Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ ΑΠΟΔΟΣΗΣ ΕΦΑΡΜΟΓΗΣ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ ΑΠΟ ΕΝΑ ΣΜΗΝΟΣ ΑΠΟ DRONES

Αντρέας Βασιλείου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση, προσομοίωση και αξιολόγηση απόδοσης εφαρμογής παρακολούθησης από ένα σμήνος από drones

Αντρέας Βασιλείου

Επιβλέπων Καθηγητής
Δρ. Γιώργος Πάλλης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή μου Δρ. Γιώργο Πάλλη, ο οποίος μου έδωσε την ευκαιρία, στα πλαίσια της διπλωματικής εργασίας, να ασχοληθώ με ένα πολύ ενδιαφέρον θέμα το οποίο θα μπορούσε να ασχοληθώ και στο μέλλον αλλά και την καθοδήγηση του καθ' όλη την διάρκεια υλοποίησης της εργασίας.

Επίσης θα ήθελα να ευχαριστήσω τον υποψήφιο Δρ. Ζαχαρία Γεωργίου, ερευνητής στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου. Με τη βοήθειά του ήμουν σε θέση να καταλάβω σχετικές πληροφορίες για την εργασία και ήμουν σε θέση να ξεκινήσω και να την φέρω εις πέρας.

Ιδιαίτερες ευχαριστίες στον υποψήφιο Δρ. Μωυσή Συμεωνίδη, επίσης ερευνητής στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου. Με την δική του στήριξη και καθοδήγηση καταφέραμε να ολοκληρώσουμε επιτυχώς την εργασία και να εξάγουμε τα σχετικά αποτελέσματα και συμπεράσματα. Θέλω να εκφράσω την ευγνωμοσύνη μου για την παροχή βοηθητικού εργαλείου στο Τμήμα Πληροφορικής.

Περίληψη

Στόχος της διπλωματικής μου εργασίας για την χρονιά 2020-2021, είναι η παρακολούθηση και ο εντοπισμός ενός κινούμενου στόχου κοντά στο πραγματικό χρόνο.

Το AirSim είναι ένας προσομοιωτής ο οποίος έχει αναπτυχθεί ως πρόσθετο του Unreal Engine και χρησιμοποιείται για να προσομοιώσει αλγορίθμους από κινητές συσκευές, όπως drones ή αυτοκίνητα, σε ένα εικονικό περιβάλλον.

Ένας πρωτεύον στόχος της εργασίας είναι να παρέχουμε μία πρότυπη μεθοδολογία με την οποία ο χρήστης θα μπορεί να εξακριβώσει την επίδοση τέτοιων αλγορίθμων χωρίς να χρειάζεται η εγκατάστασή τους σε πραγματικές συσκευές. Συγκεκριμένα το edge computing ασχολείται με την επεξεργασία δεδομένων σε εξυπηρετητές, μειώνοντας τον χρόνο απόκρισης έτσι ώστε οι πληροφορίες να μην χρειάζεται να ταξιδεύουν όσο θα ακολουθούσε μια παραδοσιακή αρχιτεκτονική cloud. Στα πλαίσια αυτής της εργασίας, εξαγάγαμε δεδομένα από το προσομοιωμένο περιβάλλον που μας παρέχει το AirSim, όπως συντεταγμένες, αντικείμενα, εικόνες κτλ., και αναπαραγάγαμε την εφαρμογή πάνω στο Fogify. Το Fogify είναι ένας εξομοιωτής για τοπολογίες Edge & Fog computing που παρέχει ρεαλιστικές συνδέσεις μεταξύ των κόμβων καθώς επίσης και υπολογιστική ισχύ.

Τα αποτελέσματα της εργασίας είναι η υλοποίηση μιας ή περισσότερων προσαρμοσμένων πτήσεων χρησιμοποιώντας το AirSim plugin εισάγοντας ένα ή περισσότερα drones σε ένα περιβάλλον του Unreal Engine. Τα drones μεταφέρουν στον εξυπηρετητή φωτογραφίες για την παρακολούθηση ενός κτηρίου. Ο server χρησιμοποιεί αλγόριθμο ανίχνευσης αντικειμένου για να εξακριβώσει εάν υπάρχει αντικείμενο κοντά στον χώρο του κτηρίου και τα drones αντιδρούν αναλόγως κατά την ανίχνευση.

Περιεχόμενα

List of Figures

Κεφάλαιο 1

Εισαγωγή

1.1 Γενική εισαγωγή

Η ανάπτυξη συστημάτων ασφαλείας λειτουργούν με βασική αρχή την διασφάλιση των σημείων εισόδου σε ένα χώρο ούτως ώστε να μην υπάρξει τυχόν παραβίαση. Βασικό στοιχείο της αποτελεσματικότητας του εκάστοτε συστήματος είναι η γρήγορη επικοινωνία του συστήματος παρακολούθησης με τον διαχειριστή του συστήματος για να ληφθούν τα κατάλληλα μέτρα, κάτι το οποίο είναι βασικό και θεμελιώδης προαπαιτούμενο για το σύστημα ούτως ώστε να πετυχαίνει τον πρωταρχικό του στόχο.

Η χρήση πολλαπλών drone για την παρακολούθηση ενός χώρου δίνει αρκετές δυνατότητες στο σύστημα αφού η παρακολούθηση μπορεί να γίνει από διάφορες θέσεις καλύπτοντας οπτικές γωνίες οι οποίες διαφορετικά θα ήταν πιο δύσκολο να παρακολουθηθούν επιτυχώς.

Το σύστημα από drones έχει ως προαπαιτούμενο την εισαγωγή διαδρομής που θα ακολουθήσουν καθώς επίσης και την διασφάλιση γρήγορης απόκρισης μεταξύ αυτού και του διαχειριστή του.

Το edge computing είναι μια κατανεμημένη, ανοιχτή αρχιτεκτονική πληροφορικής στην οποία τα δεδομένα υποβάλλονται σε επεξεργασία από την ίδια τη συσκευή ή από έναν τοπικό server, αντί να μεταδίδονται σε ένα κέντρο δεδομένων. Αυτό ελαχιστοποιεί την ανάγκη για επικοινωνία μεταξύ ενός μακρινού server και του client μειώνοντας ταυτόχρονα το latency και το bandwidth usage.

Για την επίτευξη των πιο πάνω στόχων δημιουργούνται κάποιοι περιορισμοί και προβλήματα τα οποία υπό διαφορετικές συνθήκες θα ήταν δύσκολο να λυθούν.

Το πρωταρχικό πρόβλημα και το κυριότερο είναι το κόστος της υλοποίησης ενός τέτοιου συστήματος. Υπολογίζοντας τα εργαλεία με πραγματικό εξοπλισμό για να εκτελέσουμε τα πειράματα το κόστος είναι απαγορευτικά υψηλό. Το πρώτο και κύριο εργαλείο του συστήματος είναι τα drones. Η αγορά ενός drone στις σημερινές τιμές κινείται μεταξύ τριψήφιων και τετραψήφιων αριθμών, πόσο μάλλον η αγορά πολλαπλών drones. Σε αυτό το κομμάτι μπορούμε να προσθέσουμε και το κόστος τυχόν βλαβών που θα υπάρξουν στα μηχανήματα κατά την εκτέλεση των πειραμάτων.

Πέραν της τιμής των drones υπάρχει και ο περιορισμός του χρόνου και υποθέτοντας ότι θέλουμε να εκτελέσουμε τα πειράματα με διάφορες παραμέτρους στον πραγματικό κόσμο, με πραγματικούς εξυπηρετητές και drones.

Συνοψίζοντας μπορούμε να διαπιστώσουμε ότι χρησιμοποιώντας τα εργαλεία για προσομοίωση είμαστε σε θέση να προσπεράσουμε τα παραπάνω προβλήματα, να εκτελέσουμε τα πειράματα και να φτάσουμε σε κάποια αποτελέσματα.

1.2 Ορισμός του προβλήματος

Σε αυτή την πειραματική έρευνα το πρόβλημα που είχα να αντιμετωπίσω είναι η αποτελεσματική παρακολούθηση ενός χώρου με την χρήση ενός drone και στην συνέχεια η επέκταση του συστήματος από ένα drone σε πολλά drones με διαφορετικές διαδρομές για τον ίδιο χώρο. Επίσης κατά την παρακολούθηση του χώρου να γίνει αποτελεσματικά και γρήγορα η αναγνώριση ενός κινούμενου στόχου από το σύστημα έχοντας πλήρη εικόνα για τον χώρο χρησιμοποιώντας τα drones.

1.3 Σκοπός

Έχοντας τον ορισμό του προβλήματος μπορούμε να ορίσουμε τον σκοπό ως την υλοποίηση του συστήματος ενός σμήνους από drones το οποίο θα ακολουθεί μια διαδρομή η οποία του δίνεται ως είσοδο και να εξακριβώσει κατά πόσο υπάρχει κάποιο κινούμενο αντικείμενο στον χώρο καθώς επικοινωνεί με ένα εξυπηρετητή για την ανάλυση του περιβάλλοντος. Έχοντας τον σκοπό, τα βήματα για την επίτευξή του είναι:

- Η κατανόηση της αρχιτεκτονικής του προσομοιωτή AirSim την οποία θα χρησιμοποιήσουμε και πως τα συστατικά του συστήματος επικοινωνούν μεταξύ τους για να μπορέσουμε με επιτυχία να απογειώσουμε ένα drone και δίνοντας του μια τροχιά πτήσης να την ακολουθήσει με επιτυχία.
- Εξακρίβωση προ απαιτούμενων αναγκών για την χρήση του AirSim
- Πως θα χρησιμοποιηθεί το προσομοιωτής για την υλοποίηση σμήνους από drones
- Υλοποίηση server ο οποίος θα επικοινωνεί με τα drones
- Πως ο server θα εξακριβώσει κατά πόσο υπάρχει αντικείμενο στον χώρο τον οποίο παρακολουθείται
- Κατανόηση του συστήματος fogify το οποίο θα πάρει ως είσοδο το σύστημα και να μιμηθεί το latency και άλλες μετρικές μεταξύ του drone και του server καθώς επικοινωνούν μεταξύ τους.

Στην συνέχεια σκοπός της έρευνας είναι να εξακριβώσουμε το αντίκτυπο που έχει η υποδομή στην απόδοση του συστήματος.

1.4 Σχετική δουλειά

Έχοντας υπόψιν τους περιορισμούς σχετικά με το κόστος υλοποίησης του συστήματος, η σχετική δουλεία περιορίζεται στους προσομοιωτές οι οποίοι είναι ανοικτού κώδικα.

Παρόμοιος εξομοιωτής ανοικτού κώδικα είναι το DroneKit [12]. Το DroneKit επιτρέπει στους προγραμματιστές να δημιουργούν εφαρμογές που εκτελούνται σε έναν ενσωματωμένο συνοδευτικό υπολογιστή και να επικοινωνούν με τον ελεγκτή πτήσης ArduPilot. Το API επικοινωνεί με οχήματα μέσω MAVLink.

Παρέχει πρόσβαση μέσω προγραμματισμού σε πληροφορίες τηλεμετρίας, κατάστασης και παραμέτρων ενός συνδεδεμένου οχήματος και επιτρέπει τόσο τη διαχείριση της αποστολής όσο και τον άμεσο έλεγχο της κίνησης και των λειτουργιών του οχήματος.

Ένας δεύτερος παρόμοιος εξομοιωτής ανοικτού κώδικα είναι το Webots[13]. Προσφέρει ένα γρήγορο πρωτότυπο περιβάλλον, που επιτρέπει στο χρήστη να δημιουργήσει τρισδιάστατους εικονικούς κόσμους με ιδιότητες φυσικής όπως μάζα, αρμούς, συντελεστές τριβής κ.λπ. Ο χρήστης μπορεί να προσθέσει απλά παθητικά αντικείμενα ή ενεργά αντικείμενα που ονομάζονται κινητά ρομπότ.

Πέραν των προσομοιωτών, υπάρχει πληθώρα από network emulator frameworks τα οποία είναι υλοποιημένα: iFogSim[15], EdgeCloudSim[16] τα οποία είναι βασισμένα στο CloudSim[17]. Επίσης τα FogBed[18], EmuEdge[19], EmuFog[20] και MockFog[21] είναι μερικά από αυτά.

