επειδή και ελέγχουμε δύο αυτούσια meshes, όμως με σημαντικά πιο μικρή γεωμετρική πολυπλοκότητα από τα αρχικά meshes.



Εικόνα 6: Έλεγχος Σύγκρουσης με AABBs και απεικόνιση AABB τομής

3.4 Ερώτημα 4, 5 (Έλεγχος συγκρούσεων με 14-DOPs και τα ίδια τα Meshes και Απεικόνιση σημείων Σύγκρουσης)

Για το ερώτημα αυτό αναπτύχθηκαν οι εξής μέθοδοι:

```
def collision_detection_kdops(self, mesh1:Mesh3D, mesh_name1:str,
                              mesh2:Mesh3D, mesh_name2:str) -> bool:···

def collision_detection_meshes(self, mesh1:Mesh3D, mesh_name1:str,
                               mesh2:Mesh3D, mesh_name2:str, vis:bool=True) -> bool:
```

- **collision_detection_kdops()**: Αυτή η μέθοδος ελέγχει αν δύο 3Δ meshes (**mesh1** και **mesh2**) τέμνονται (συγκρούονται) βάσει των 14-DOP περιγραφών τους. Επιστρέφει **True** εάν ανιχνεύεται σύγκρουση και **False** εάν όχι. Πρώτα, λαμβάνουμε τις μέγιστες και ελάχιστες προβολές του 14-DOP στις προκαθορισμένες 14 κατευθύνσεις. Στην συνέχεια, ελέγχει αν υπάρχει κάποιος διαχωρισμός κατά μήκος κάθε μιας από τις 7 κατευθύνσεις. Το βασικό εδώ είναι το **Θεώρημα Διαχωριστικών Αξόνων (Separating Axis Theorem - SAT)**. Το SAT αναφέρει ότι δύο κυρτά αντικείμενα δεν τέμνονται αν υπάρχει τουλάχιστον ένας άξονας κατά μήκος του οποίου είναι εντελώς διαχωρισμένα.
    - **Έλεγχος για διαχωρισμό**: Για κάθε άξονα , ελέγχει αν η μέγιστη προβολή του πρώτου πλέγματος ή αν η μέγιστη προβολή του δεύτερου πλέγματος. Εάν αυτό ισχύει για κάποιον άξονα, τότε τα meshes δεν τέμνονται.
    - Αν διαχωρίζονται, επίσης δεν τέμνονται.
    - Αν δεν βρεθεί διαχωρισμός σε κανέναν άξονα, τότε τα meshes τέμνονται.

  Αν υπάρχουν συγκρούσεις τυπώνονται στο διάλογο της σκηνής. Αυτή η μέθοδος μας δίνει ακριβή αποτελέσματα όπως και τα convex hulls αλλά με καλύτερη ταχύτητα από την μέθοδο με τα convex hulls επειδή δεν ελέγχουμε αυτούσια δυο meshes.

- **collision_detection_meshes()**: Έλεγχος συγκρούσεων βάσει των meshes χρησιμοποιώντας την Trimesh βιβλιοθήκη και προαιρετικά απεικόνιση των σημείων τομής. Αν υπάρχουν συγκρούσεις τυπώνονται στο διάλογο της σκηνής. Να σημειωθεί ότι αυτή ή μέθοδος ελέγχει πρώτα όλους τους άλλους μεθόδους σύγκρουσης και μετά συγκρίνει αυτούσια τα meshes. Έτσι μπορούμε να απορρίψουμε περιπτώσεις όχι σύγκρουσης πολύ πιο γρήγορα.

Εικόνα 7: Απεικόνιση σημείων σύγκρουσης

4. Ερωτήματα Μέρος Β(Προσομοίωση)

Σε όλα τα παρακάτω ερωτήματα γίνεται η χρήση των εξής μεθόδων:

```
# SIMULATION FUNCTIONS
def on_idle(self): ···

def move_drone(self, mesh:Mesh3D, mesh_name:str, ···

def move_drone_to_point(self, mesh:Mesh3D, mesh_name:str, ···
```

