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 εστιάζει στην αναπαράσταση και προσομοίωση πολλαπλών μη επανδρωμένων αεροσκαφών (UAVs) σε τρισδιάστατο χώρο, με σκοπό την ανάπτυξη ενός ολοκληρωμένου συστήματος που θα επιτρέπει την ασφαλή και αποτελεσματική πλοήγηση τους. Συγκεκριμένα, η εργασία περιλαμβάνει τη δημιουργία διάφορων περιβαλλοντικών όγκων, οι οποίοι προσομοιώνουν ρεαλιστικά σενάρια πτήσης, καθώς και την κατασκευή 2D προβολών σε διάφορα επίπεδα για την καλύτερη οπτικοποίηση της κίνησης των UAVs.

Ένας από τους κεντρικούς στόχους της εργασίας είναι η ανίχνευση πιθανών συγκρούσεων σε πραγματικό χρόνο. Αυτό επιτυγχάνεται μέσω της εφαρμογής αλγορίθμων ανίχνευσης συγκρούσεων, που ελέγχουν την αλληλεπίδραση των UAVs με τα περιβαλλοντικά αντικείμενα, εντοπίζοντας τα σημεία τομής και τις πιθανές επικαλύψεις. Στη διαδικασία αυτή, εξετάζονται διάφορες μέθοδοι ανίχνευσης, όπως οι Axis-Aligned Bounding Boxes (AABBs), οι Convex Hulls και οι K-DOPs, προκειμένου να επιτευχθεί μια ισορροπία μεταξύ ταχύτητας και ακρίβειας.

Η μοντελοποίηση της κίνησης των UAVs γίνεται με προκαθορισμένες ταχύτητες, οι οποίες ανανεώνονται δυναμικά με βάση τις συνθήκες του περιβάλλοντος και τις ανιχνεύσεις συγκρούσεων. Σημαντική πτυχή της εργασίας είναι η δυνατότητα εντοπισμού συγκρούσεων σε συγκεκριμένα χρονικά διαστήματα, γεγονός που καθιστά την προσομοίωση πιο ρεαλιστική και ευέλικτη.

Για την αποφυγή των συγκρούσεων, αναπτύσσονται στρατηγικές κίνησης που βασίζονται σε αλγορίθμους πλοήγησης, οι οποίες επιτρέπουν στα UAVs να τροποποιούν τις διαδρομές τους, αποφεύγοντας εμπόδια και άλλες αεροπορικές κινήσεις. Αυτές οι στρατηγικές περιλαμβάνουν τη χρήση του surface normal σε σημεία σύγκρουσης και τη μελέτη πιθανών διαδρομών που δεν προκαλούν συγκρούσεις, εξασφαλίζοντας έτσι την ασφαλή πτήση των UAVs.

Ο τελικός στόχος της εργασίας είναι η δημιουργία πρωτοκόλλων πλοήγησης που θα επιτρέπουν στα UAVs να λειτουργούν αυτόνομα, ακολουθώντας ασφαλείς διαδρομές κατά την απογείωση και την προσγείωση. Με την ολοκλήρωση της εργασίας, αναμένεται η ανάπτυξη ενός ολοκληρωμένου συστήματος που θα μπορεί να αξιοποιηθεί σε πραγματικές εφαρμογές, όπως η παράδοση φορτίων, η παρακολούθηση περιβαλλοντικών αλλαγών και η ασφάλεια πτήσεων σε αστικά περιβάλλοντα.

2. Επισκόπηση της εφαρμογής

2.1 Δομή κώδικα

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

- Vvrpywork: Η βιβλιοθήκη του VVR εργαστηρίου του Πανεπιστημίου Πατρών. Χρησιμοποιείται για την απεικόνιση όλων των 3Δ γεωμετριών.
- Open3D: Μία βιβλιοθήκη για επεξεργασία 3D δεδομένων
- Trimesh: Για έλεγχο συγκρούσεων μεταξύ meshes
- Random: Παραγωγή τυχαίων αριθμών
- NumPy: Για γρήγορες πράξεις μεταξύ πινάκων
- Time: Για καταγραφή διαφόρων χρόνων εκτέλεσης λειτουργιών

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]: ···
```

Εικόνα 1: Προσθήκες στην Cuboid3D κλάση της Vvrpywork

- 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: ···
```