Παρ' όλα αυτά κανένα από τα υπάρχοντα δεν υποστηρίζουν real-time παρακολούθηση. Επίσης όλα τα παραπάνω δεν μπορούν να υποστηρίξουν πειραματισμό μεγάλης κλίμακας.

Κεφάλαιο 2

Μεθοδολογία και Βασικές Έννοιες

2.1 Μεθοδολογία

Στην έρευνα χρησιμοποιήθηκαν τα λειτουργικά συστήματα Windows 10 και Ubuntu 20.04. Η γλώσσα προγραμματισμού η οποία χρησιμοποιήθηκε είναι η Python Anaconda μέσα από το Microsoft Visual Studio IDE. Χρησιμοποιήθηκε repository στο GitHub για αποθήκευση του κώδικα.

Στην έρευνα αναπτύχθηκε το πρόγραμμα το οποίο χειρίζεται τα drones και στη συνέχεια αναπτύχθηκε ο server ο οποίος επικοινωνεί με τα drones. Αφού επιτεύχθηκε ο στόχος για υλοποίηση του συστήματος με ένα drone, στην συνέχεια αναπτύχθηκε το πρόγραμμα το οποίο χειρίζεται τα inputs τα οποία είναι σε μορφή json και αναλόγως με τα δεδομένα εισόδου από τον χρήστη γίνεται το ανάλογο σενάριο με τον αριθμό drones και μονοπατιών τα οποία θα

ακολουθήσουν. Έγινε επέκταση του αρχικού προγράμματος για περισσότερο από ένα drones και η παράλληλη τους εκτέλεση. Τέλος έγινε η υλοποίηση container το οποίο δόθηκε ως input στο fogify για εξαγωγή αποτελεσμάτων.

2.2 Βασικές Έννοιες

2.2.1 AirSim Simulator

2.2.1.1 Overview

Το AirSim είναι ένα Simulator[1] για drones και αυτοκίνητα χτισμένο στο Unreal Engine. Είναι ανοιχτού κώδικα, cross platform. Αναπτύσσεται ως ένα Unreal plugin που μπορεί απλώς να μεταφερθεί σε οποιοδήποτε Unreal περιβάλλον. [6] Αντιμετωπίζει δύο βασικά προβλήματα τα οποία συναντούμε κατά την ανάπτυξη αυτόνομων συστημάτων. Η απαίτηση μεγάλου όγκου δεδομένων για εκπαίδευση και δοκιμή συστημάτων καθώς επίσης και την δυνατότητα εντοπισμού σφαλμάτων σε ένα προσομοιωτή. Το AirSim διαθέτει ένα ευρύ φάσμα από επαιδευτικές εμπειρίες έτσι ώστε τα αυτόνομα συστήματα να μπορούν να εκτεθούν σε διάφορα σενάρια προτού αναπτυχθούν στον πραγματικό κόσμο. Το AirSim υποστηρίζει hardware-in-the-loop και μπορεί εύκολα να επεκταθεί για να φιλοξενήσει διάφορους νέους τύπους αυτόνομων οχημάτων, πλατφόρμες υλικού και πρωτόκολλα λογισμικού.

Figure 2.1 Drone in AirSim

Στο σχήμα 2.1 μπορούμε να διακρίνουμε την προσομοίωση του AirSim με την χρήση drone σαν όχημα.

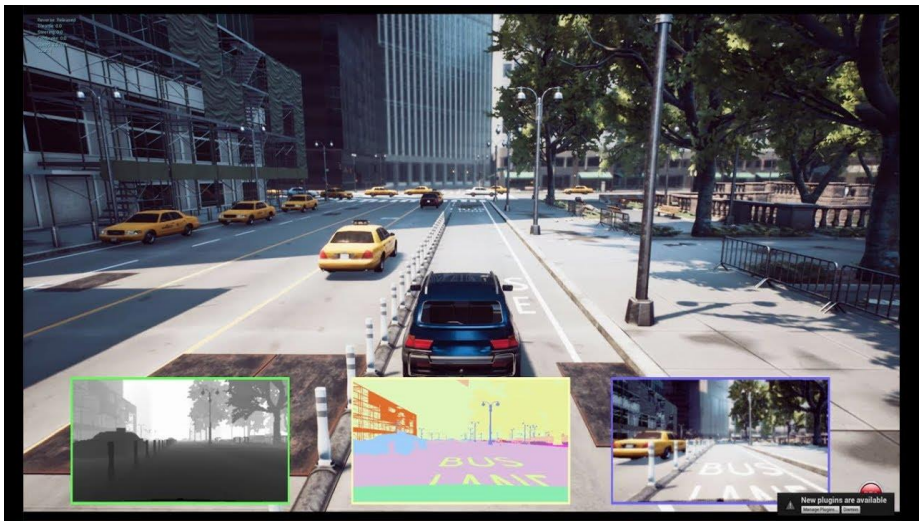


Figure 2.2 Car in AirSim

Στο σχήμα 2.2 μπορούμε να διακρίνουμε την προσομοίωση του AirSim με την χρήση αυτοκινήτου σαν όχημα.

2.2.1.2 Settings

Το αρχείο settings είναι σε μορφή json. Κατά την πρώτη εκκίνηση το AirSim θα δημιουργήσει το αρχείο settings.json[2] χωρίς ρυθμίσεις στα Documents. Μέσα από το συγκεκριμένο αρχείο γίνεται η αρχικοποίηση των οχημάτων τα οποία θα τρέξουν στο περιβάλλον και διάφορες ρυθμίσεις σχετικά με αυτά. Το AirSim αναζητά τον ορισμό των ρυθμίσεων με 4 διαφορετικούς τρόπους με την ακόλουθη σειρά. Η πρώτη αντιστοίχιση που θα βρεθεί θα χρησιμοποιηθεί:

1) Κοιτάζοντας το path που καθορίζεται από το command line --settings.

Για παράδειγμα:

Στα Windows: AirSim.exe --settings 'C: \ path \ to \ settings.json'

Στα Linux: ./Blocks.sh --settings '/home/$USER/path/to/settings.json'

2) Αναζητώντας ένα έγγραφο json που μεταβιβάστηκε ως argument --settings στο command line.

Για παράδειγμα:

Στα Windows: AirSim.exe --settings '{"foo": "bar"}'

Στα Linux ./Blocks.sh --settings '{"foo": "bar"}'

3) Ψάχνει στο φάκελο του executable για ένα αρχείο που ονομάζεται settings.json.
4) Ψάχνει στον αρχικό φάκελο των χρηστών για ένα αρχείο που ονομάζεται settings.json.

    Στα Windows: Documents\AirSim

    Στα Linux: ~/Documents/AirSim

Για να αποφύγετε προβλήματα, χρησιμοποιήστε πάντα τη μορφή ASCII για να αποθηκεύσετε το αρχείο json. Για περισσότερες πληροφορίες σχετικά με το αρχείο settings.json. Εάν κάποια από τις ρυθμίσεις λείπει από το αρχείο json, τότε χρησιμοποιείται η προεπιλεγμένη τιμή. Ορισμένες προεπιλεγμένες τιμές καθορίζονται απλά ως "" που σημαίνει ότι η πραγματική τιμή μπορεί να επιλεγεί με βάση το όχημα που χρησιμοποιούμε. Για παράδειγμα, η ρύθμιση ViewMode έχει την προεπιλεγμένη τιμή "" που μεταφράζεται σε "FlyWithMe" για Drone και "SpringArmChase" για αυτοκίνητα.

Για χρήση drone θέτουμε το “SimMode”:”Multirotor”.

Το αρχείο settings.json διαθέτει αρκετές επιλογές:

SimMode: Καθορίζει ποια προσομοίωση θα χρησιμοποιηθεί (drone ή αυτοκίνητο).

ViewMode: Καθορίζει ποια κάμερα θα χρησιμοποιηθεί ως default και πώς η κάμερα θα ακολουθεί το όχημα.

TimeOfDay: Αυτή η επιλογή καθορίζει την τοποθεσία του ήλιου στο περιβάλλον.

OriginGeopoint: Αυτή η ρύθμιση καθορίζει το γεωγραφικό πλάτος, μήκος και υψόμετρο του οχήματος που τοποθετείται στο περιβάλλον Unreal. Όλες οι συντεταγμένες που εκτίθενται μέσω API χρησιμοποιούν σύστημα NED σε μονάδες SI, πράγμα που σημαίνει ότι κάθε όχημα ξεκινά από (0, 0, 0) στο σύστημα NED. Οι ρυθμίσεις Ώρας της ημέρας υπολογίζονται για γεωγραφικές συντεταγμένες που καθορίζονται στο OriginGeopoint.

SubWindows: Αυτή η ρύθμιση καθορίζει τι εμφανίζεται σε καθένα από τα 3 δευτερεύοντα παράθυρα που είναι ορατά όταν πατάμε τα πλήκτρα 0,1,2. Το WindowID μπορεί να είναι 0 έως 2, το CameraName είναι οποιαδήποτε διαθέσιμη κάμερα στο όχημα. Ο ακέραιος αριθμός ImageType καθορίζει το είδος

της εικόνας που εμφανίζεται σύμφωνα με το ImageType enum. Η συμβολοσειρά VehicleName επιτρέπει να καθορίσετε το όχημα από το οποίο θα χρησιμοποιείται η κάμερα, που χρησιμοποιείται όταν καθορίζονται πολλά οχήματα στις ρυθμίσεις. Η κάμερα του πρώτου οχήματος θα χρησιμοποιηθεί εάν υπάρχουν λάθη όπως λάθος όνομα οχήματος ή μόνο ένα όχημα.

ClockSpeed: Αυτή η ρύθμιση μας επιτρέπει να ορίσουμε την ταχύτητα του ρολογιού προσομοίωσης. Για παράδειγμα, η τιμή 5,0 σημαίνει ότι το ρολόι προσομοίωσης έχει περάσει 5 δευτερόλεπτα όταν έχει περάσει 1 δευτερόλεπτο (δηλαδή η προσομοίωση εκτελείται πιο γρήγορα).

Segmentation Settings: Η InitMethod καθορίζει τον τρόπο προετοιμασίας των αναγνωριστικών αντικειμένων κατά την εκκίνηση για τη δημιουργία segmentation.

WindSettings: Αυτή η ρύθμιση καθορίζει την ταχύτητα ανέμου σε παγκόσμιο πλαίσιο, σε κατεύθυνση NED. Οι τιμές είναι σε m / s. Από προεπιλογή, η ταχύτητα είναι 0, δηλ. Χωρίς άνεμο.

Camera Director Settings: Αυτό το στοιχείο καθορίζει τις ρυθμίσεις που χρησιμοποιούνται για την κάμερα μετά το όχημα στο ViewPort.

Camera Settings: Το στοιχείο CameraDefaults σε επίπεδο root καθορίζει τις προεπιλογές που χρησιμοποιούνται για όλες τις κάμερες. Αυτές οι προεπιλογές μπορούν να παρακαμφθούν για κάποια συγκεκριμένη κάμερα στο στοιχείο Cameras εντός των στοιχείων Vehicles.

CaptureSettings: Το CaptureSettings καθορίζει τον τρόπο απόδοσης διαφορετικών τύπων εικόνας, όπως σκηνή, βάθος, disparity, κανονικές επιφάνειες και προβολές segmentation.

NoiseSettings: Το NoiseSettings επιτρέπει την προσθήκη θορύβου στον καθορισμένο τύπο εικόνας με στόχο την προσομοίωση θορύβου αισθητήρα κάμερας, παρεμβολών και άλλων αντικειμένων.

Gimbal: Το στοιχείο Gimbal επιτρέπει να παγώσει τον προσανατολισμό της κάμερας για pitch, roll και / ή yaw.