- **on_idle()**: Αυτή η μέθοδος τρέχει πάντα και αναλόγως διαφόρων flag παύσης τρέχει την εκάστοτε προσομοίωση του κάθε ερωτήματος. Το time step και τα flags έχουν υλοποιηθεί με τον εξής τρόπο:

```
def on_idle(self):
    '''The idle function of the scene.'''

    # Time check to update the scene
    if time.time() - self.last_update < self.dt:
        return

    # Simulation with the drones moving and stopping if they collide
    if not self.paused: ···

    # Simulation with the drones moving and changing their speed to avoid collisions
    if not self.paused_no_collisions: ···

    # Landing Protocol
    if not self.pause_landing_simulation: ···

    return False
```

  Η τιμή της last_update ανανεώνεται κάθε φορά που κουνιούνται τα drones
- **move_drone()**: Μετακινεί ένα drone με συγκεκριμένη ταχύτητα καθώς και τις προβολές του
- **move_drone_to_point()**: Μετακινεί ένα drone σε ένα συγκεκριμένο σημείο καθώς και τις προβολές του

4.1 Ερώτημα 6 (Προσομοίωση κίνησης και Έλεγχος Σύγκρουσης σε κάθε frame)

Για το ερώτημα αυτό αναπτύχθηκε η εξής μέθοδος:

```python
def simulate(self):
    """Simulate the scene, moving the drones and stopping them if they collide."""

    # Remove any aabbs, chs, kdops and labels from the scene
    self.clear_attributes(col_points=False)

    for mesh_name, mesh in self.meshes.items():

        # If all the drones have zero speeds, pause the simulation
        if all(np.array_equal(speed, np.array([0, 0, 0])) for speed in self.speeds.values()):···

        # If all drones are out of bounds, reset the scene
        if self.in_bounds == {}:···

        # If the drone is out of bounds, remove it
        if not self.bounding_cuboid.check_mesh_in_cuboid(mesh):···

        # Move the drone if it is in bounds and its speed is not zero
        if mesh_name in self.in_bounds and not np.array_equal(self.speeds[mesh_name], np.array([0, 0, 0])):
            self.move_drone(mesh, mesh_name, self.speeds[mesh_name])

        # Check and visualize the collisions
        for mesh_name2, mesh2 in self.meshes.items():···
```

Αυτή η μέθοδος καλείται σε κάθε frame από την **on_idle()** όταν το αντίστοιχο flag παύσης είναι ανενεργό. Σε κάθε frame:

- Καθαρίζουμε την σκηνή από οποιαδήποτε AABBs, Convex Hulls, 14-DOPs αν υπάρχουν
- Ελέγχουμε αν όλα τα drones είναι ακίνητα (έχουν σταματήσει) και τότε κάνουμε παύση την προσομοίωση
- Αν όλα τα drones έχουν βγει εκτός προκαθορισμένων ορίων τότε καθαρίζουμε την σκήνη και spawnάρουμε πάλι drones
- Αν κάποιο drone είναι εκτός ορίων, το βγάζουμε από την σκηνή
- Αν το drone είναι εντός ορίων και δεν έχει ταχύτητα μηδενική το μετακινούμε με την ταχύτητα του. Να σημειωθεί ότι η αρχική ταχύτητα των drones είναι μια αυθαίρετη τιμή για κάθε 3Δ μοντέλο και αλλάζει υπό περιπτώσεις που θα δούμε στη συνέχεια(σύγκρουση, πρωτόκολλο προσγείωσης)
- Ελέγχουμε αν το drone συγκρούεται με κάποιο άλλο drone στην σκηνή και τα σταματάμε και τα δύο. Αν και τα δύο drones είναι σταματημένα δεν γίνεται έλεγχος σύγκρουσης

4.2 Ερώτημα 7 (Ανανέωση ταχυτήτων για αποφυγή συγκρούσεων)

Για το ερώτημα αυτό αναπτύχθηκε η εξής μέθοδος:

```
def simulate_no_collisions(self):
    """Simulate the scene , moving the dornes and changing their speed if they collide to avoid collisions."""

    # Remove any aabbs, chs, kdops, collision points and labels from the scene
    self.clear_attributes()

    for mesh_name1, mesh1 in self.meshes.items():

        # If all the drones are out of bounds, reset the scene
        if self.in_bounds == {}: ···

        # If the drone is out of bounds, remove it
        if not self.bounding_cuboid.check_mesh_in_cuboid(mesh1): ···

        # Move the drone if it is in bounds
        if mesh_name1 in self.in_bounds: ···

        # Check for collisions and adjust the speed to avoid them
        for mesh_name2, mesh2 in self.meshes.items(): ···
```