Εικόνα 2: Προσθήκες στην Mesh3D κλάση της Vvrpywork

- 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): ···
```

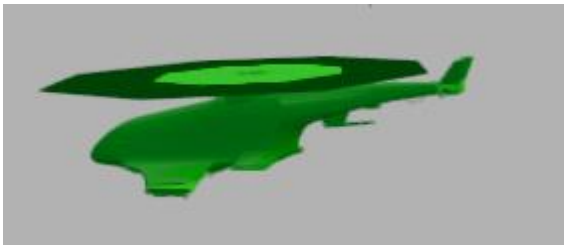
Εικόνα 3: Προσθήκη για αλλαγή θέσης κάμερας στην Scene3D κλάση της Vvrpywork βιβλιοθήκης

2.2 3Δ Μοντέλα

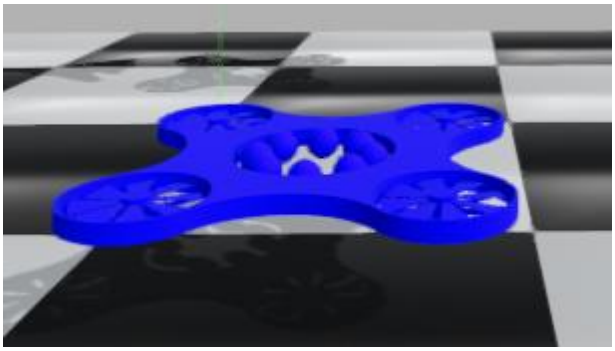
Προτείνεται να χρησιμοποιούνται μόνο τα πρώτα δύο μοντέλα επειδή έχουν μικρότερη γεωμετρική πολυπλοκότητα (λιγότερα τρίγωνα) και τρέχει πιο γρήγορα η προσομοίωση. Όλα τα ερωτήματα είναι λειτουργικά για όλα τα μοντέλα.

- F52.obj:

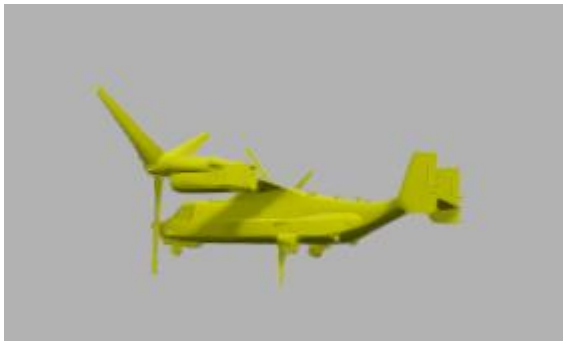
- Helicopter.obj:



- quadcopter_scifi.obj:



- v22_osprey:

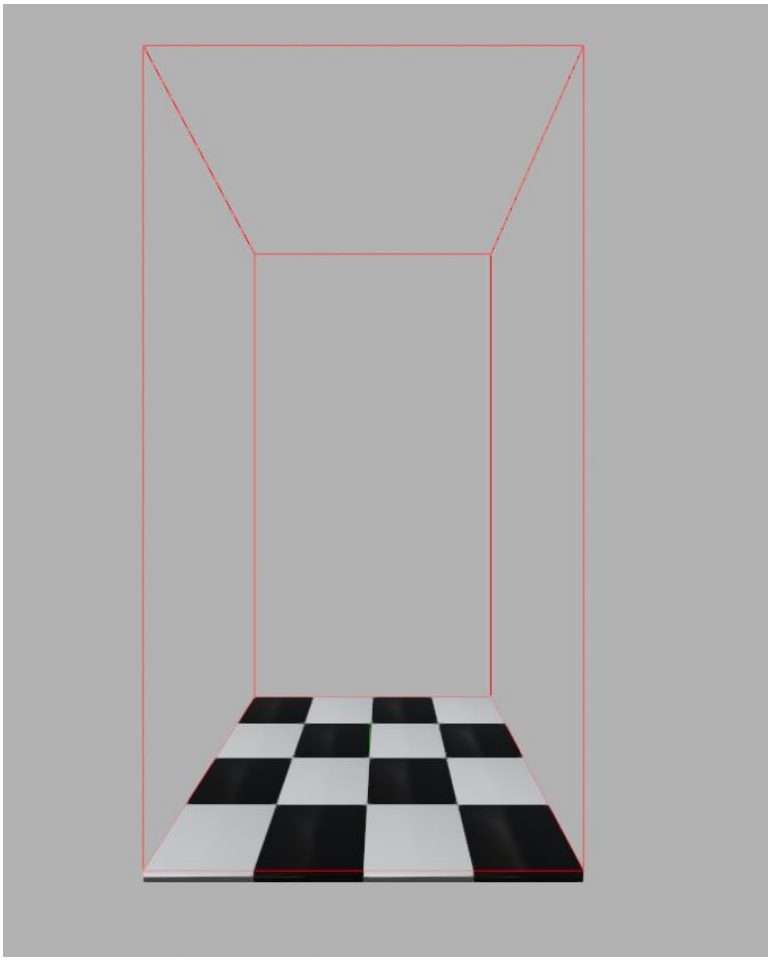


3. Ερωτήματα Μέρος Α

3.1 Ερώτημα 1 (Οπτικοποίηση)

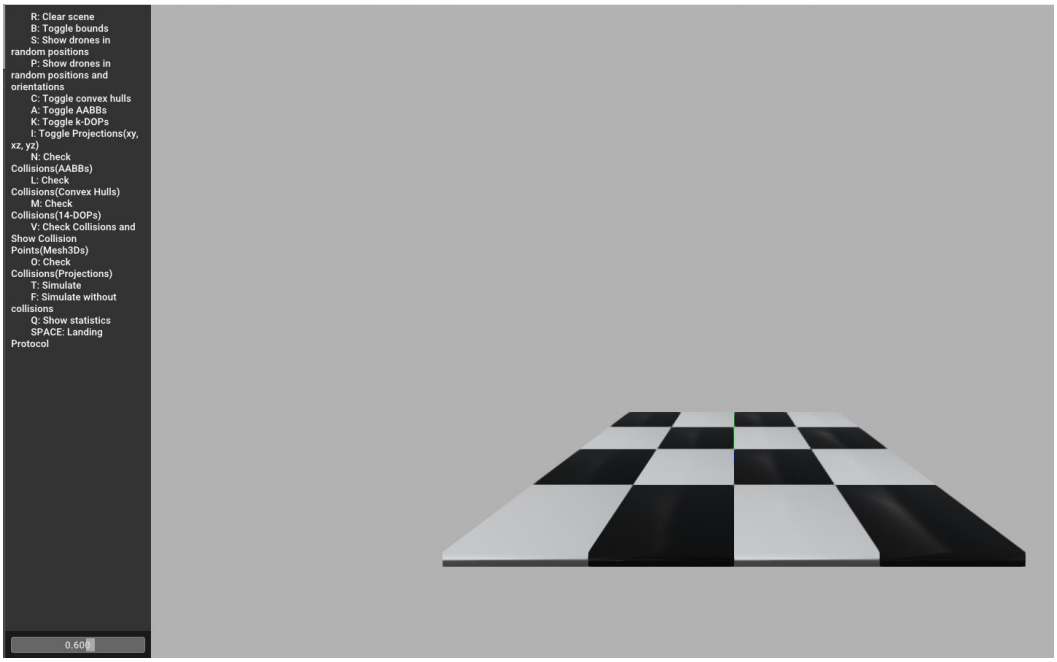
Έχοντας τα παραπάνω μοντέλα, στο παρών ερώτημα σκοπός είναι να τα απεικονίσουμε σε τυχαίες θέσεις καθώς και να φτιαχτεί η επιφάνεια προσγείωσης με NxN θέσεις. Επιλέχτηκε N=4 για λόγο που θα αναφερθεί παρακάτω.

Η σκηνή μας επιπλέον έχει κάποια προκαθορισμένα όρια, που μόνο μέσα σε αυτά μπορούν να εμφανίζονται και να κουνιούνται τα drones. Τα όρια φαίνονται παρακάτω:



Εικόνα 4: Το bounding Cuboid της σκηνής

Όταν ξεκινάει η εφαρμογή εμφανίζεται απευθείας η επιφάνεια προσγείωσης και αριστερά της σκηνής φαίνονται όλα τα διαθέσιμα commands στον «διάλογο» της σκηνής. Σε αυτόν τον διάλογο θα τυπώνονται και διάφορες πληροφορίες για το κάθε ένα από τα επόμενα ερωτήματα:



Εικόνα 5: Η εικόνα που βλέπει ο χρήστης όταν εκκινεί την εφαρμογή

Πατώντας το ***S*** ή το ***P*** εμφανίζεται ο αριθμός drones, με τυχαίο μοντέλο και τυχαίο χρώμα, που επέλεξε το χρήστης με το slider κάτω αριστερά με τυχαία θέση ή/και τυχαίο προσανατολισμό αντίστοιχα. Το slider δέχεται μόνο floats από 0 έως 1 για αυτό αντίστοιχα μπορούμε να εμφανίσουμε από 0 έως 10 drones. Η default τιμή είναι 6 για το αριθμό των drones. Σε περίπτωση που ο χρήστης επιλέξει να εμφανιστούν drones περισσότερα από τις θέσεις των επιφανειών προσγείωσης τότε εμφανίζεται αριθμός drones ίσο με τον αριθμό των θέσεων που υπάρχουν. Για τις τιμές που έχουμε επιλέξει για την επιφάνεια προσγείωσης αυτό δεν συμβαίνει ποτέ για αυτό έγινε και επιλογή N=4 για την επιφάνεια προσγείωσης.

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:···
```

Εικόνα 6: Μέθοδοι για το Ερώτημα 1

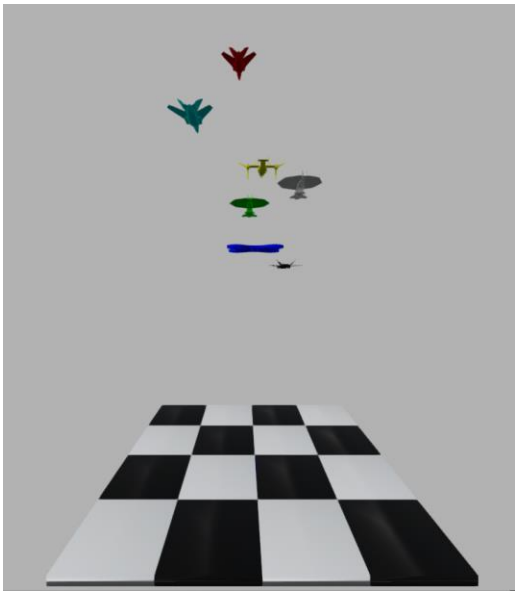
- **landing_pad()**: Δημιουργεί μια επιφάνεια προσγείωσης με NxN θέσεις και διαστάσεις 2x2 για κάθε επιφάνεια
- **randomize_mesh()**: Επειδή κάθε μοντέλο έχει διαφορετικές διαστάσεις, για λόγους ομοιομορφίας κλιμακώνουμε κάθε mesh, ώστε να εφάπτεται στη μοναδιαία σφαίρα και να χωράει σε μια επιφάνεια προσγείωσης. Για την τυχαία εμφάνιση των drones, δημιουργούμε ένα τυχαίο translation vector τηρώντας πάντα τα όρια της σκηνής και προαιρετικά έναν τυχαίο πίνακα περιστροφής και έπειτα εφαρμόζουμε τον μετασχηματισμό στα drones που θα εμφανιστούν.

  $$v_{new} = v_{old} \cdot {R_{rand}}^{T} + t$$

  Όπου :
  v_{new}: τα μετασχηματισμένα vertices
  v_{old}: τα αρχικά vertices
  R_{rand}: ο τυχαίος πίνακας περιστροφής
  t: το τυχαίο translation vector

- **show_drones()**: Οπτικοποιεί ένα συγκεκριμένο αριθμό drones σε τυχαίες θέσεις ή/και με προσανατολισμούς

Εικόνα 8: Οπτικοποίηση drones σε τυχαίες θέσεις και με τυχαίους προσανατολισμούς



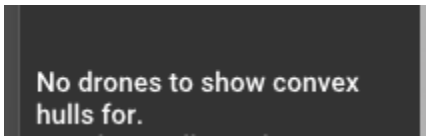
Εικόνα 7: Οπτικοποίηση drones σε τυχαίες θέσεις

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

Σε αυτό το ερώτημα σκοπός είναι η δημιουργία των παρακάτω περιβαλλόντων όγκων:

- **Κυρτό Περίβλημα (Convex Hull):** η μικρότερη κυρτή επιφάνεια που περικλείει το drone.
- **Axis Aligned Bounding Box**: ορθογώνιο κουτί που περικλείει το drone, ευθυγραμμισμένο με τους άξονες του συστήματος συντεταγμένων (X, Y, Z).
- **K-Discrete Oriented Polytope**: ένα πολύεδρο που χρησιμοποιείται για την περικύκλωση του drone, με πλευρές προσανατολισμένες σε **K** προκαθορισμένες κατευθύνσεις. Παρέχει πιο ακριβή περιγράμματα από το AABB, επιτρέποντας καλύτερη προσαρμογή στο σχήμα του drone. Για $\boldsymbol{K} = 6$ λαμβάνουμε το AABB. Στο ερώτημα αυτό δημιουργήθηκε το 14-DOP.