Vehicles Settings: Κάθε λειτουργία προσομοίωσης θα περάσει από τη λίστα των οχημάτων που καθορίζονται σε αυτήν τη ρύθμιση και θα δημιουργήσει αυτά που έχουν το "AutoCreate": true. Κάθε όχημα που καθορίζεται σε αυτήν τη ρύθμιση έχει κλειδί που γίνεται το όνομα του οχήματος. Εάν λείπει το στοιχείο

"Vehicles", τότε αυτή η λίστα συμπληρώνεται με το προεπιλεγμένο αυτοκίνητο με το όνομα "PhysXCar" και το προεπιλεγμένο multirotor με το όνομα "SimpleFlight".

PawnPaths: Αυτό μας επιτρέπει να καθορίσουμε τα δικά μας blueprints του οχήματος, για παράδειγμα, μπορούμε να αντικαταστήσουμε το προεπιλεγμένο αυτοκίνητο στο AirSim με το δικό μας αυτοκίνητο.

LocalHostIp Setting: Όταν συνδεόμαστε σε απομακρυσμένα μηχανήματα, ίσως χρειαστεί να επιλέξουμε έναν συγκεκριμένο προσαρμογέα Ethernet για να φτάσουμε σε αυτά τα μηχανήματα. LocalHostIp μας επιτρέπει να διαμορφώσουμε τον τρόπο με τον οποίο φτάνουμε σε αυτά τα μηχανήματα. Η προεπιλογή του 127.0.0.1 δεν είναι δυνατή η πρόσβαση σε εξωτερικά μηχανήματα, αυτή η προεπιλογή χρησιμοποιείται μόνο όταν όλα περιέχονται σε έναν μόνο υπολογιστή.

ApiServerPort: Αυτή η ρύθμιση καθορίζει τη θύρα διακομιστή που χρησιμοποιείται από τους πελάτες του airsim, η προεπιλεγμένη θύρα είναι 41451. Με τον καθορισμό διαφορετικών θυρών, ο χρήστης μπορεί να εκτελεί παράλληλα πολλαπλά περιβάλλοντα για να επιταχύνει τη διαδικασία συλλογής δεδομένων.

SpeedUnitFactor: Συντελεστής μετατροπής μονάδας για ταχύτητα που σχετίζεται με m / s, η προεπιλογή είναι 1. Χρησιμοποιείται σε συνδυασμό με το SpeedUnitLabel. Αυτό μπορεί να χρησιμοποιηθεί μόνο για σκοπούς προβολής, για παράδειγμα, ταχύτητα στην οθόνη όταν οδηγούμε αυτοκίνητο.

SpeedUnitLabel: Ετικέτα μονάδας για ταχύτητα, η προεπιλογή είναι m / s. Χρησιμοποιείται σε συνδυασμό με το SpeedUnitFactor.

2.2.1.3 AirSim APIs

Το API είναι το αρκτικόλεξο για το Application Programming Interface, το οποίο είναι ένας διαμεσολαβητής λογισμικού που επιτρέπει σε δύο εφαρμογές να μιλούν μεταξύ τους. Το AirSim εκθέτει API για την ανάκτηση δεδομένων και τον έλεγχο οχημάτων με τρόπο ανεξάρτητο από την πλατφόρμα. [5]

Χρησιμοποιώντας τα API που παρέχονται από το AirSim μπορούμε να πάρουμε τον έλεγχο των οχημάτων από το περιβάλλον στο οποίο θα τρέξει η προσομοίωση μας και να τα κατευθύνουμε ανάλογα με τις ανάγκες μας. Υπάρχει πληθώρα API τα οποία παρέχονται.

Αρχικά υπάρχουν τα **confirmConnection**, **enableApiControl** και **armDisarm** τα οποία ελέγχουν την σύνδεση και παίρνουν τον έλεγχο του οχήματος. Το simGetObjectPose παίρνει ως αποτέλεσμα, ή καθορίζει τις τιμές για το όχημα όπως την θέση του το ύψος του την ταχύτητα του και άλλες μετρικές.

Τα **Image / Computer Vision APIs** προσφέρονται για ανάκτηση συγχρονισμένων εικόνων από πολλές κάμερες ορίζοντας τις παραμέτρους από το αρχείο settings.json.

Το **σύστημα συντεταγμένων** του AirSim χρησιμοποιεί το σύστημα συντεταγμένων NED, δηλαδή, το + X είναι Βορράς, το + Y είναι Ανατολή και το + Z είναι Κάτω. Όλες οι μονάδες βρίσκονται σε σύστημα SI. Αυτό διαφέρει από το σύστημα συντεταγμένων που χρησιμοποιείται εσωτερικά από την Unreal Engine. Στο Unreal Engine, το + Z είναι αντί για κάτω και η μονάδα μήκους είναι σε εκατοστά αντί για μέτρα. Τα AirSim API φροντίζουν τις κατάλληλες μετατροπές. Το σημείο εκκίνησης του οχήματος είναι πάντα συντεταγμένες (0, 0, 0) στο σύστημα NED. Έτσι, κατά τη μετατροπή από συντεταγμένες Unreal σε NED, αφαιρούμε πρώτα την αρχική μετατόπιση και μετά κλιμακώνουμε κατά 100 για μετατροπή cm σε m.

Το **GetMultirotorState** επιστρέφει την κατάσταση του οχήματος με μία κλήση. Η κατάσταση περιλαμβάνει, σύγκρουση, εκτιμώμενη κινηματική (δηλαδή κινηματική που υπολογίζεται από αισθητήρες τήξης) και χρονική σήμανση (νανο δευτερόλεπτα από την εποχή). Η κινηματική εδώ σημαίνει 6 ποσότητες: θέση, προσανατολισμό, γραμμική και γωνιακή ταχύτητα, γραμμική και γωνιακή επιτάχυνση. Όλες οι ποσότητες είναι στο σύστημα συντεταγμένων NED, μονάδες SI σε παγκόσμιο πλαίσιο εκτός από τη γωνιακή ταχύτητα και τις επιταχύνσεις.

Τα APIs για το αυτοκίνητο συγκεκριμένα είναι το setCarControls (το οποίο καθορίζει το γκάζι, τιμόνι, χειρόφρενο και αυτόματη ή χειροκίνητη ταχύτητα) και το getCarState το οποίο επιστρέφει τις πληροφορίες με την ίδια δομή όπως και το GetMultirotorState.

Πολλές από τις μεθόδους API έχουν παραμέτρους οι οποίες ονομάζονται duration και max_wait_seconds, και έχουν async ως κατάληξη. Αυτές οι μέθοδοι θα επιστρέψουν αμέσως μετά την έναρξη της εργασίας στο AirSim, έτσι ώστε ο κωδικός πελάτη να μπορεί να κάνει κάτι άλλο κατά την εκτέλεση αυτής της

εργασίας. Αν θέλουμε να περιμένουμε να ολοκληρωθεί αυτή η εργασία, μπορούμε να καλέσουμε το waitOnLastTask ως εξής: client.takeoffAsync().join() Εάν ξεκινήσει μια άλλη εντολή, τότε ακυρώνει αυτόματα την προηγούμενη εργασία και ξεκινά νέα εντολή.

Επιπρόσθετα υπάρχουν τα Collision API το οποίο επιστρέφει μια δομή η οποία περιέχει πληροφορίες για το αν υπήρξε σύγκρουση του οχήματος, Time Of Day API το οποίο είναι ορισμένο να μην αλλάζει όσο περνά η ώρα by default, weather APIs, wind APIs, Lidar APIs, Pause and Continue APIs και recording APIs.

2.2.2 Unreal Engine

Το Unreal Engine (UE4) είναι μια πλήρης σειρά εργαλείων δημιουργίας για ανάπτυξη παιχνιδιών, οπτικοποίηση αρχιτεκτονικής και αυτοκινητοβιομηχανίας, δημιουργία γραμμικού περιεχομένου ταινιών και τηλεόρασης, παραγωγή εκπομπών και ζωντανών εκδηλώσεων, εκπαίδευση και προσομοίωση και άλλες εφαρμογές σε πραγματικό χρόνο. [3]

Το **Unreal Editor Blueprint** είναι ένα εξειδικευμένο είδος κλάσης που προορίζεται για λογική που πρέπει να εκτελέσετε μόνο εντός του Unreal Editor, ποτέ στο χρόνο εκτέλεσης και που δεν χρειάζεται προσαρμοσμένη διεπαφή χρήστη.[4] Σε αυτό το κομμάτι θα αναπτύξουμε την κίνηση του αντικειμένου το οποίο θα αναγνωρίζεται από τα drones.

Το **unreal environment** είναι το περιβάλλον στο οποίο θα τρέξει το AirSim plugin. Υπάρχουν έτοιμα διαθέσιμα περιβάλλοντα [6].

Ο προσομοιωτής AirSim κατά την εγκατάσταση του, έχει έτοιμο περιβάλλον για να τρέξει κάτω από τον φάκελο Unreal/Environments/**Blocks.**

Σε αυτή την πειραματική έρευνα έχουμε χρησιμοποιήσει το περιβάλλον το οποίο είναι ήδη έτοιμο όταν έγινε η εγκατάσταση του AirSim.[9]

2.2.3 Server-Client Αρχιτεκτονική

Ο socket programming είναι ένας τρόπος σύνδεσης δύο κόμβων σε ένα δίκτυο για επικοινωνία μεταξύ τους. Μια υποδοχή (κόμβος) ακούει ένα συγκεκριμένο port σε μια διεύθυνση IP, ενώ μια άλλη υποδοχή φτάνει στην άλλη για να σχηματίσει μια σύνδεση[7]. Στην αρχική υλοποίηση του συστήματος το socket programming έχει χρησιμοποιηθεί για να μπορεί το drone να έρχεται σε επαφή με τον server.

Στο τελικό version του προγράμματος έχει αντικατασταθεί με **Flask HTTP Server[14].**

2.2.4 YOLO v3 machine learning algorithm

Το YOLOv3 (You Only Look Once, Version 3) είναι ένας αλγόριθμος ανίχνευσης αντικειμένων σε πραγματικό χρόνο που προσδιορίζει συγκεκριμένα αντικείμενα σε βίντεο, ζωντανές ροές ή εικόνες. Οι εκδόσεις 1-3 του YOLO δημιουργήθηκαν από τους Joseph Redmon και Ali Farhadi [22]. Στην εργασία αυτός ο αλγόριθμος έχει χρησιμοποιηθεί για την ανίχνευση του αντικειμένου μέσα από φωτογραφίες τις οποίες βγάζει το εκάστοτε drone.

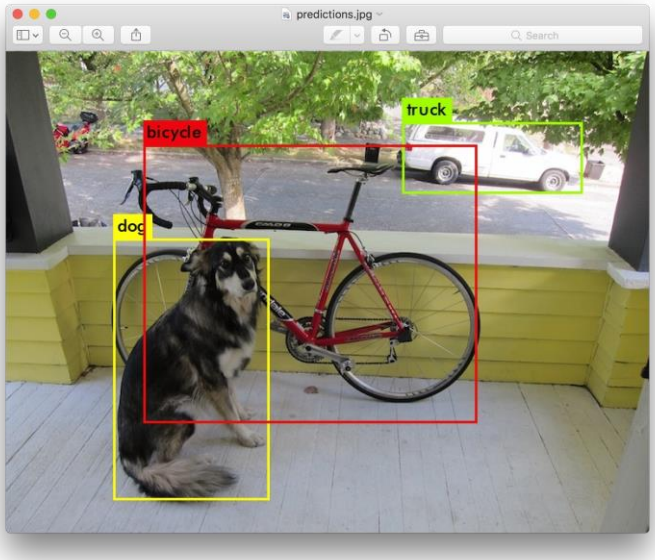


Figure 2.3 Yolo Detection Algorithm

Στο σχήμα 2.3 μπορούμε να διακρίνουμε πως ο αλγόριθμος έχει διακρίνει στην φωτογραφία, μετά την επεξεργασία της, τον σκύλο το ποδήλατο και το αυτοκίνητο.

Χρησιμοποιώντας τον παραπάνω αλγόριθμο είμαστε σε θέση να αναγνωρίσουμε το αντικείμενο το οποίο μας ενδιαφέρει στον χώρο τον οποίο τα drones θα επιβλέπουν. Βάζοντας ως είσοδο ένα από τα αντικείμενα τα οποία ο αλγόριθμος μπορεί να αναγνωρίσει σε μια φωτογραφία, θα μπορέσουμε να εξακριβώσουμε κατά πόσο το εκάστοτε αντικείμενο βρίσκεται στον χώρο.