Αυτή η μέθοδος καλείται σε κάθε frame από την **on_idle()** όταν το αντίστοιχο flag παύσης είναι ανενεργό. Σε κάθε frame:

- Καθαρίζουμε την σκηνή από οποιαδήποτε AABBs, Convex Hulls, 14-DOPs και σημείων σύγκρουσης αν υπάρχουν
- Αν όλα τα drones έχουν βγει εκτός προκαθορισμένων ορίων τότε καθαρίζουμε την σκήνη και spawnάρουμε πάλι drones
- Αν κάποιο drone είναι εκτός ορίων, το βγάζουμε από την σκηνή
- Αν το drone είναι εντός ορίων το μετακινούμε με την ταχύτητα του.
- Ελέγχουμε αν το drone συγκρούεται με κάποιο άλλο drone στην σκηνή και ανανεώνουμε τις ταχύτητες τους ώστε να αποφευχθεί η σύγκρουση. Αναλυτικότερα, ελέγχουμε το σύγκρουση μεταξύ ενός drone και ενός άλλου στη θέση που θα ήταν στα πέντε επόμενα frames. Αν ελέγχαμε σύγκρουση στο τωρινό frame θα γινόντουσαν πάρα πολλές ενδιάμεσες συγκρούσεις μέχρι η ταχύτητα να ανανεωθεί επαρκώς για να μην αγγίζουν καθόλου τα meshes και με αυτόν τον τρόπο αποφεύγονται πολλά crashes. Η ταχύτητα των drones ανανεώνεται με βάση το surface normal ενός σημείου σύγκρουσης, συγκεκριμένα το πρώτο που βρίσκεται από την Trimesh βιβλιοθήκη. Ο τύπος που χρησιμοποιείται είναι ο εξής και έχει ως αποτέλεσμα τα δυο meshes να «γλιστρούν» ανά μεταξύ τους:

$$v_{new} = v_{old} - 2(v_{old} \cdot n)n$$

```
# Change the speed of the drones to avoid collisions(reflect the mesh's velocity vector across the collision surface normal)
new_speed1 = self.speeds[mesh_name1]  - 2 * np.dot(self.speeds[mesh_name1], surface_normal) * surface_normal
self.speeds[mesh_name1] = new_speed1
```

Όπου n είναι το surface normal στο επιλεγμένο σημείο σύγκρουσης. Αυτό προβάλει την ταχύτητα πάνω στην εφαπτομένη της επιφάνειας σύγκρουσης, κάτι που θα προκαλέσει το αντικείμενο να "γλιστράει" κατά μήκος της επιφάνειας αντί να αναπηδά.

4.3 Ερώτημα 8 (Πρωτόκολλο προσγείωσης-απογείωσης χωρίς συγκρούσεις)

Για το ερώτημα αυτό αναπτύχθηκε η εξής μέθοδος:

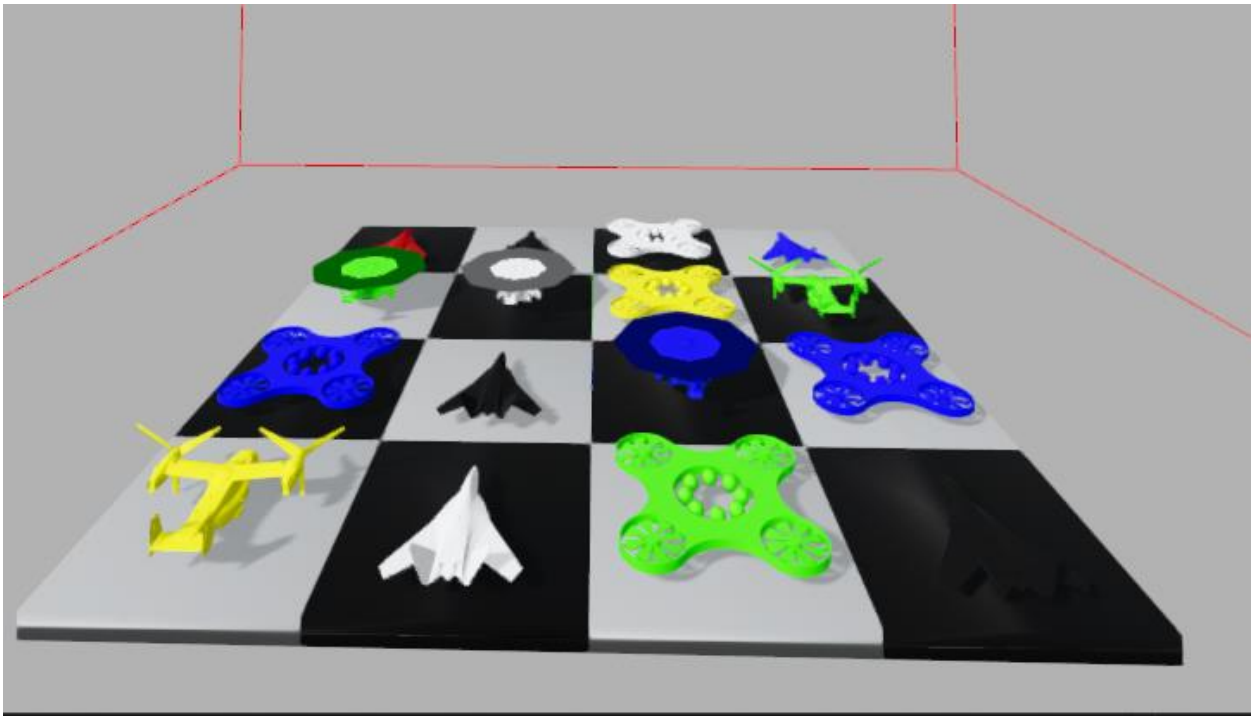
```python
def landing_protocol(self):
    '''Simulate the landing protocol.'''

    # Remove any aabbs, chs, kdops, collision points and labels from the scene
    self.clear_attributes()

    # Spawn drones at random time intervals until there are N^2 drones (one drone per landing pad)
    if not len(self.meshes) == self.N**2: ···

    for i, (mesh_name1, mesh1) in enumerate(self.meshes.items()):

        # Select the landing pad for the drone
        landing_pad = list(self.landing_pads.values())[i]

        # Adjust the landing point so that the drone is above the landing pad, not inside it
        landing_point = landing_pad.get_center() + np.array([0, 0.3, 0])

        # Calculate the distance between the drone and the landing pad
        distance = np.linalg.norm(mesh1.get_center() - landing_point)

        # Move the drone to the direction of the landing pad
        if distance > 0.1: ···

        # If the drone is close to the landing pad, move it to the landing pad
        if distance <= 0.1: ···
```

Αυτή η μέθοδος καλείται σε κάθε frame από την **on_idle()** όταν το αντίστοιχο flag παύσης είναι ανενεργό. Σε κάθε frame:

- Καθαρίζουμε την σκηνή από οποιαδήποτε AABBs, Convex Hulls, 14-DOPs και σημείων σύγκρουσης αν υπάρχουν
- Αν ο αριθμός των drones στην σκηνή είναι μικρότερος από τον μέγιστο που χωράει η επιφάνεια προσγείωσης τότε εμφανίζουν σε τυχαία στιγμιότυπα drones μέχρι να φτάσουμε το όριο
- Το πρωτόκολλο προσγείωσης λειτουργεί βρίσκοντας την απόσταση του drone από την καθορισμένη επιφάνεια προσγείωσης του. Αν η απόσταση αυτή είναι πολύ μικρή τότε χρησιμοποιώντας την **move_drone_to_point()** προσγειώνουμε το drone και έπειτα

ακινητοποιείται. Αλλιώς, μετακινούμε το drone με τυχαία ταχύτητα και κατεύθυνση προς την επιφάνεια προσγείωσης. Αν κατά την διάρκεια αυτής της κίνησης προκύψει σύγκρουση τότε ανανεώνουμε την ταχύτητα του drone με βάση το προηγούμενο ερώτημα και το μετακινούμε για τα επόμενα 3 frames με την ταχύτητα αποφυγής. Μετά από αυτά τα 3 frames χρησιμοποιούμε μια τυχαία ταχύτητα και με κατεύθυνση προς την επιφάνεια προσγείωσης. Να σημειωθεί ότι και εδώ γίνεται έλεγχος σύγκρουσης μεταξύ ενός drone και ενός άλλου στη θέση που θα ήταν στα πέντε επόμενα frames για τον λόγο που αναφέρθηκε παραπάνω.