Ο χρήστης πατώντας τα ***C, A, K*** μπορεί να εμφανίσει ή να σβήσει τα Convex Hulls, AABBs, 14-DOPs αντίστοιχα για τα drones που υπάρχουν στην σκηνή. Αν δεν υπάρχουν drones, τότε εμφανίζει ένα προειδοποιητικό μήνυμα ανάλογα τον περιβάλλον όγκος στον διάλογο της σκηνής.



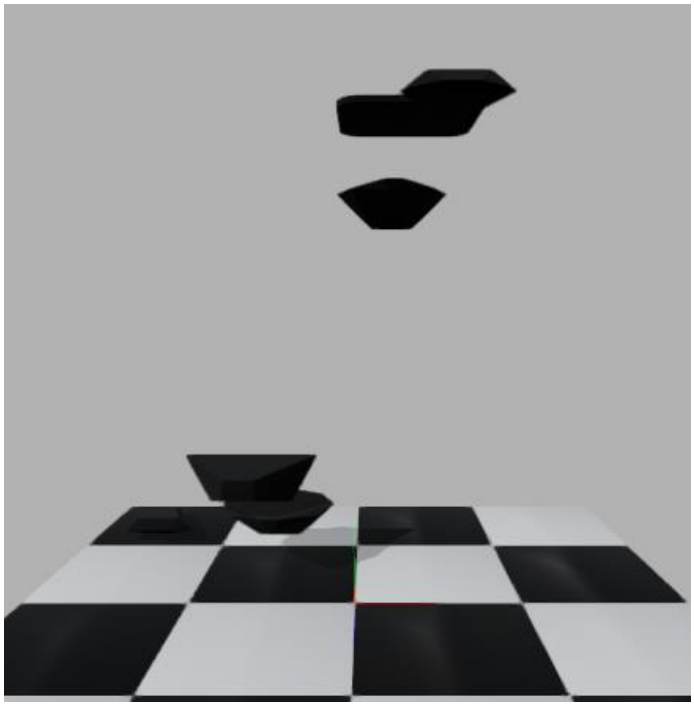
Εικόνα 9: Μήνυμα απεικόνισης Convex Hulls αν δεν υπάρχουν drone στην σκηνή

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:
```

Εικόνα 10:Μέθοδοι για το Ερώτημα 2

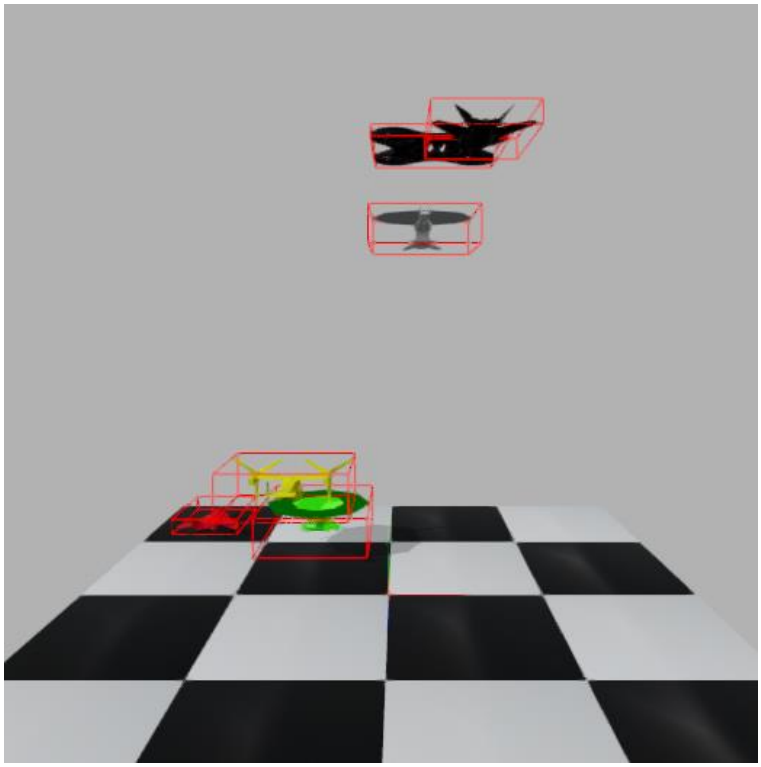
- **get_convex_hull()**: Επιστρέφει το convex hull του Mesh χρησιμοποιώντας την Open3D βιβλιοθήκη με μια μέθοδο βασισμένη στην Quick Hull.
- **show_convex_hull()**: Απεικονίζει και αποθηκεύει σε ένα λεξικό το convex hull



Εικόνα 11: Δημιουργία και Απεικόνιση 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 12: Δημιουργία και Απεικόνιση 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]

])
```

Εικόνα 13: Οι 14 κατευθύνσεις που ορίζουν τα planes που θα περικλείσουν το drone

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

$$proj = v \cdot dir^T$$

Όπου:

$proj$: η προβολή του vertex v σε μια συγκεκριμένη κατεύθυνση dir

v: τα vertices του drone

dir: η κατεύθυνση προβολής

Για κάθε τέτοια κατεύθυνση βρίσκουμε την μέγιστη και ελάχιστη τιμή αυτών των προβολών και βρίσκουμε σε ποια vertices του drone αντιστοιχούν.
Αυτά τα vertices καθορίζουν τις θέσεις των planes που θα περικλείσουν το mesh. Αφού βρούμε τις εξισώσεις αυτών των planes (σε αυτή την υλοποίηση δημιουργήθηκαν τετράγωνα planes 3x3 κεντραρισμένα στα vertices που αντιστοιχούν στις μέγιστες και ελάχιστες προβολές και με κατεύθυνση την αντίστοιχη για την κάθε προβολή) έπειτα βρίσκουμε τις τομές των planes με τις γραμμές (ευθείες) του AABB του mesh. Σε αυτό το στάδιο ελέγχουμε τομές μεταξύ plane διαγώνιων κατευθύνσεων και με γραμμές του AABB μόνο. Αν ελέγχαμε τομές μεταξύ των planes των οριζόντιων και κάθετων κατευθύνσεων και με ευθείες του AABB θα λαμβάναμε πάρα πολλά άκυρα σημεία τομής που δεν μας ενδιαφέρουν.
Για την τομή ευθείας-plane χρησιμοποιούμε τις παραμετρικές εξισώσεις του:

- Για την ευθεία:

$$L = p_1 + t \cdot line_{dir}$$

p_1: ένα σημείο της ευθείας (το πρώτο εδώ πέρα)
$line_{dir}$: το διάνυσμα κατεύθυνσης της ευθείας
t: παράμετρος ευθείας