2.2.5 Docker

Το Docker είναι μια ανοιχτή πλατφόρμα για ανάπτυξη, αποστολή και εκτέλεση εφαρμογών. Το Docker επιτρέπει τον διαχωρισμό των εφαρμογών από την υποδομή, ώστε να μπορείτε να γίνεται γρήγορη παράδοση λογισμικού. Με το Docker γίνεται ο διαχειρισμός της υποδομής με τον ίδιο τρόπο που γίνεται στις εφαρμογές.[23]

Το Containerization είναι μια μέθοδος virtualization, για ανάπτυξη και εκτέλεση εφαρμογών χωρίς την ανάγκη για εκκίνηση ολόκληρων εικονικών μηχανών (VM) στο λειτουργικό σύστημα του κεντρικού υπολογιστή. Τα containers χωρίζουν τους πόρους που διαχειρίζεται το λειτουργικό σύστημα σε απομονωμένες ομάδες.

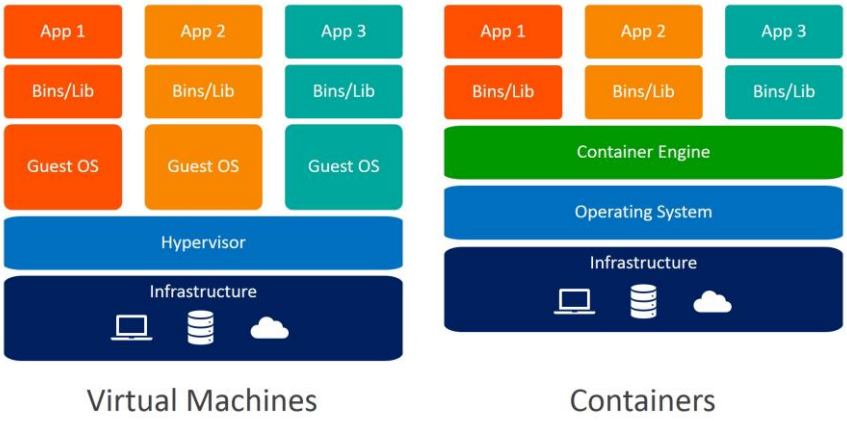


Figure 2.4 Virtual Machines Vs Containers

Η πιο σημαντική διαφορά σε σχέση με άλλες τεχνολογίες virtualization είναι ότι υπάρχει η δυνατότητα να δημιουργηθούν virtual instances τα οποία μπορούν να μοιράζονται το ίδιο λειτουργικό σύστημα και σχετικά binaries, dependencies, drivers ενώ τα application containers για να τρέξουν ένα επιθυμητό λογισμικό χρειάζονται αρχεία, environmental variables, libraries κλπ.

Επειδή τα containers δεν έχουν τα γενικά έξοδα ενός ολόκληρου λειτουργικού συστήματος που απαιτούνται από τα VM για να λειτουργήσουν, το μέγεθός τους είναι μικρότερο από τα VM που τους καθιστά ευκολότερη τη μετεγκατάσταση, ταχύτερη εκκίνηση, απαιτούν λιγότερη μνήμη και ως αποτέλεσμα, είναι δυνατή η εκτέλεση πολλών περισσότερων κοντέινερ στην ίδια υποδομή αντί για VM.

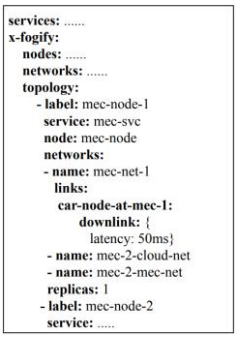
Η πιο δημοφιλής πλατφόρμα container είναι η Docker Engine η οποία έχει κατασκευαστεί πάνω από το linux kernel. Το Docker παρέχει ένα πλήρες σύνολο εργαλείων που δίνει τη δυνατότητα packaging και εκτέλεσης εφαρμογών σε container. Με τη χρήση containers όπως το Docker, όλα τα containers σε έναν

συγκεκριμένο κεντρικό υπολογιστή λειτουργούν κάτω από τον ίδιο πυρήνα, με μόνο application resources απομονωμένα ανά container.

2.2.6 Fogify

Εξομοιωτής ο οποίος διευκολύνει τη μοντελοποίηση της ανάπτυξης και του πειραματισμού μεγάλης κλίμακας fogs και testbeds [8]. Στην εργασία αυτός ο αλγόριθμος έχει χρησιμοποιηθεί για την εξομοίωση και την εξαγωγή συμπερασμάτων σχετικά με το latency μεταξύ του εκάστοτε drone και Server.

Μια τυπική ροή εργασίας ξεκινά με τον χρήστη να επεξεργάζεται το αρχείο dockercompose της εφαρμογής IoT και να το επεκτείνει ώστε να κάνει encapsulate το μοντέλο της Fogify.



2.5 Fogify Topology

Το σχήμα 2.5 απεικονίζει μια συντομευμένη έκδοση της τοπολογίας της επίδειξης. Το μοντέλο Fogify αποτελείται από: (i) Fog Templates, επιτρέποντας την περιγραφή Υπηρεσιών, Κόμβων και Δικτύων. και (ii) την τοπολογία ομίχλης, επιτρέποντας στους χρήστες να καθορίσουν blueprints.

Ένα Blueprint αντιπροσωπεύει μια προσομοιωμένη συσκευή και είναι ένας συνδυασμός κόμβου, υπηρεσίας, δικτύων, αντιγράφων και ετικέτας. Οι υπηρεσίες μεταβιβάζονται από το docker-compose, ενώ η ενότητα x-fogify παρέχει όλα τα πρωτότυπα Fogify. Το Σχήμα 2.5, απεικονίζει ένα σχεδιάγραμμα ενός κόμβου Fog, δηλαδή του mec-node-1, ο οποίος εκτελεί την υπηρεσία mec-csv σε συσκευή mec-node και συνδέεται με το mec-net-1, mex-2-cloud-net και mec-2-mec-net δίκτυα. Ο χρήστης πρέπει να περιγράφει με αυτόν τον τρόπο κάθε κόμβο Fog. Όταν η περιγραφή είναι έτοιμη, ο χρήστης αναπτύσσει την εφαρμογή χρησιμοποιώντας το FogifySDK μέσω ενός Jupyter Notebook, με την περιγραφή που λαμβάνεται από το Fogify Controller. Εάν δεν εντοπιστεί σφάλμα από τον

Controller, δημιουργεί τις προσομοιωμένες συσκευές και δημιουργεί τα δίκτυα επικάλυψης μεταξύ τους, δημιουργεί τις υπηρεσίες και μεταδίδει (οποιουσδήποτε) περιορισμούς δικτύου στους Fogify Agents. Συγκεκριμένα, ο Controller μεταφράζει τις προδιαγραφές του μοντέλου σε υποκείμενα πρωτόγονα ενορχηστρώσεων και τα αναπτύσσει μέσω του Cluster Orchestrator, διασφαλίζοντας την παρουσία των υπηρεσιών που περιέχονται στο περιβάλλον. Βρίσκεται σε κάθε κόμβο συμπλέγματος, το Fogify Agents εκθέτει ένα API για αποδοχή αιτημάτων από τον Ελεγκτή, εφαρμογή αρχικών QoS δικτύου και παρακολούθηση των προσομοιωμένων συσκευών. Σε μια προσομοιωμένη ανάπτυξη που εκτελείται, το Fogify επιτρέπει στους προγραμματιστές να εφαρμόζουν ενέργειες και σενάρια «what-if» στις υπηρεσίες τους IoT, όπως σφάλματα ad-hoc και αλλαγές τοπολογίας. Οι ενέργειες και τα σενάρια γράφονται ακολουθώντας το Fogify Runtime Evaluation Model. Όταν υποβάλλεται μια ενέργεια ή ένα σενάριο, ο Fogify Controller συντονίζει την εκτέλεσή του με το Cluster Orchestrator και τους αντίστοιχους Fogify Agents. Επιπλέον, το Fogify καταγράφει μετρήσεις επιδόσεων και επιπέδου εφαρμογής μέσω της μονάδας παρακολούθησης του Fogify Agent. Όλες οι μετρήσεις αποθηκεύονται στον τοπικό χώρο αποθήκευσης του Agent και μπορούν να ανακτηθούν από το FogifySDK.

Κεφάλαιο 3

Περιγραφή Συστήματος

3.1 Προεπισκόπηση Συστήματος

3.1.1 Αποστολή

Αποστολή του συστήματος στα αρχικά στάδια είναι να μπορεί να αναδείξει μέσα από ένα περιβάλλον προσομοίωσης το πως ένα drone σε ένα οποιοδήποτε περιβάλλον, δίνοντας του κάποια στοιχεία ως είσοδο να μπορεί να εκτελέσει κάποιες οδηγίες τις οποίες λαμβάνει μετά από επικοινωνία με ένα Server.

Συγκεκριμένα να μπορεί μέσα από κάποιες συντεταγμένες οι οποίες του δίνονται, και οι οποίες περιγράφουν ένα χώρο ο οποίος στην ουσία θα είναι ο χώρος παρακολούθησης του drone, να μπορεί να την ακολουθήσει και να παρακολουθήσει εάν υπάρχει αντικείμενο στον χώρο, το οποίο κινείται.

Δεύτερο στάδιο είναι μέσα από αυτή την προσομοίωση να μπορεί να γίνει εξαγωγή κάποιων δεδομένων όπως για παράδειγμα τη διαδρομή την οποία ακολούθησε καθώς επίσης και τα δεδομένα τα οποία συλλέγηκαν από το περιβάλλον.

Στην συνέχεια θα πρέπει το σύστημα να έχει την υποδομή και την δυνατότητα να παράγει σενάρια με περισσότερο από ένα drone και διαδρομές ανάλογα με την είσοδο του χρήστη και να μπορεί να εκτελέσει εξίσου αποτελεσματικά την αρχικά αποστολή.

3.1.2 Περιορισμοί Συστήματος

Οι περιορισμοί του συστήματος αφορούν κυρίως τους περιορισμούς που υπάρχουν στο plugin του AirSim Simulator. Περιορισμοί οι οποίοι υπάρχουν στις κλήσεις των API και συγκεκριμένα στην διαχείριση πολλαπλών drones την ίδια ώρα.

Επίσης ένας άλλος περιορισμός ο οποίος δυσκόλεψε την υλοποίηση του συστήματος ήταν η αδυναμία του συστήματος να γίνει containerized για να μπορέσουν να παράγονται real-time τα δεδομένα από τον προσομοιωτή αφού το πρόγραμμα έχει αναπτυχθεί σε λογισμικό Windows.

3.1.3 Απαιτήσεις συστήματος

Εφόσον το σύστημα το οποίο θα αναπτύξουμε θα είναι σε ένα περιβάλλον προσομοίωσης, τότε οι περιορισμοί αφορούν τα χαρακτηριστικά της μηχανής στην οποία θα τρέξει το σύστημα.

Για παράδειγμα όσο αφορά την μεταφορά και ανάλυση μίας φωτογραφίας από τον αλγόριθμο έχοντας 16GB RAM στην μηχανή μας, σε μια μηχανή με 8GB RAM ίσως πάρει τον διπλάσιο χρόνο. Αυτό έχει επίπτωση στον πρωταρχικό μας στόχο ο οποίος αφορά το real-time response του συστήματος.

Με την ίδια λογική αν θέλουμε στο fogify να τρέξουμε 2 servers με 8GB RAM το καθένα σε ένα μηχάνημα με 4GB RAM δεν μπορούμε να το κάνουμε λόγω περιορισμού της μηχανής.

3.1.4 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική του συστήματος μπορεί να περιγραφεί με μερικά απλά βήματα τα οποία μπορεί ο χρήστης να εκτελέσει ούτως ώστε να τρέξει το σύστημα.