- Δεν αναπτύχθηκε κάποιο συγκεκριμένο πρωτόκολλο απογείωσης και χρησιμοποιείται η μέθοδος του ερωτήματος 7.



Εικόνα 8: Όλα τα drones προσγειωμένα

4.4 Ερώτημα 9 (Απεικόνιση Προσομοίωσης και Στατιστικά στοιχεία)

Η απεικόνιση γίνεται μέσα στις μεθόδους που αναφέρθηκαν παραπάνω και τα στατιστικά στοιχεία του μπορεί να δει ο χρήστης είναι τα εξής:

```python
# STATS FUNCTION
def stats(self):
    """Print the statistics of the scene."""

    # Print the statistics of the scene
    self.print("----------------------------------------")
    self.print("Statistics:")
    self.print("----------------------------------------")
    self.print(f"-Number of drones: {len(self.meshes)}")
    self.print(f"---Time to create the Convex Hulls: {np.sum(list(self.ch_t.values())):.4f} seconds")
    self.print(f"---Time to create the AABBs: {np.sum(list(self.aabb_t.values())):.4f} seconds")
    self.print(f"---Time to create the 14-DOPs: {np.sum(list(self.kdop_t.values())):.4f} seconds")
    self.print(f"---Time to create the Projections: {np.sum(list(self.projections_t.values())):.4f} seconds")
    self.print(f"---Time to check for AABB collisions: {np.sum(list(self.aabb_col_t.values())):.4f} seconds")
    self.print(f"---Time to check for Convex Hull collisions: {np.sum(list(self.ch_col_t.values())):.4f} seconds")
    self.print(f"---Time to check for 14-DOP collisions: {np.sum(list(self.kdop_col_t.values())):.4f} seconds")
    self.print(f"---Time to check for Mesh3D collisions: {np.sum(list(self.mesh_col_t.values())):.4f} seconds")
    self.print(f"---Time to check for Projections collisions: {np.sum(list(self.projections_col_t.values())):.4f} seconds")
    self.print("----------------------------------------")
```

4.5 Extra Ερώτημα (Απεικόνιση προβολών στα xy, xz, yz επίπεδα σε κάθε frame και Έλεγχος Συγκρούσεων βάσει αυτών)

Για το ερώτημα αυτό αναπτύχθηκαν οι εξής μέθοδοι:

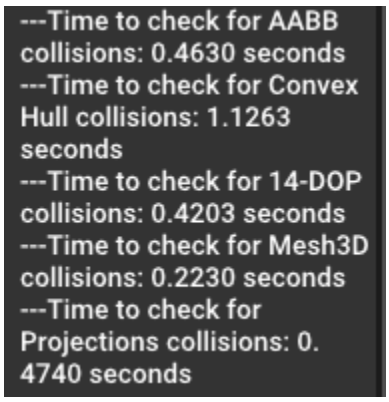
```
# PROJECTIONS FUNCTIONS
def get_projections(self, mesh:Mesh3D, mesh_name:str) -> tuple[Mesh3D, Mesh3D, Mesh3D]: ···

def show_projections(self, mesh:Mesh3D, mesh_name:str) -> None: ···

def collision_detection_projections(self, mesh1:Mesh3D, mesh_name1:str, ···
```

- **get_projections()**: Επιστρέφει τις προβολές στα xy, xz, yz επίπεδα ενός mesh με την **get_projection()** της utility.py και τις μετακινεί στα άκρα των ορίων της σκηνής:

```
def get_projection(mesh:Mesh3D, plane:str) -> Mesh3D:
    """Get the projection of a mesh on a plane.

    Args:
        mesh : The mesh
        plane : The plane of projection ("xy", "xz", "yz")

    Returns:
        proj_mesh : The projected mesh
    """
    vertices = np.array(mesh.vertices)
    v = vertices.copy()

    if plane == "xy":
        v[:, 2] = 0
    elif plane == "xz":
        v[:, 1] = 0
    elif plane == "yz":
        v[:, 0] = 0

    proj_mesh = Mesh3D(color=mesh.color)
    proj_mesh.vertices = v
    proj_mesh.triangles = mesh.triangles

    # Remove duplicated and unreferenced vertices
    proj_mesh.remove_duplicated_vertices()
    proj_mesh.remove_unreferenced_vertices()

    return proj_mesh
```