- Για το plane:

$$a \cdot x + b \cdot y + c \cdot z + d = 0$$

(a, b, c): το normal διάνυσμα του plane

d: μια σταθερά

Αντικαθιστώντας την εξίσωσης ευθείας στην εξίσωση του plane και λύνοντας ως προς την παράμετρο t:

$$t = \frac{(a \cdot x_0 + b \cdot y_0 + c \cdot z_0 + d}{N}$$

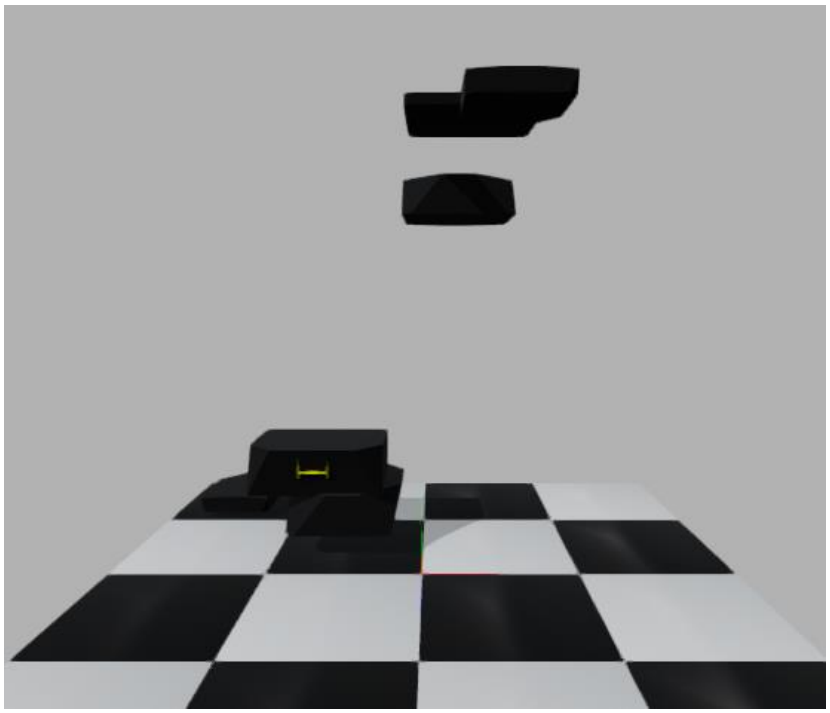
N: το εσωτερικό γινόμενο μεταξύ του normal του plane και του διανύσματος κατεύθυνσης της ευθείας

Το σημείο τομής βρίσκεται μετά ως εξής:

$$p_{inter} = p_1 + t \cdot line_{dir}$$

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. Μια καλύτερη προσέγγιση θα ήταν να βρίσκαμε και σημεία τομής μεταξύ των διαγώνιων planes όμως αυτή η μέθοδος δεν έχει υλοποιηθεί.

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

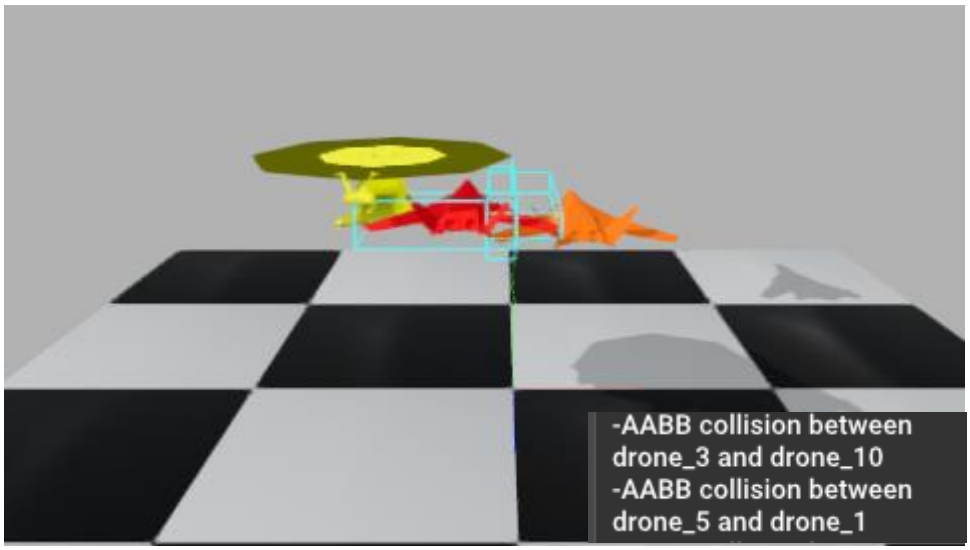


Εικόνα 14: Δημιουργία και Απεικόνιση 14-DOPs

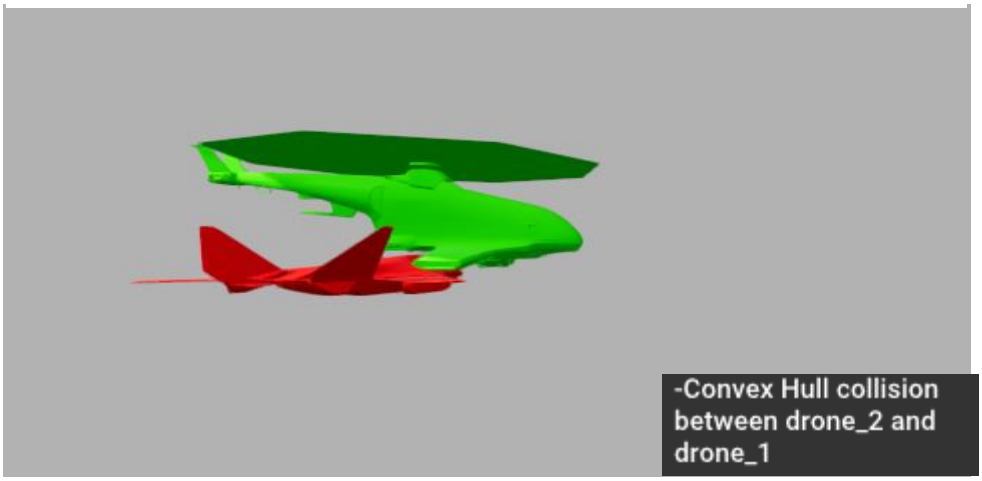
3.3 Ερώτημα 3 (Έλεγχος συγκρούσεων με AABBs και Convex Hulls)

Σκοπός του παρών ερωτήματος είναι η υλοποίηση αλγορίθμων ανίχνευσης σύγκρουσης βάσει κάποιων περιβαλλόντων όγκων. Υλοποιήθηκαν αλγόριθμοι βάσει των AABB και των Convex Hulls.

Ο χρήστης πατώντας τα ***N, L*** ελέγχει με AABBs και Convex Hulls αντίστοιχα και αν εμφανίζεται σύγκρουση μεταξύ οποιωνδήποτε drones της σκηνής το αποτέλεσμα τυπώνεται στον διάλογο της σκηνής. Στην περίπτωση του ελέγχου βάσει τα AABBs εμφανίζεται και το κυβοειδές τομής.