Τα βήματα αφαιρετικά χωρίζονται στα εξής:

- User Input
- Προσομοίωση

- Εξαγωγή μετρικών και αποτελεσμάτων
- Emulation
- Εκτίμηση Απόδοσης Συστήματος

User Input:

Αρχικά ο χρήστης θα χρειαστεί να εισάγει κάποια στοιχεία τα οποία χρειάζεται το σύστημα για να τρέξει. Στον υποφάκελο Documents βρίσκεται το αρχείο editSettings.py το οποίο θα ζητήσει τον αριθμό των μονοπατιών τα οποία θα ακολουθήσουν τα εκάστοτε drones. Ό,τι αριθμός δοθεί σε αυτό το σημείο τόσα drones θα υλοποιηθούν στο αρχείο settings.json και θα κάνουν spawn στο περιβάλλον προσομοίωσης, και ο ίδιος αριθμός μονοπατιών θα δημιουργηθεί στο αρχείο paths.json. Για την προσαρμογή των μονοπατιών του κάθε drone μπορεί να γίνει και χειροκίνητα μπαίνοντας στο αρχείο paths.json.

Προσομοίωση:

Σε αυτό το σημείο είμαστε σε θέση να τρέξουμε την προσομοίωση. Ανοίγοντας το περιβάλλον Blocks (AirSim/Unreal/Environments/Blocks/blocks.uproject) θα ανοίξει το περιβάλλον προσομοίωσης στο εργαλείο Unreal Engine. Πατώντας το play ξεκινά το περιβάλλον να τρέχει και μπορούμε να διακρίνουμε το κάθε drone στην αρχική του θέση, σε στάση disarmed. Ταυτόχρονα θα ξεκινήσουμε το αρχείο server.py που θα τρέχει στο background και θα αναλύσει την κάθε εικόνα από τα drones. Στην συνέχεια θα τρέξουμε το αρχείο drone Controller το οποίο βρίσκεται στον υποφάκελο AirSim/PythonClient/multirotor/ . Ξεκινώντας το αρχείο drone controller θα μας ζητηθεί ο αριθμός των μονοπατιών τα οποία θα τρέξουν και θα δώσουμε τον ίδιο αριθμό με αυτόν που δώσαμε στο αρχείο editSettings.py. Στην συνέχεια θα μας ζητηθεί επίσης η τοποθεσία του αρχείου path.json.

Για παράδειγμα εάν το αρχείο μας βρίσκεται στα documents κάτω από το φάκελο AirSim τότε θα δώσουμε ως είσοδο: C:\Users\User\Documents\AirSim\path.json

Μετά από αυτό το σημείο ξεκινά η προσομοίωση η οποία μπορεί να παρακολουθηθεί και ζωντανά βλέποντας το παράθυρο του Unreal Engine.

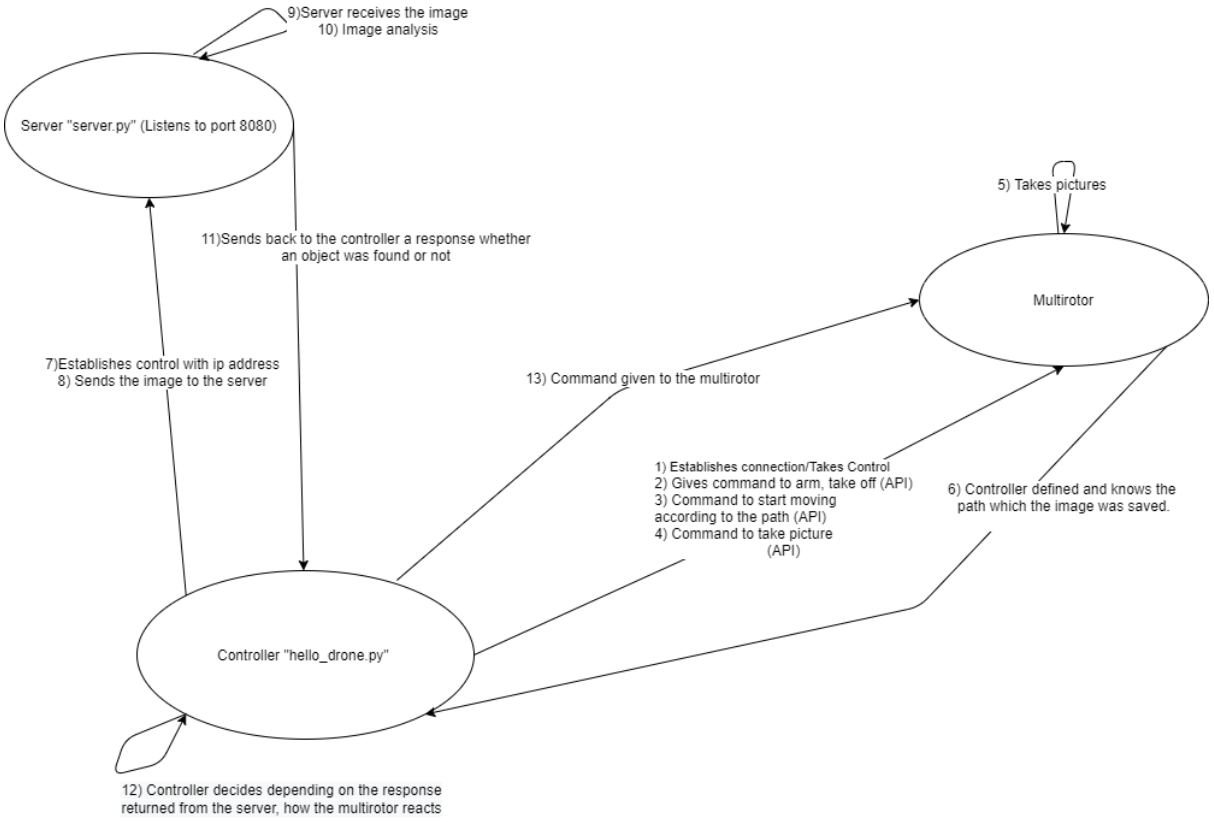


Figure 3.1 Components Interaction Diagram

Στο σχήμα 3.1 μπορούμε να δούμε πως τα υποσυστήματα αλληλοεπιδρούν μεταξύ τους για το αποτέλεσμα το οποίο μας ενδιαφέρει από την προσομοίωση. Σε όλα τα στάδια ο αριθμός των drones μπορεί να είναι από ένα μέχρι πολλά, ακολουθώντας την ίδια δομή φέρνει τα ίδια επιθυμητά αποτελέσματα.

Αρχικά ξεκινούμε το hello_drone.py το οποίο φορτώνει τον κώδικα ο οποίος θα εκτελεστεί από τα drones. Παίρνει τον έλεγχο των Drones και δίνει εντολή για take off μέσω κλήσης API. Στην συνέχεια ξεκινά η εκτέλεση των διαδρομών οι οποίες είναι αποθηκευμένες στο json file path.json. Καθ' όλη την διάρκεια της κίνησης των drones, τα drones συλλέγουν φωτογραφίες της πτήσης τους.

Κάθε φωτογραφία αποθηκεύεται και αμέσως μετά όταν γίνει η σύνδεση με τον server, ο server λαμβάνει την φωτογραφία και εκτελεί την ανάλογη επεξεργασία εικόνας.

Όταν γίνει η επεξεργασία εικόνας ο server επιστρέφει πίσω κάποια απάντηση η οποία καθορίζει κατά πόσο το drone έχει φωτογραφίσει κάποιο αντικείμενο το οποίο ψάχνουμε ή όχι.

Ανάλογα με το αν έχει εντοπιστεί το αντικείμενο ή όχι το hello_drone καθορίζει την επόμενη κίνηση του drone.

Όταν τελειώσει η προσομοίωση το αρχείο hello_drone.py αποθηκεύει ένα αρχείο traces.yaml το οποίο περιέχει στοιχεία σχετικά με την προσομοίωση η οποία έχει εκτελεστεί. Το αρχείο traces έχει την μορφή την οποία χρειάζεται το Fogify για να μπορεί να το πάρει ως είσοδο και να γίνει το emulation με τον ίδιο τρόπο στο fogify όπως έγινε στο simulation. Επίσης οι φωτογραφίες οι οποίες λήφθηκαν από τα drones είναι ήδη αποθηκευμένες στην μηχανή η οποία έχει γίνει η εκτέλεση της προσομοίωσης για να χρησιμοποιηθούν από το fogify.

Εξαγωγή Μετρικών και Αποτελεσμάτων:

Μετά το τέλος της προσομοίωσης τα αποτελέσματα και οι μετρικές έχουν ήδη αποθηκευτεί στην μηχανή μας. Συγκεκριμένα υπάρχει το αρχείο traces.yaml το οποίο έχει αποθηκευμένα τις τροχιές του κάθε drone σε κάθε σημείο της διαδρομής μαζί με άλλες μετρικές και το οποίο by default δημιουργείται στο desktop της μηχανής. Επίσης για το κάθε drone έχει δημιουργηθεί ένας φάκελος στην τοποθεσία C:\Users\user\AppData\Local\Temp\airsim. Για παράδειγμα για το drone1 έχει δημιουργηθεί ο φάκελος με ονομασία 1 κ.ο.κ. Αυτά τα αποτελέσματα μπορούν να χρησιμοποιηθούν ξανά χωρίς την χρήση του unreal engine και του airsim, και θα μας δώσουν τον ίδιο χρόνο απόκρισης, τον χρόνο εντοπισμό του αντικειμένου και τις υπόλοιπες μετρικές και αποτελέσματα όπως βγήκαν αρχικά από το airsim και unreal engine.

Emulation:

Έχοντας δημιουργήσει τις εφαρμογές και τα docker images για να αναπαράξουμε τα δεδομένα και την εκτέλεση του αλγορίθμους μηχανικής μάθησης καθώς και τα αρχεία εικόνων και τροχιών, τώρα είμαστε έτοιμοι να αξιολογήσουμε την επίδοσή τους με τη χρήση του Fogify. Στα πλαίσια αυτής της εργασίας, χρησιμοποιήσαμε μια επέκταση του Fogify η οποία ενθυλακώνει λειτουργίες για προσομοίωσης κινητών κόμβων, όπως drones. Η εγκατάσταση του Fogify έγινε σε μία εικονική μηχανή στο κέντρο δεδομένων του εργαστηρίου η οποία είχε 16 πυρήνες στα 2.4Ghz και 16 GB μνήμη. Εν κατακλείδι, μεταφέραμε όλα τα αρχεία που χρειάζεται για την εκτέλεση των πειραμάτων στην εικονική μηχανή, όπως, εικόνες από τη προσομοίωση, τροχιές των drones, docker images.

Εκτίμηση Απόδοσης Συστήματος:

Μετά το emulation γίνεται η εξαγωγή διαφόρων αποτελεσμάτων τα οποία μπορούν να συγκριθούν μεταξύ τους. Αρχικά έγινε εκτέλεση προσομοίωσης για να μετρήσουμε την απόδοση του συστήματος μας για να δούμε αν και πως η επεξεργαστική δύναμη του σταθμού βάσης επηρεάζει το χρόνο εκτέλεσης του μοντέλου μηχανικής μάθησης. Επίσης δοκιμάσαμε διαφορετικά μαθηματικά μοντέλα για ασύρματο δίκτυο για να δούμε πως επηρεάζει το καθένα την επίδοση της εφαρμογής. Εκτός από τις αλλαγές στις παραμέτρους του περιβάλλοντος, διαλέξαμε να αλλάξουμε και παραμέτρους στην ίδια την εκτέλεση του προγράμματος. Συγκεκριμένα αλλάξαμε την εκτέλεση της αποστολής εικόνων στον εξυπηρετητή. Μια ακόμα παράμετρος που αλλάξαμε είναι η συμπίεση των εικόνων από την υπηρεσία που τρέχει πάνω στο drone. Συγκεκριμένα, στη βασική υλοποίηση του αλγορίθμου, πριν στείλει το drone τη φωτογραφία στο σταθμό βάσης την συμπιέζει για να μειώσει τον όγκο των δεδομένων που μεταφέρονται στο δίκτυο.