- **show_projections()**: Απεικονίζει τις προβολές

- **collision_detection_projections()**: Έλεγχος συγκρούσεων με βάση τις προβολές. Ο τρόπος που ελέγχονται οι συγκρούσεις είναι παίρνοντας τα AABB των 2Δ προβολών και ελέγχοντας αν τέμνονται στο εκάστοτε επίπεδο. Αν δεν έχουμε τομή σε κάποιο από τα τρία επίπεδα προβολής τότε δεν έχουμε σύγκρουση μεταξύ των mesh. Αν έχουμε τομή και στα τρία επίπεδα προβολής, τότε ίσως έχουμε σύγκρουση. Αυτό εξαρτάται από το 3Δ μοντέλο, όμως αποδείχθηκε δοκιμαστικά ότι τις περισσότερες φορές δίνει σωστό αποτέλεσμα. Αύτη η μέθοδος σύγκρουσης όμως χρησιμοποιεί την μέθοδο σύγκρουσης μεταξύ AABBs τρεις φορές. Η ταχύτητα του είναι καλύτερη από όλες τις άλλες μεθόδους πέρα από αυτή με τα AABBs προφανώς αλλά η αξιοπιστία του υστερεί.



5 Οδηγίες Εγκατάστασης

Απαιτήσεις

- Έκδοση Python 3.10.13 ή μεταγενέστερη
- Conda (προαιρετικό)

Εγκατάσταση κώδικα

Clone το repo ή αποσυμπίεση του .zip:

- git clone https://github.com/Papiqulos/Project_3D_UAV_2024

Δημιουργία venv (προτείνεται το conda)

- conda create --name drone-sim python=3.10.13
- conda activate drone-sim

Εγκατάσταση βιβλιοθηκών

- pip install -r requirements.txt

6 Οδηγίες Χρήσης

Εκτέλεση της Εφαρμογής

Αφού ολοκληρωθεί η ρύθμιση, μπορείτε να εκτελέσετε την εφαρμογή προσομοίωσης drone. Παρακάτω αναφέρονται οι διαθέσιμες εντολές που λειτουργούν μέσα στην εφαρμογή:

Κύριες Εντολές

- R: Εκκαθάριση σκηνής
- B: Εναλλαγή εμφάνισης ορίων
- S: Εμφάνιση drones σε τυχαίες θέσεις
- P: Εμφάνιση drones σε τυχαίες θέσεις και προσανατολισμούς
- C: Εναλλαγή εμφάνισης κυρτών περιβλημάτων
- A: Εναλλαγή εμφάνισης AABBs
- K: Εναλλαγή εμφάνισης k-DOPs
- I: Εναλλαγή εμφάνισης προβολών (xy, xz, yz)
- N: Έλεγχος συγκρούσεων (AABBs)

- L: Έλεγχος συγκρούσεων (Κυρτά Περιβλήματα)
- M: Έλεγχος συγκρούσεων (14-DOPs)
- V: Έλεγχος συγκρούσεων και εμφάνιση σημείων σύγκρουσης (Mesh3Ds)
- O: Έλεγχος συγκρούσεων (Προβολές)
- T: Προσομοίωση
- F: Προσομοίωση χωρίς συγκρούσεις
- Q: Εμφάνιση στατιστικών
- SPACE: Ενεργοποίηση πρωτοκόλλου προσγείωσης

7 Βιβλιογραφία

- Open3D: https://www.open3d.org/docs/release/
- Vvrpywork: Εργαστήριο 3D, Τμήμα HMTY. Μία βιβλιοθήκη-wrapper για πολλές από τις λειτουργίες της Open3D.
- Trimesh: https://trimesh.org/
- Numpy: https://numpy.org/doc/
- https://www.realtimerendering.com/resources/GraphicsGems/gemsiii/rand_rotation.c
- Efficient Algorithms for Collision Avoidance at Intersections: https://dl.acm.org/doi/pdf/10.1145/2185632.2185656
- https://www.khronos.org/opengl/wiki/Calculating_a_Surface_Normal
- https://www.sciencedirect.com/science/article/abs/pii/S0094114X21000951