Εικόνα 16: Έλεγχος σύγκρουσης βάσει των AABBs και εμφάνιση τομής των AABBs



Εικόνα 15: Έλεγχος σύγκρουσης βάσει των 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:...
```

Εικόνα 17: Μέθοδοι για το Ερώτημα 3

- **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 και βλέποντας αν υπάρχει επικάλυψη**.**
- **collision_detection_chs()**: Έλεγχος συγκρούσεων βάσει των convex_hulls χρησιμοποιώντας την Trimesh βιβλιοθήκη και συγκεκριμένα με την συνάρτηση **collision()** του utility.py συγκρίνοντας αυτούσια τα Convex Hull meshes του κάθε drone. Αν υπάρχουν συγκρούσεις τυπώνονται στο διάλογο της σκηνής.

Συμπεράσματα για αυτούς τους αλγορίθμους ανίχνευσης συγκρούσεων:

AABBs

Τα AABBs είναι ορθογώνια κουτιά που περικλείουν αντικείμενα, ευθυγραμμισμένα με τους άξονες των συντεταγμένων. Το κύριο τους πλεονέκτημα είναι η απλότητα υπολογισμού και η ταχύτητα στην ανίχνευση συγκρούσεων. Οι υπολογισμοί για τη σύγκρουση μεταξύ δύο AABBs είναι γρήγοροι, καθώς απαιτούν μόνο συγκρίσεις των ελάχιστων και μέγιστων συντεταγμένων. Ωστόσο, το μειονέκτημά τους είναι ότι μπορεί να μην παρέχουν ακριβή αποτελέσματα, ειδικά για μη κυρτά αντικείμενα, καθώς μπορεί να υπάρξουν περιπτώσεις ψευδών θετικών.

Convex Hulls

Αντίθετα, οι Convex Hulls προσφέρουν μια πιο ακριβή αναπαράσταση του σχήματος ενός αντικειμένου, καθώς περικλείουν όλα τα σημεία του με μια κυρτή επιφάνεια. Οι αλγόριθμοι ανίχνευσης σύγκρουσης με Convex Hulls μπορούν να είναι πιο ακριβείς, αποφεύγοντας ψευδείς θετικούς. Ωστόσο, απαιτούν περισσότερο υπολογιστικό κόστος, καθώς οι υπολογισμοί για την κατασκευή του convex hull και τη σύγκρουση είναι πιο περίπλοκοι.

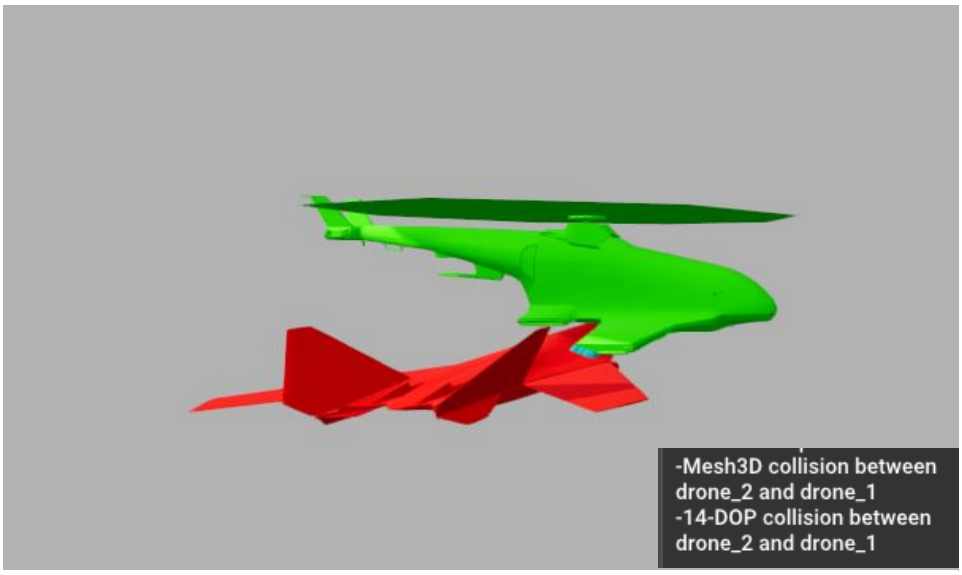
Σύγκριση

Εν κατακλείδι, η επιλογή μεταξύ AABBs και Convex Hulls εξαρτάται από τις απαιτήσεις της εφαρμογής. Τα AABBs προσφέρουν ταχύτητα και απλότητα, κάνοντάς τα κατάλληλα για εφαρμογές που απαιτούν γρήγορη ανίχνευση συγκρούσεων, ενώ τα Convex Hulls παρέχουν μεγαλύτερη ακρίβεια, ιδίως σε πολύπλοκα σχήματα, αλλά σε κόστος υπολογιστικής απόδοσης.

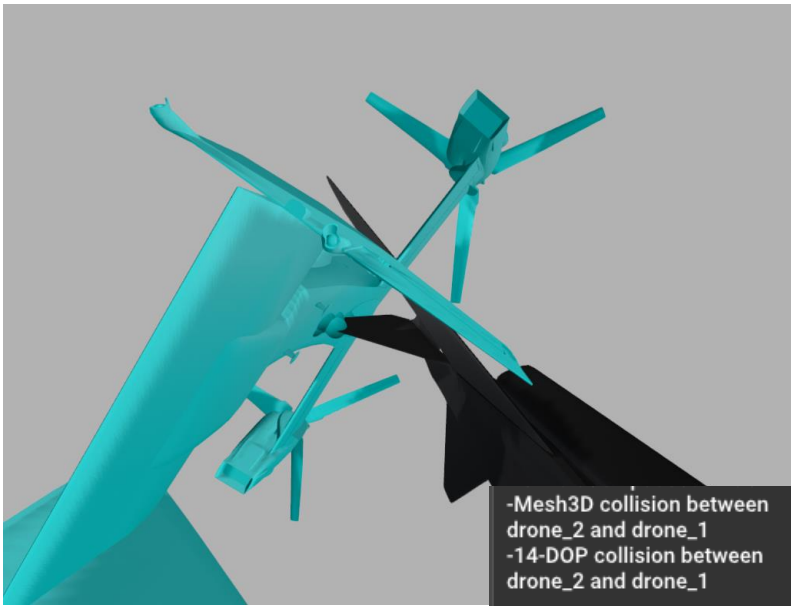
3.4 Ερώτημα 4, 5 (Έλεγχος συγκρούσεων με 14-DOPs και τα ίδια τα Meshes και Απεικόνιση σημείων Σύγκρουσης)

Σκοπός των ερωτημάτων αυτών είναι η υλοποίηση δύο αλγορίθμων ανίχνευσης σύγκρουσης μία λίγο καλύτερη ακρίβεια από τους προηγούμενους και μία με απόλυτη ακρίβεια καθώς και η απεικόνιση των συγκρούσεων όταν τα drones έχουν και τυχαίους προσανατολισμούς. Υλοποιήθηκαν αλγόριθμοι βάσει των 14-DOPs και των ίδιων των Meshes αντίστοιχα.

Ο χρήστης πατώντας τα ***M, V*** ελέγχει με 14-DOPs και τα ίδια τα Mesh αντίστοιχα αν εμφανίζεται σύγκρουση μεταξύ οποιωνδήποτε drones της σκηνής το αποτέλεσμα τυπώνεται στον διάλογο της σκηνής. Στην περίπτωση του ελέγχου βάσει των ίδιων των Meshes εμφανίζονται και τα σημεία τομής. Επίσης για λόγους ταχύτητας στην απόλυτα ακριβή μέθοδο με τα ίδια τα Meshes ελέγχονται πρώτα όλοι οι άλλοι μέθοδοι σύγκρουσης και μετά συγκρίνει αυτούσια τα meshes. Έτσι μπορούμε να απορρίψουμε περιπτώσεις όχι σύγκρουσης πολύ πιο γρήγορα.



Εικόνα 18: Έλεγχος σύγκρουσης βάσει των 14-DOPs και των ίδιων των Meshes και εμφάνιση σημείων τομής για την δεύτερη μέθοδο (με χρώμα CYAN). Χωρίς τυχαίο προσανατολισμό.



Εικόνα 19: Έλεγχος σύγκρουσης βάσει των 14-DOPs και των ίδιων των Meshes και εμφάνιση σημείων τομής για την δεύτερη μέθοδο (με χρώμα CYAN). Με τυχαίο προσανατολισμό.

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:
```

Εικόνα 20: Μέθοδοι για τα Ερωτήματα 4, 5

- **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 14 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.

Πιο συγκεκριμένα:

- **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.

- **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.

Συμπεράσματα για αυτούς τους αλγορίθμους ανίχνευσης συγκρούσεων:

K-DOPs

Τα K-DOPs, και συγκεκριμένα α K-DOPs με K=14, είναι πολύεδρα σχήματα που χρησιμοποιούνται για να περικλείουν αντικείμενα σε τρισδιάστατο χώρο. Αυτά τα K-DOPs ορίζονται από 14 κατευθύνσεις, που επιτρέπουν μια πιο ακριβή προσαρμογή στο σχήμα των αντικειμένων σε σύγκριση με απλά AABBs. Η ανίχνευση σύγκρουσης με K-DOPs είναι ταχύτερη από αυτή που περιλαμβάνει την άμεση σύγκριση meshes, καθώς απαιτεί λιγότερες υπολογιστικές διαδικασίες για να ελεγχθεί εάν οι K-DOPs επικαλύπτονται. Παρόλα αυτά, η κατασκευή και η συντήρηση των K-DOPs είναι πιο πολύπλοκες και απαιτούν περισσότερο υπολογιστικό κόστος.