3.2 Λειτουργίες Συστήματος

Το σύστημα αποτελείται από τα εξής στοιχεία:

1) Περιβάλλον στο οποίο θα τρέξει η προσομοίωση
2) AirSim Plugin
3) Settings.json
4) Επιπρόσθετες Ρυθμίσεις
5) Server
6) Drone Controller
7) Αποτελέσματα προσομοίωσης

Στο επόμενο στάδιο το σύστημα τρέχει χρησιμοποιώντας τα στοιχεία:

1) Workload Generator
2) Server Docker
3) Fogify

Περιβάλλον:

Το φυσικό περιβάλλον στο οποίο θα τρέξει το σύστημα μας τρέχει στο Unreal Engine. Το περιβάλλον έχει ως προαπαιτούμενο τον χώρο που ουσιαστικά θα γίνει η παρακολούθηση του από το drone. Στην δική μας περίπτωση είναι το περιβάλλον blocks του AirSim. Επίσης στο περιβάλλον πρέπει να υπάρχει το αντικείμενο το οποίο θα εντοπιστεί από το drone κατά την διάρκεια της πτήσης του.

AirSim Blocks:

Κατά την εγκατάσταση του AirSim Simulator στην μηχανή την οποία θα γίνει η προσομοίωση, ο προσομοιωτής AirSim έχει ήδη εγκατεστημένο το περιβάλλον Blocks το οποίο είναι έτοιμο να τρέξει μόλις γίνει η εγκατάσταση.[9]

AirSim Plugin

Το AirSim Plugin θα εισαχθεί στο περιβάλλον ως Plugin και θα μπορεί όταν ξεκινά το περιβάλλον να εμφανίζεται μέσα. Για να εισαχθεί το plugin στο environment χρειάζονται κάποιες ρυθμίσεις οι οποίες μπορούν να επεξεργαστούν στο αρχείο settings.json όπως θα δούμε στο επόμενο κεφάλαιο.

Settings.json

Το αρχείο settings.json αρχικοποιείται κατά την πρώτη εκκίνηση το AirSim χωρίς ρυθμίσεις στα Documents. Μέσα από το συγκεκριμένο αρχείο γίνεται η αρχικοποίηση των οχημάτων τα οποία θα τρέξουν στο περιβάλλον και διάφορες ρυθμίσεις σχετικά με αυτά. Όταν γίνει αλλαγή στο αρχείο settings.json πρέπει να γίνει επανεκκίνηση του περιβάλλοντος ούτως ώστε να εφαρμοστούν οι αλλαγές στο περιβάλλον και το πως εμφανίζονται τα drones στο περιβάλλον.

Επιπρόσθετες Ρυθμίσεις

Πέραν από το αρχείο settings.json χρειάζονται επιπρόσθετες ρυθμίσεις ούτως ώστε το σύστημα να τρέχει κανονικά στο περιβάλλον το οποίο επιθυμούμε. Αρχικά θα χρειαστεί να καθοριστεί ο αριθμός των drones και ο αριθμός των διαδρομών οι οποίες θα ακολουθήσουν τα drones στο εκάστοτε περιβάλλον.

Ένα αρχείο path.json έχει δημιουργηθεί το οποίο ανάλογα με τα drones τα οποία θα εισαχθούν στο περιβάλλον, θα παράξει τον ανάλογο αριθμό διαδρομών για τα drones να ακολουθήσουν.

Επίσης μαζί με τα 2 json αρχεία έχει δημιουργηθεί το αρχείο editSettings.py το οποίο παίρνει ως είσοδο από τον χρήστη τον αριθμό των drones τα οποία χρειάζονται και επεξεργάζεται τα αρχεία settings.json και path.json παράγοντας τον κατάλληλο αριθμό drones και διαδρομές τα οποία χρειάζονται.

Server

Ο server χρησιμοποιεί τον αλγόριθμο YOLO object detection. Πριν την εκκίνηση της προσομοίωσης ξεκινά ο server να ακούει σε ένα port και περιμένει το εκάστοτε drone να επικοινωνήσει μαζί του. Όταν γίνει η επικοινωνία ο server περιμένει από το drone να πάρει ως μήνυμα μια φωτογραφία την οποία ο server θα αναλύσει χρησιμοποιώντας τον αλγόριθμο YOLO και θα δώσει απάντηση πίσω στο drone κατά πόσο υπάρχει κάποιο αντικείμενο στον χώρο τον οποίο βρίσκεται το drone ή όχι. Ο server χρησιμοποιεί τρία αρχεία τα οποία βοηθούν στον εντοπισμό του αντικειμένου. Τα αρχεία αυτά είναι:

- Το αρχείο coco.names το οποίο περιέχει τα ονόματα από τα αντικείμενα τα οποία έχει την δυνατότητα ο αλγόριθμος να εντοπίσει.
- Το αρχείο yolov3.cfg το οποίο είναι το configuration file του αλγορίθμου
- Το αρχείο yolov3.weights το οποίο περιέχει τα προ-εκπαιδευμένα βάρη τα οποία βοηθούν στον εντοπισμό του αντικειμένου.

Drone Controller

Σε αυτό το σημείο το σύστημα είναι έτοιμο να ξεκινήσει την προσομοίωση. Κατά την προσομοίωση εκτελείται το αρχείο hello_drone.py το οποίο ελέγχει τα εκάστοτε drones τα οποία αρχικοποιήθηκαν από τις προηγούμενες λειτουργίες. Στο τέλος της εκτέλεσης του αρχείου, παράγεται ένα αρχείο traces.yaml το οποίο περιέχει σημαντικές πληροφορίες σχετικά με τις διαδρομές που ακολούθησαν τα drones. Επίσης αποθηκευμένα στη μηχανή που έτρεξε την προσομοίωση υπάρχουν και όλες οι φωτογραφίες από τα drones τα οποία συμμετείχαν στην προσομοίωση.

Workload Generator

Αυτή η λειτουργία προσομοιώνει την εργασία η οποία εκτελέστηκε πιο πριν από το Unreal Engine Blocks Environment σε συνδυασμό με το AirSim Plugin και τον Server. Δίνοντας ως είσοδο τις φωτογραφίες από την προηγούμενη προσομοίωση σε συνδυασμό με το αρχείο traces.yaml, τρέχοντας το αρχείο simulation_script.py, επικοινωνεί με τον server βγάζοντας τα ίδια αποτελέσματα με την προηγούμενη προσομοίωση.

Server Docker και Fogify

Έχουν χρησιμοποιηθεί για το emulation. Μετά από την εξαγωγή των traces και των φωτογραφιών από την προσομοίωση είμαστε σε θέση όπως είδαμε και με τον workload generator, να παράξουμε τα ίδια αποτελέσματα χωρίς την χρήση του AirSim και Unreal Engine. Έτσι δημιουργώντας τις εφαρμογές και τα docker images αναπαράξαμε τα δεδομένα, την εκτέλεση του αλγορίθμου μάθησης, τα αρχεία εικόνων και τροχιών και αξιολογήσαμε την επίδοση του χρησιμοποιώντας το Fogify.

3.3 Interfaces

Περιβάλλον Blocks

Κατά την εκκίνηση του περιβάλλον blocks.uproject που βρίσκεται κάτω από την διεύθυνση AirSim/Unreal/Environments/Blocks ανοίγει η εφαρμογή Unreal Editor στην οποία μπορούμε να επεξεργαστούμε τον περιβάλλον Blocks.

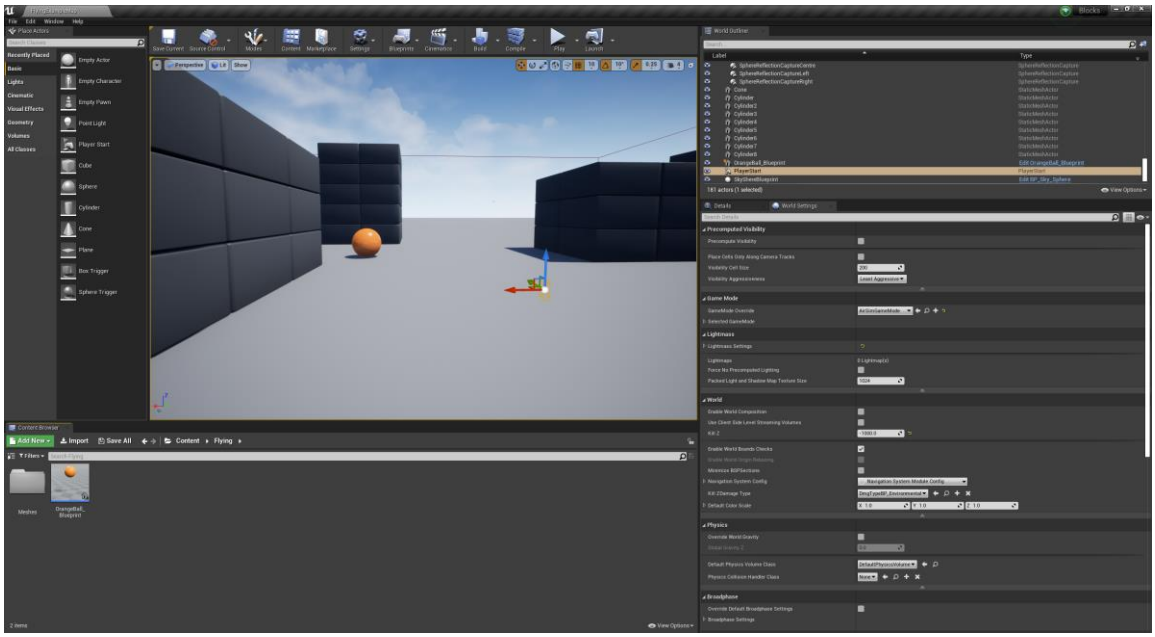


Figure 3.2 Unreal Editor Blocks Environment

Όπως φαίνεται και στο πιο πάνω σχήμα κατά την εκκίνηση του αρχείου blocks.uproject ανοίγει το πρόγραμμα unreal editor το οποίο εμφανίζει το περιβάλλον στο οποίο θα γίνει η προσομοίωση. Στα δεξιά του παραθύρου βλέπουμε τα αντικείμενα που βρίσκονται στον χώρο και ανάλογες ρυθμίσεις για τα αντικείμενα.

Στην εργασία χρειαζόμαστε ένα αντικείμενο το οποίο θα κινείται σε κάποιο χώρο ούτως ώστε να γίνει η κατάλληλη αναγνώριση από τα drones. Όπως φαίνεται από το σχήμα υπάρχει ήδη αντικείμενο στον χώρο το οποίο είναι ευδιάκριτο και μπορεί να χρησιμοποιηθεί ως το σχήμα προς αναγνώριση από τα drones.

Orange Ball Blueprint

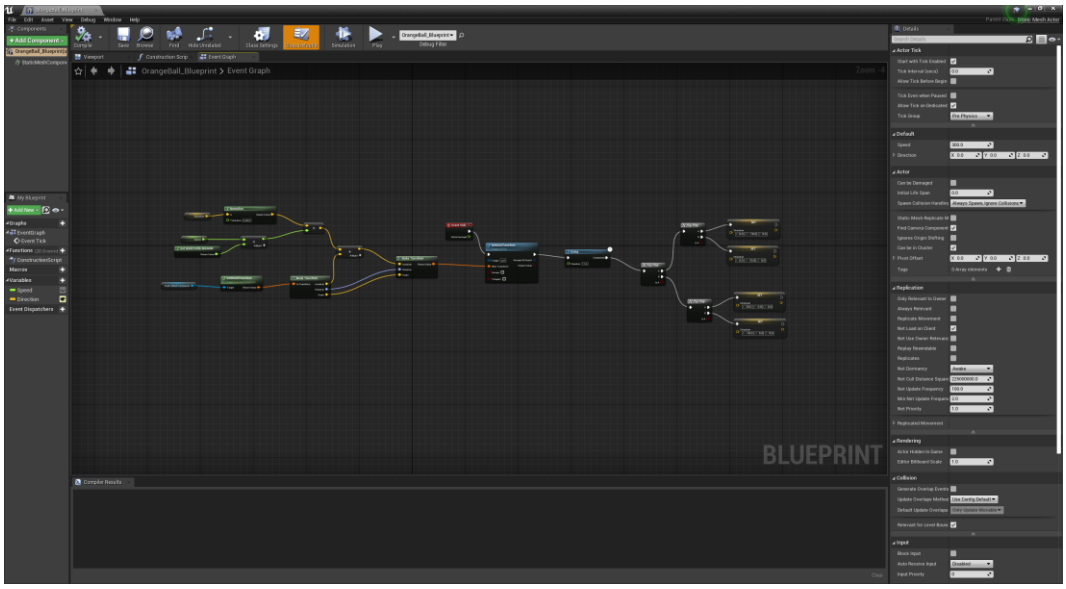


Figure 3.3 Orange Ball Blueprint

Στο πιο πάνω σχήμα διακρίνουμε το Blueprint του αντικειμένου που αναφερθήκαμε στο σχήμα 3.2. Σε αυτή την διεπαφή θα γίνει η ανάπτυξη του script το οποίο θα δίνει στο αντικείμενο την κίνηση γύρω από κάποιο χώρο. Συγκεκριμένα αυτό που μας ενδιαφέρει σε αυτό το σημείο όπως βλέπουμε και στο σχήμα είναι το παράθυρο event graph το οποίο ασχολείται με την κίνηση του αντικειμένου.