Άμεσες Συγκρίσεις Meshes

Από την άλλη, η άμεση σύγκριση των meshes παρέχει τη μεγαλύτερη ακρίβεια στην ανίχνευση συγκρούσεων, καθώς εξετάζει τις ακριβείς γεωμετρικές μορφές των αντικειμένων. Αυτή η μέθοδος αποφεύγει ψευδείς θετικούς που μπορεί να προκύψουν με τις K-DOPs ή άλλες μεθόδους περιβλήματος. Ωστόσο, οι υπολογισμοί για την ανίχνευση σύγκρουσης μεταξύ των meshes είναι υπολογιστικά εντατικοί, απαιτώντας αλγόριθμους όπως η μέθοδος GJK (Gilbert-Johnson-Keerthi) ή η χρήση τριγώνων για σύγκριση.

Σύγκριση

Συνοψίζοντας, η επιλογή μεταξύ K-DOPs και άμεσων συγκρίσεων meshes εξαρτάται από τις απαιτήσεις της εφαρμογής. Οι K-DOPs προσφέρουν μια καλή ισορροπία μεταξύ ταχύτητας και ακρίβειας, ιδιαίτερα για δυναμικά σενάρια όπου η ταχύτητα είναι κρίσιμη. Αντίθετα, οι άμεσες συγκρίσεις meshes παρέχουν τη μέγιστη ακρίβεια, αλλά με υψηλότερο υπολογιστικό κόστος. Για εφαρμογές όπου η ταχύτητα είναι πιο σημαντική από την απόλυτη ακρίβεια, οι K-DOPs με K=14 είναι μια εξαιρετική επιλογή. Ωστόσο, σε περιβάλλοντα όπου η ακριβής αναπαράσταση των αντικειμένων είναι απαραίτητη, η άμεση σύγκριση των meshes είναι προτιμότερη.

4. Ερωτήματα Μέρος Β(Προσομοίωση)

Στα παρακάτω ερωτήματα υλοποιήθηκαν τρία είδη προσομοιώσεων:

- Μετακίνηση των drones με συγκεκριμένη ταχύτητα και διακοπή κίνησης αν εμφανιστεί σύγκρουση
- Μετακίνηση των drones με συγκεκριμένη ταχύτητα και ανανέωση ταχύτητας αν εμφανιστεί σύγκρουση
- Όλα τα drones κινούνται με τυχαία ταχύτητα αλλά κατεύθυνση προς μία διαθέσιμη επιφάνεια προσγείωσης (Πρωτόκολλο Προσγείωσης)

Ο χρήστης πατώντας τα ***T, F, SPACE*** ενεργοποιεί την αντίστοιχη 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, ···
```

Εικόνα 21: Μέθοδοι για τα Ερωτήματα 6, 7, 8, 9

- **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: ···
```

Εικόνα 22: Υλοποίηση του time step και των flag παύσεων της κάθε προσομοίωσης στην μέθοδο που τρέχει πάντα όταν είναι ανοιχτή η εφαρμογή

Η τιμή της last_update ανανεώνεται κάθε φορά που κουνιούνται τα drones

- **move_drone()**: Μετακινεί ένα drone με συγκεκριμένη ταχύτητα μετατοπίζοντας όλα τα vertices του καθώς και τις προβολές του.
- **move_drone_to_point()**: Μετακινεί ένα drone σε ένα συγκεκριμένο σημείο μετατοπίζοντας όλα τα vertices του ώστε να έχουν κέντρο μάζας το συγκεκριμένο σημείο καθώς και τις προβολές 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).

```
DRONES = [
          "models/F52.obj",
          "models/Helicopter.obj",
          "models/quadcopter_scifi.obj",
          "models/v22_osprey.obj"
         ]


SPEEDS = np.array([
                   [0.0, 0.1, 0.0],
                   [0.0, -0.1, 0.0],
                   [0.0, 0.3, 0.3],
                   [0.1, 0.5, 0.1]
                   ])

SPEED_MAP = {DRONES[i]: SPEEDS[i] for i in range(len(DRONES))}
```

Εικόνα 23: Αρχικές ταχύτητες κάθε μοντέλου drone

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

Στο παρακάτω ερώτημα υλοποιήθηκε προσομοίωση μετακίνησης των drones με συγκεκριμένη ταχύτητα και διακοπή κίνησης αν εμφανιστεί σύγκρουση. Οι ανιχνεύσεις συγκρούσεων γίνονται με την απόλυτα ακριβή μέθοδο του Ερωτήματος 4.

Ο χρήστης πατώντας τα ***T*** ενεργοποιεί την προσομοίωση.

Για το ερώτημα αυτό αναπτύχθηκε η εξής μέθοδος:

```
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():···
```

Εικόνα 24: Μέθοδος προσομοίωσης του Ερωτήματος 6

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 μηδενική ταχύτητα) 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.
- Ελέγχουμε αν το drone συγκρούεται με κάποιο άλλο drone στην σκηνή και τα σταματάμε και τα δύο και απεικονίζουμε τα σημεία σύγκρουσης. Αν και τα δύο drones είναι σταματημένα δεν γίνεται έλεγχος σύγκρουσης.

4.2 Ερώτημα 7 (Ανανέωση ταχυτήτων για αποφυγή συγκρούσεων)

Στο παρακάτω ερώτημα υλοποιήθηκε προσομοίωση μετακίνησης των drones με συγκεκριμένη ταχύτητα και ανανέωση ταχύτητας αν εμφανιστεί σύγκρουση. Οι ανιχνεύσεις συγκρούσεων γίνονται με την απόλυτα ακριβή μέθοδο του Ερωτήματος 4. Σε αυτή την προσομοίωση επειδή θέλουμε τα drones να αποφεύγουν το ένα με το άλλο η ανίχνευση συγκρούσεων γίνεται μεταξύ ενός drone και ενός άλλου στην θέση που θα ήταν στα επόμενα 5 frames. Αναλυτικότερα παρακάτω στην επεξήγηση της αντίστοιχης μεθόδου.

Ο χρήστης πατώντας τα ***F*** ενεργοποιεί την προσομοίωση.