Edit Path File

Η διεπαφή σε αυτό το σημείο αφορά την επεξεργασία των αρχείων settings.json και path.json. Το αρχείο editSettings.py έχει επεξεργαστεί στο Visual Studio IDE και εκτελείται από την γραμμή εντολής.

Drone Controller

Η επεξεργασία του κώδικα στο αρχείο hello_drone.py έχει γίνει στο Visual Studio IDE ενώ η διεπαφή στην οποία αλληλοεπιδρά ο χρήστης κατά την εκτέλεση του αρχείου είναι η κονσόλα της Anaconda3/python.exe.

Workload Generator

Η επεξεργασία του κώδικα στο αρχείο simulation_script.py έχει γίνει στο Visual Studio IDE ενώ η διεπαφή στην οποία αλληλοεπιδρά ο χρήστης κατά την εκτέλεση του αρχείου είναι στο Command Line Interface.

Server

Η επεξεργασία του κώδικα στο αρχείο server.py έχει γίνει στο Visual Studio IDE ενώ η διεπαφή στην οποία αλληλοεπιδρά ο χρήστης κατά την εκτέλεση του αρχείου είναι στο Command Line Interface.

Fogify

Στα πλαίσια αυτής της εργασίας, χρησιμοποιήσαμε μια επέκταση του Fogify η οποία ενθυλακώνει λειτουργίες για προσομοίωσης κινητών κόμβων, όπως drones. Η εγκατάσταση του Fogify έγινε σε μία εικονική μηχανή στο κέντρο δεδομένων του εργαστηρίου η οποία είχε 16 πυρήνες στα 2.4Ghz και 16 GB μνήμη. Εν κατακλείδι, μεταφέραμε όλα τα αρχεία που χρειάζεται για την εκτέλεση των πειραμάτων στην εικονική μηχανή, όπως, εικόνες από τη προσομοίωση, τροχιές των drones, docker images. Στη παρακάτω εικόνα φαίνεται το γραφικό περιβάλλον Jupyter Notebook που είναι η βασική διεπαφή του Fogify με το χρήστη.

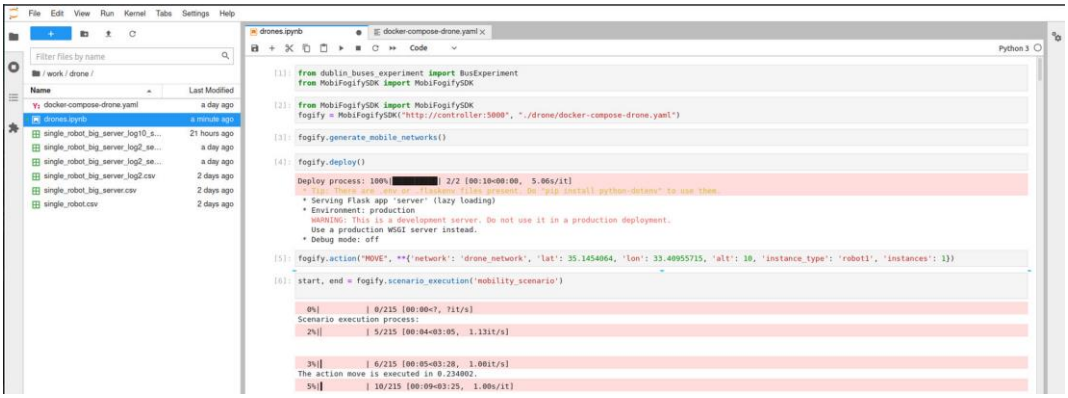


Figure 3.4 Fogify Interface

Όπως φαίνεται στην εικόνα 3.4, αριστερά βρίσκονται τα βασικά αρχεία που χρειάζεται το Fogify για να τρέξει, συγκεκριμένα ένα docker-compose αρχείο που έχει την περιγραφή της εφαρμογής μαζί με το μοντέλο του Fogify, και ένα ipynb αρχείο που έχει το κώδικα ο οποίος θα τρέξει από το jupyter. Επίσης στη συγκεκριμένη εικόνα υπάρχουν και οι μετρήσεις σε μορφή csv όπως τις εξαγάγαμε μετά την εκτέλεση των διαφόρων πειραμάτων. Όσον αφορά το κώδικα της εικόνας, στη πρώτη γραμμή εισάγουμε την βιβλιοθήκη MobiFogifySDK που είναι επέκταση του FogifySDK, μιας βιβλιοθήκης που χρησιμοποιείται για να επικοινωνεί το γραφικό με το Fogify. Στην επόμενη γραμμή αρχικοποιούμε το

Fogify, και δίνουμε ως παράμετρο το αρχείο docker-compose με το μοντέλο του Fogify που έχουμε επεκτύνει με τις τροχιές όπως αυτές παρήχθησαν κατα την προσομοίωση στο AirSim. Στην επόμενη γραμμή(3) παράγονται οι απαραίτητες μεταβλητές για τα δίκτυα με κινούμενους κόμβους. Στη γραμμή 4 στέλνουμε το μοντέλο στο Fogify και όταν η μπάρα φορτώσει, έχουμε τη προσομοίωση να εκτελείται. Στη γραμμή 5 έχουμε την εκτέλεση εντολής για μετακίνηση του drone σε συγκεκριμένη θέση, και στη γραμμή 6 εκτελείται ένα σενάριο (mobility_scenario). Πρακτικά το σενάριο είναι οι τροχιές που έχουμε παράξει, και καθώς εκτελείται το drone είναι σαν να μετακινείται μέσα στον εικονικό χώρο της προσομοίωσης. Η ποιότητα της σύνδεση του drone με τον εξυπηρετητή επηρεάζεται από την απόστασή τους βάση μαθηματικών συναρτήσεων, συγκεκριμένα το Fogify έχει τρία μοντέλα, συγκεκριμένα, γραμμικό, λογαριθμικό με βάση το 2 και λογαριθμικό με βάση το 10. Κατα την συγγραφή του Fogify μοντέλου, για να μπορέσουν να λειτουργήσουν αυτές οι συναρτήσεις, ο χρήστης πρέπει να δώσει τα καλύτερα και τα χειρότερα χαρακτηριστικά που έχει η σύνδεση στην κοντινότερη απόσταση (0) και στην πιο μακρινή απόσταση (ακτίνα του ασύρματου δικτύου). Ως καλύτερα χαρακτηριστικά ορίσαμε 1ms καθυστέρηση δικτύου και 11MBps και ως χειρότερα χαρακτηριστικά 50ms και 1MBps. Οι παρακάτω εικόνες 3.4 και 3.5, δείχνουν τα χαρακτηριστικά του δικτύου και τις τροχιές των drones αντίστοιχα.

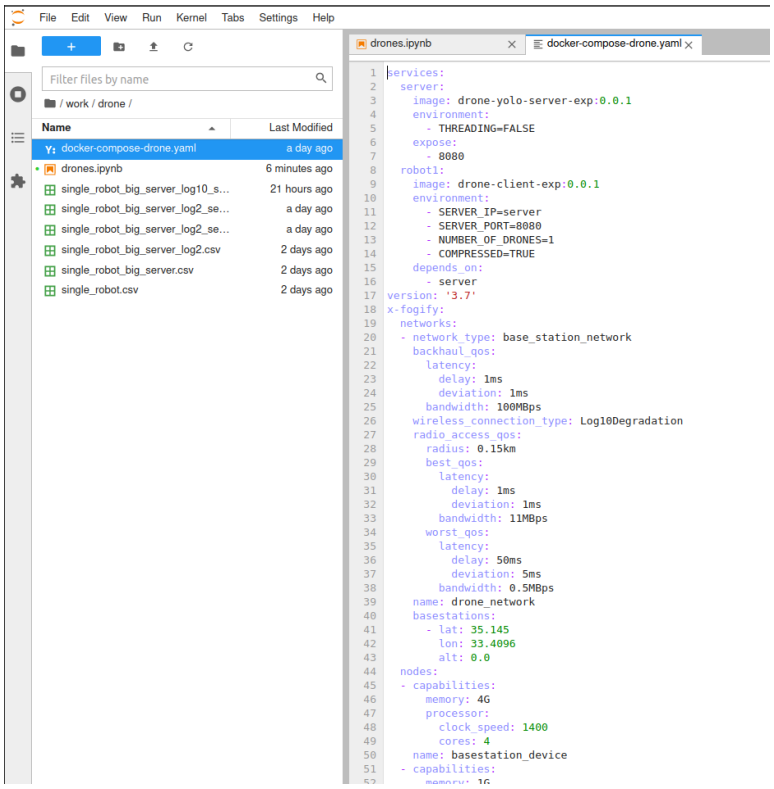


Figure 3.5 Fogify Network Characteristics Interface

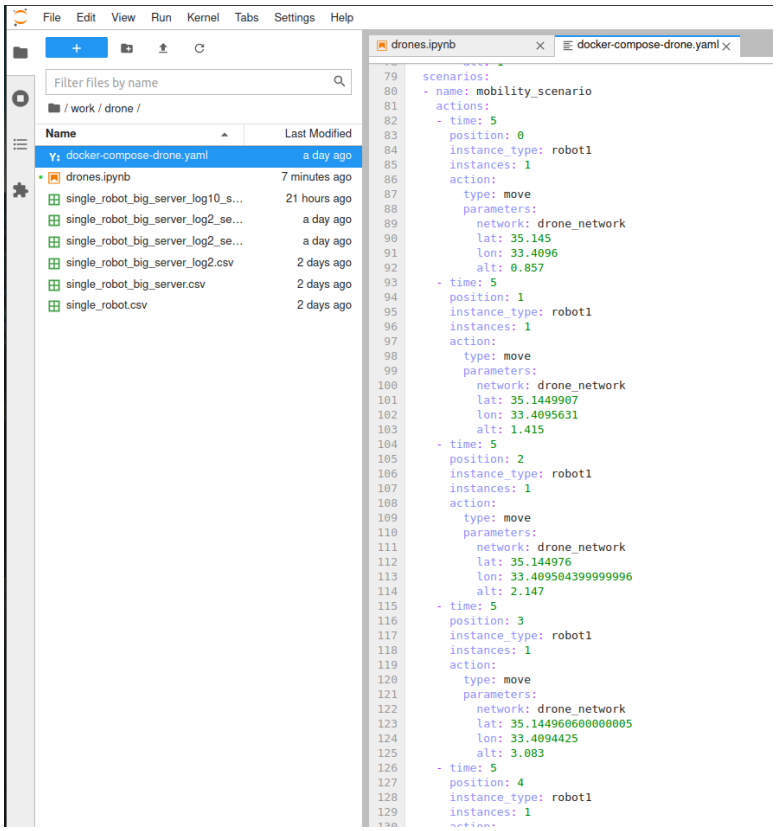


Figure 3.6 Fogify Drone Traces Interface

Κεφάλαιο 4

Υλοποίηση Συστήματος

4.1 Ρυθμίσεις Συστήματος

Για την έναρξη της προσομοίωσης και της ομαλής διεξαγωγής της, πρώτα χρειάζονται μερικές ρυθμίσεις στα αρχεία τα οποία παίρνει το AirSim σαν είσοδο ούτως ώστε να μπορέσει να τρέξει με τα δεδομένα που εμείς χρειαζόμαστε για το σύστημα.