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(): ···
```

Εικόνα 25: Μέθοδος προσομοίωσης του Ερωτήματος 7

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:

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

  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.

Συνοψίζοντας, στην παραπάνω περιγραφή, εξετάζεται η διαδικασία ανίχνευσης και διαχείρισης συγκρούσεων μεταξύ drones σε μια σκηνή, με στόχο την αποφυγή ατυχημάτων. Ο έλεγχος γίνεται για τις πιθανές θέσεις των drones στα επόμενα πέντε frames, κάτι που ελαχιστοποιεί τις ενδιάμεσες συγκρούσεις και διευκολύνει τη σταθερότητα της κίνησης. Η ανανέωση των ταχυτήτων των drones με βάση το surface normal σε ένα σημείο σύγκρουσης, χρησιμοποιώντας τον καθορισμένο τύπο, επιτρέπει στα drones να «γλιστρούν» ομαλά το ένα δίπλα στο άλλο, αντί να αναπηδούν. Αυτή η προσέγγιση συμβάλλει στη βελτίωση της αλληλεπίδρασης των αντικειμένων στον τρισδιάστατο χώρο, προάγοντας μια πιο ρεαλιστική και ασφαλή συμπεριφορά. Αρχικά η ανανέωση ταχυτήτων γινόταν αντιστρέφοντας τις ταχύτητες των drone όμως σε περίπτωση πολλών γειτονικών συγκρούσεων μπορεί κάποιο drone να ταλαντωνόταν χωρίς να κουνιέται καθόλου στη σκηνή.

4.3 Ερώτημα 8 (Πρωτόκολλο προσγείωσης-απογείωσης χωρίς συγκρούσεις)

Στο παρακάτω ερώτημα υλοποιήθηκε προσομοίωση Πρωτοκόλλου Προσγείωσης. Τα drones προσγειώνονται ένα ένα όμως με σχολιασμό μια γραμμής κώδικα υπάρχει δυνατότητα να προσγειώνονται όλα μαζί. Αν ο αριθμός των drones δεν είναι ίσος με τον μέγιστο της επιφάνειας προσγείωσης τότε εμφανίζονται σε τυχαία στιγμιότυπα drones μέχρι ο αριθμός των drones στη σκηνή να φτάσει τον μέγιστο. Οι ανιχνεύσεις συγκρούσεων γίνονται με την απόλυτα ακριβή μέθοδο του Ερωτήματος 4. Σε αυτή την προσομοίωση ακολουθούμε την ίδια λογική ανίχνευση και αποφυγής συγκρούσεων όπως και στο Ερώτημα 8.

Ο χρήστης πατώντας τα ***SPACE*** ενεργοποιεί την προσομοίωση.

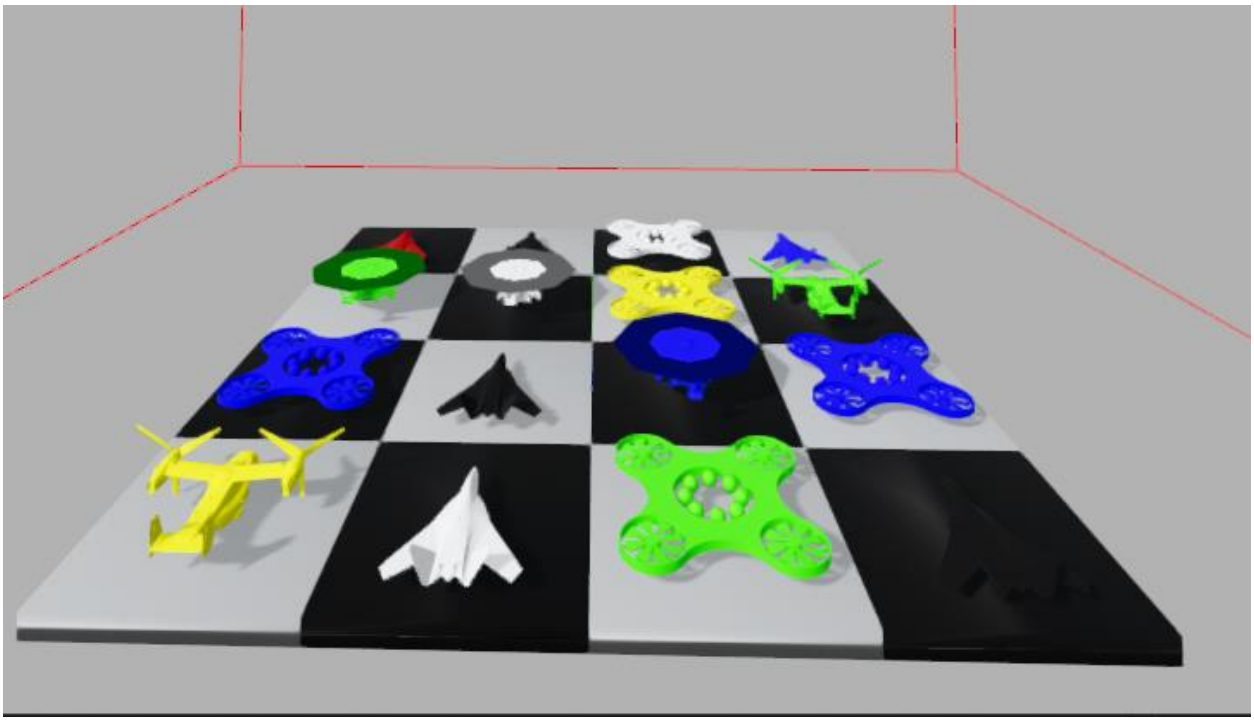
Για το ερώτημα αυτό αναπτύχθηκε η εξής μέθοδος:

```
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: ···
```

Εικόνα 26: Μέθοδος προσομοίωσης Πρωτοκόλλου Προσγείωσης του Ερωτήματος 8

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 27: Όλα τα drones προσγειωμένα

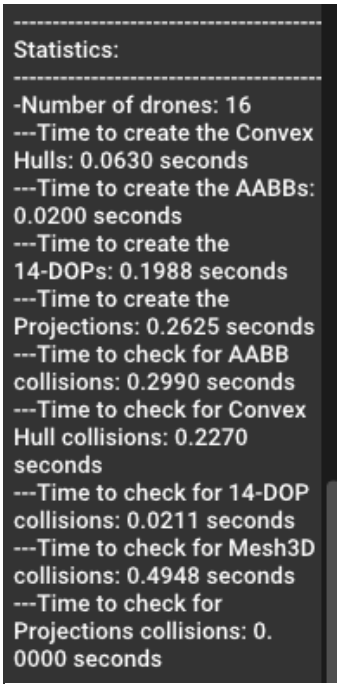
4.4 Ερώτημα 9 (Απεικόνιση Προσομοίωσης και Στατιστικά στοιχεία)

Σε αυτό το ερώτημα απεικονίζουμε τις παραπάνω προσομοιώσεις και τυπώνουμε στον διάλογο της σκηνής ενδιαφέροντα στατιστικά.

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("----------------------------------------")
```

Εικόνα 29: Μέθοδο εμφάνισης διαφόρων στατιστικών στοιχείων Ερωτήματος 9



Εικόνα 28: Στατιστικά στοιχεία τυπωμένα στον διάλογο της σκηνής αφού τρέξει μια προσομοίωση Πρωτοκόλλου Προσγείωσης

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

Σε αυτό το ερώτημα υλοποιήθηκαν μέθοδοι προβολής και απεικόνισης των περιβλημάτων των drones στα τρία επίπεδα xy, xz, yz για κάθε στιγμιότυπο. Έπειτα υλοποιήθηκε μέθοδος ανίχνευσης σύγκρουσης βάσει αυτών των προβολών και θα σχολιαστεί στο τέλος της ενότητας η ταχύτητα και η αξιοπιστία του.

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, ···
```

Εικόνα 30: Μέθοδοι για το Extra Ερώτημα

- **get_projections()**: Επιστρέφει τις προβολές στα xy, xz, yz επίπεδα ενός mesh με την **get_projection()** της utility.py μηδενίζοντας την συντεταγμένη που δεν ανήκει στο επίπεδο προβολής και τις μετακινεί στα άκρα των ορίων της σκηνής Για να μειώσουμε την πολυπλοκότητα των Mesh των προβολών αφαιρούμε τα διπλά και τα vertices που πιθανόν να μην ανήκουν σε κάποιο τρίγωνα μετά τον μετασχηματισμό προβολής:

```
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
```

Εικόνα 31: Μέθοδος για εύρεση προβολών στα xy, xz, yz επίπεδα

- **show_projections()**: Απεικονίζει τις προβολές στα επίπεδα που ορίζει το bounding cuboid
- **collision_detection_projections()**: Έλεγχος συγκρούσεων βασισμένη στη χρήση 2D προβολών και των αντίστοιχων **Axis-Aligned Bounding Boxes (AABBs)**. Ουσιαστικά, η μέθοδος ελέγχει τις συγκρούσεις των αντικειμένων σε τρεις κατευθύνσεις, μέσω των προβολών τους σε τρία επίπεδα. Αν οι AABBs των 2D προβολών δεν τέμνονται σε κάποιο από τα επίπεδα, μπορούμε με σιγουριά να πούμε ότι δεν υπάρχει σύγκρουση μεταξύ των 3D mesh. Αν, αντίθετα, παρατηρήσουμε τομή σε όλα τα επίπεδα, αυτό υποδεικνύει μια πιθανή σύγκρουση.

Αξιοπιστία και Ταχύτητα

Η αξιοπιστία της συγκεκριμένης μεθόδου είναι υψηλή, καθώς η χρήση τριών διαφορετικών επιπέδων προβολής ελαχιστοποιεί τις ψευδείς θετικούς και αυξάνει την πιθανότητα έγκαιρης ανίχνευσης πιθανών συγκρούσεων. Ωστόσο, πρέπει να σημειωθεί ότι, σε περιπτώσεις όπου η γεωμετρία των αντικειμένων είναι πολύπλοκη ή τα shapes τους δεν είναι ευθυγραμμισμένα με τους άξονες, η μέθοδος μπορεί να οδηγήσει σε συγκρούσεις που δεν εντοπίζονται αμέσως, αν και το ίδιο ισχύει και για τις περισσότερες μεθόδους ανίχνευσης.

Σε σχέση με άλλες μεθόδους ανίχνευσης συγκρούσεων:

- **AABBs**: Η χρήση AABBs προσφέρει γρήγορη ανίχνευση, καθώς οι υπολογισμοί για τη σύγκρουση είναι απλοί και απαιτούν μόνο συγκρίσεις των ελάχιστων και μέγιστων συντεταγμένων. Ωστόσο, οι AABBs δεν είναι τόσο ακριβείς όσο η μέθοδος προβολών, καθώς μπορεί να παραβλέπουν συγκρούσεις λόγω της ευθείας τους φύσης.
- **Convex Hulls**: Οι Convex Hulls παρέχουν μεγαλύτερη ακρίβεια στην ανίχνευση συγκρούσεων, καθώς περιλαμβάνουν την πλήρη γεωμετρία των αντικειμένων. Παρόλα αυτά, οι υπολογισμοί είναι πιο απαιτητικοί και μπορεί να επηρεάσουν την ταχύτητα σε εφαρμογές όπου απαιτείται γρήγορη ανίχνευση συγκρούσεων.
- **K-DOPs**: Τα K-DOPs, ειδικά με K=14, συνδυάζουν τα πλεονεκτήματα των AABBs και των Convex Hulls, προσφέροντας καλή ισορροπία μεταξύ ταχύτητας και ακρίβειας. Ωστόσο, η κατασκευή τους είναι πιο περίπλοκη, ενώ η μέθοδος προβολών είναι πιο απλή στην υλοποίηση.

Συμπέρασμα για αυτή την μέθοδο ανίχνευσης συγκρούσεων:

Συνολικά, η μέθοδος **collision_detection_projections()** συνδυάζει τα πλεονεκτήματα της ταχύτητας και της αξιοπιστίας, προσφέροντας μια αποτελεσματική λύση για την ανίχνευση συγκρούσεων. Ειδικά σε σενάρια όπου η απλότητα είναι σημαντική και η γεωμετρία των αντικειμένων δεν είναι ιδιαίτερα πολύπλοκη, αυτή η μέθοδος μπορεί να είναι εξαιρετικά χρήσιμη. Ωστόσο, για εφαρμογές που απαιτούν υψηλότερη ακρίβεια, μπορεί να είναι αναγκαία η χρήση πιο προηγμένων μεθόδων όπως οι Convex Hulls ή τα K-DOPs.

5. Σύνοψη και Περιθώρια Βελτίωσης

Η ανίχνευση συγκρούσεων είναι ένα κρίσιμο στοιχείο σε πολλές εφαρμογές, όπως τα βιντεοπαιχνίδια, οι προσομοιώσεις φυσικής και η ρομποτική. Οι μέθοδοι που εξετάστηκαν περιλαμβάνουν AABBs, Convex Hulls, K-DOPs και την ανίχνευση συγκρούσεων μέσω προβολών, και καθεμία έχει τα πλεονεκτήματα και τα μειονεκτήματά της.

Συγκριτική Ανάλυση Μεθόδων Ανίχνευσης Συγκρούσεων

1. **AABBs**: Οι AABBs είναι γρήγορες στην ανίχνευση συγκρούσεων λόγω της απλότητας τους. Η χρήση τους επιτρέπει γρήγορους υπολογισμούς, αλλά η ακριβής τους γεωμετρική αναπαράσταση μπορεί να παραλείψει πιθανές συγκρούσεις, ιδιαίτερα όταν τα αντικείμενα είναι μη ευθυγραμμισμένα με τους άξονες. Έτσι, αν και είναι χρήσιμες σε δυναμικά σενάρια όπου απαιτείται ταχύτητα, η αξιοπιστία τους μπορεί να είναι περιορισμένη.
2. **Convex Hulls**: Τα Convex Hulls προσφέρουν μεγαλύτερη ακρίβεια καθώς περιλαμβάνουν την πλήρη γεωμετρία των αντικειμένων. Αυτή η μέθοδος αποφεύγει ψευδείς θετικούς και παρέχει πιο αξιόπιστα αποτελέσματα, αλλά απαιτεί περισσότερους υπολογιστικούς πόρους, γεγονός που μπορεί να επηρεάσει την ταχύτητα. Είναι ιδανικές για περιβάλλοντα όπου η ακριβής αναπαράσταση της γεωμετρίας είναι απαραίτητη.
3. **K-DOPs**: Τα K-DOPs, ειδικά με K=14, προσφέρουν μια ισχυρή εναλλακτική, καθώς συνδυάζουν πλεονεκτήματα ταχύτητας και ακρίβειας. Η χρήση 14-DOPs επιτρέπει την καλύτερη αναπαράσταση της γεωμετρίας των αντικειμένων, παρέχοντας μια καλή ισορροπία μεταξύ υπολογιστικής αποδοτικότητας και ακριβούς ανίχνευσης συγκρούσεων. Επιπλέον, η δημιουργία 14-DOPs με την προσθήκη σημείων τομής μεταξύ διαγώνιων planes θα μπορούσε να προσφέρει ακόμα καλύτερα αποτελέσματα στην αναγνώριση συγκρούσεων.
4. **Προβολές**: Η μέθοδος ανίχνευσης συγκρούσεων μέσω 2D προβολών με AABBs είναι αποδοτική και απλή στην υλοποίηση. Αν και παρέχει γρήγορη ανίχνευση συγκρούσεων, μπορεί να μην είναι πάντα ακριβής σε περίπλοκες γεωμετρίες. Ωστόσο, η χρήση της σε συνδυασμό με άλλες μεθόδους θα μπορούσε να βελτιώσει την αξιοπιστία των αποτελεσμάτων.

Συμπεράσματα και Συστάσεις

Συνολικά, καμία μέθοδος δεν είναι ιδανική για κάθε εφαρμογή. Η επιλογή της μεθόδου ανίχνευσης συγκρούσεων εξαρτάται από τις συγκεκριμένες ανάγκες της εφαρμογής, όπως η απαιτούμενη ταχύτητα και ακρίβεια. Στην περίπτωση που απαιτείται ένα ρεαλιστικότερο αποτέλεσμα στις προσομοιώσεις, οι τυχαίες ταχύτητες των αντικειμένων, σε συνδυασμό με την πρόβλεψη και ανίχνευση συγκρούσεων βάσει των κινούμενων περιβαλλόντων όγκων (όπως 14-DOPs ή άλλες μεθόδους), θα μπορούσαν να βελτιώσουν την ποιότητα και την ακριβή αναπαράσταση των αλληλεπιδράσεων στον τρισδιάστατο χώρο

6. 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

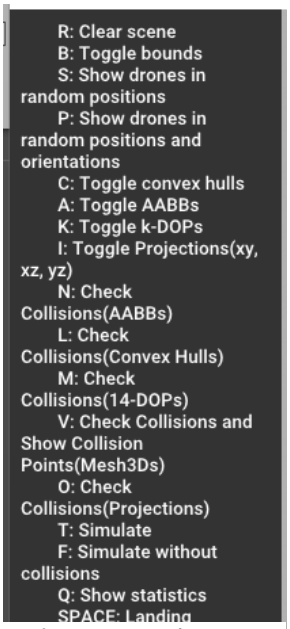
Εγκατάσταση βιβλιοθηκών

- pip install -r requirements.txt

7. 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:



Εικόνα 32: Εντολές Εφαρμογής

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

8. Βιβλιογραφία

Βιβλιοθήκες Python:

- 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/

Επιστημονικά Άρθρα:

- Allesandro Colombo, Domitilla Del Vecchio. Efficient Algorithms for Collision Avoidance at Intersections. 2012. ACM Digital Library: https://dl.acm.org/doi/pdf/10.1145/2185632.2185656

- C Fünfzig, DW Fellner. 2003. Easy Realignment of k-DOP Bounding Volumes, - Graphics Interface. Researchgate.net https://www.researchgate.net/profile/Christoph-Fuenfzig/publication/2892535_Easy_Realignment_of_k-DOP_Bounding_Volumes/links/53d3d7760cf2a7fbb2e9eedc/Easy-Realignment-of-k-DOP-Bounding-Volumes.pdf?_sg%5B0%5D=started_experiment_milestone&origin=journalDetail

Διάφορες άλλες πηγές:

- https://en.wikipedia.org/wiki/Bounding_volume
- https://www.khronos.org/opengl/wiki/Calculating_a_Surface_Normal
- Random Rotation Matrix: https://www.realtimerendering.com/resources/GraphicsGems/gemsiii/rand_rotation.c