Αυτό μπορεί να γίνει μέσα από το αρχείο editSettings.py το οποίο χειρίζεται τα αρχεία που παίρνει το αρχείο του Drone για να τρέξει. Τα αρχεία τα οποία δέχονται επεξεργασία είναι τα αρχεία settings.json και path.json. Το αρχείο settings. json όπως έχουμε προαναφέρει είναι το αρχείο το οποίο παίρνει το AirSim για να αρχικοποιήσει διάφορα options τα οποία μπορούμε να δούμε και στο επόμενο σχήμα. Το αρχείο path.json είναι το αρχείο το οποίο θα περιέχει τις τροχιές οι οποίες θα ακολουθήσει το κάθε drone.

Schema: <No Schema Selected>

Figure 4.1 Settings JSON File

Όπως φαίνεται στο σχήμα 4.1 υπάρχουν οι ρυθμίσεις οι οποίες αφορούν την προσομοίωση του AirSim. Συγκεκριμένα το AirSim υποστηρίζει πολλές ρυθμίσεις οι οποίες μπορούν να αλλάξουν με την επεξεργασία αυτού του αρχείου. Όπως φαίνεται πιο πάνω ο τύπος της προσομοίωσης έχει οριστεί σε SimMode:Multirotor. Στην συνέχεια διακρίνουμε κάτω από την κατηγορία Vehicles ότι υπάρχει ήδη ένα όχημα με το όνομα "robot1".

Κάτω από αυτό το όχημα μπορούν να οριστούν διάφορες ρυθμίσεις σχετικά με το όχημα. Αυτό που μας ενδιαφέρει σχετικά με το drone είναι οι φωτογραφίες οι οποίες θα λαμβάνονται από το drone καθώς επίσης και η αρχική τοποθεσία του drone.

Available ImageType Values

```
Scene = 0,
DepthPlanar = 1,
DepthPerspective = 2,
DepthVis = 3,
DisparityNormalized = 4,
Segmentation = 5,
SurfaceNormals = 6,
Infrared = 7
```

Figure 4.2 Image Types

Το όχημα υποστηρίζει διάφορους τύπους φωτογραφίας [10] όπως βλέπουμε και από το σχήμα 4.2. Ο τύπος φωτογραφίας που μας ενδιαφέρει είναι ο τύπος Scene = 0.

Όπως φαίνεται στο σχήμα 4.1 ο τύπος φωτογραφίας έχει οριστεί με ανάλυση 2160x1080 pixels. Η ίδια ανάλυση έχει καθοριστεί για όλα τα drones τα οποία παράγονται και συμμετέχουν στην προσομοίωση.

Το δεύτερο σημείο το οποίο είναι σημαντικό σχετικά με την αρχικοποίηση των drones είναι η θέση στην οποία θα ξεκινήσουν κατά την εκκίνηση του περιβάλλοντος. Όπως φαίνεται στο σχήμα 4.1 αυτό το σημείο μπορεί να καθοριστεί στα σημεία X, Y, Z.

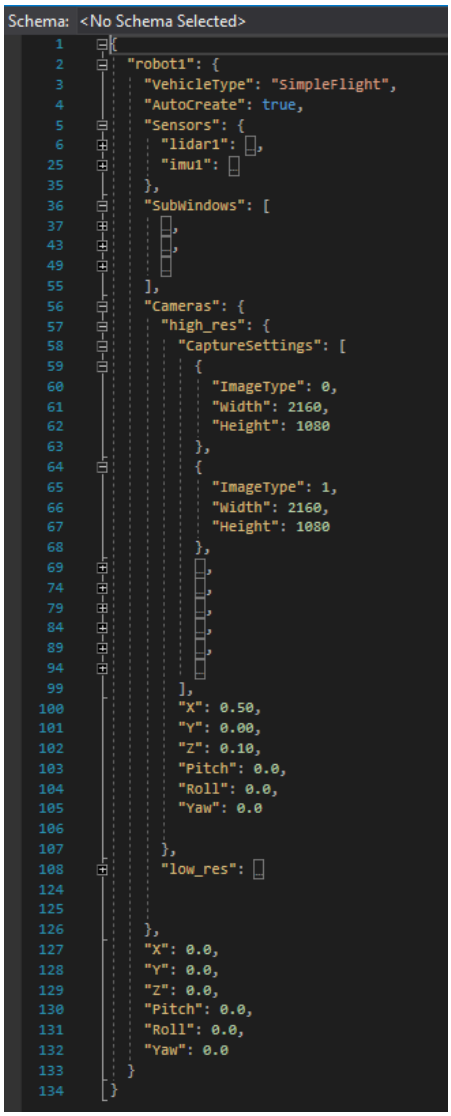


Figure 4.3 Add Robot JSON File

Το αρχείο addRobot.json που απεικονίζεται στο σχήμα 4.3, δείχνει την αρχικοποίηση του “robot1” και τα υπόλοιπα drones αρχικοποιούνται με βάση τον πιο πάνω κώδικα. Το συγκεκριμένο αρχείο βρίσκεται στον ίδιο φάκελο μαζί με τα αρχεία settings.json και editSettings.py.

Μαζί με αυτά τα αρχεία βρίσκεται και το αρχείο path.json. Το συγκεκριμένο αρχείο περιέχει τις διαδρομές τις οποίες θα ακολουθήσουν τα drones κατά την έναρξη της προσομοίωσης.

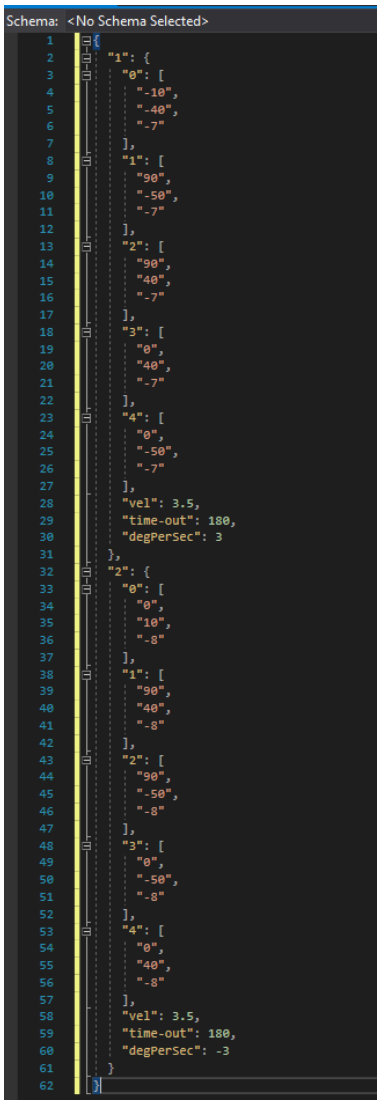


Figure 4.4 Path JSON File

Όπως φαίνεται στο σχήμα 4.4 έχουν αρχικοποιηθεί δύο διαδρομές που μας ενδιαφέρουν στο περιβάλλον Blocks. Οι συγκεκριμένες διαδρομές είναι αντίθετες μεταξύ τους και παρακολουθούν τον ίδιο χώρο. Η πρώτη διαδρομή έχει ονομαστεί «1» και η δεύτερη συνεπώς έχει ονομαστεί «2».

Οι διαδρομές ακολουθούν σύστημα συντεταγμένων που ξεκινά από το 0, 0. Η Τρίτη συντεταγμένη είναι το ύψος το οποίο θα βρίσκεται το drone σε κάθε θέση. Το ύψος εισάγεται με αρνητικό πρόσημο δηλ. -8 θα βρίσκεται 8 μέτρα πάνω από το σημείο 0 που είναι το πάτωμα ως σημείο αναφοράς. Οι διαδρομές οι οποίες δίνονται στο drone είναι μια λίστα 5 σημείων, κάτι το οποίο μπορεί να τύχει επεξεργασίας αναλόγως του σεναρίου του οποίου μας δίνεται, καθώς επίσης και του περιβάλλοντος στο οποίο θα γίνει η προσομοίωση του συστήματος.

Στην περίπτωση που το σενάριο θα τρέξει στο περιβάλλον blocks τα σημεία που έχουμε δώσει για την διαδρομή 1 είναι τα σημεία:

Διαδρομή 1: [(-10,-40,-7),(90,-50,-7),(90,40,-7),(0,40,-7),(0,-50,-7)]

Διαδρομή 2: [(0,10,-8),(90,40,-8),(90,-50,-8),(0,-50,-8),(0,40,-8)]

Το στοιχείο vel αντιπροσωπεύει την ταχύτητα με την οποία το drone θα ακολουθήσει την διαδρομή σε μορφή m/s. Το time-out αντιπροσωπεύει τον χρόνο τον οποίο θα πάρει στο drone για να διανύσει την εκάστοτε διαδρομή ενώ το degPerSec αντιπροσωπεύει τις μοίρες τις οποίες θα περιστρέφεται το drone σε μορφή degrees/seconds. Αν η τιμή είναι θετική το drone κινείται δεξιόστροφα ενώ αν είναι αρνητική κινείται αριστερόστροφα.

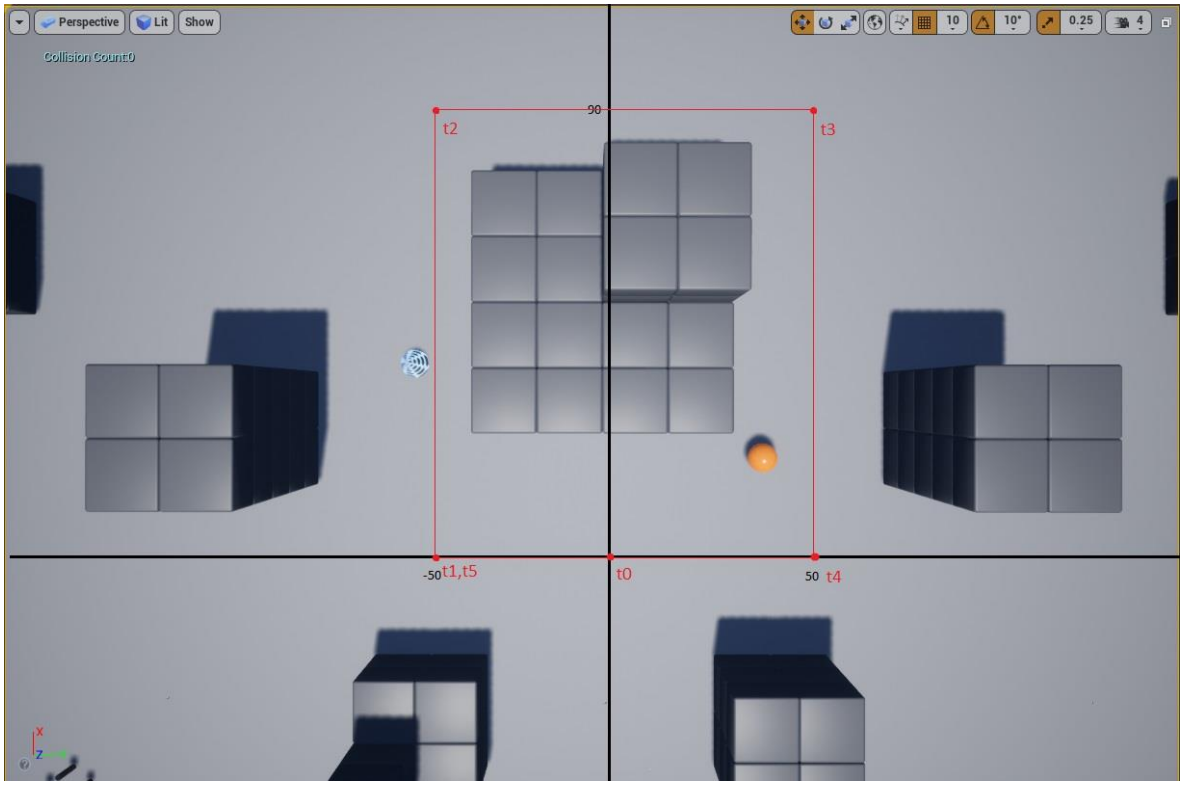


Figure 4.5 Drone Route Example

Όπως φαίνεται στο σχήμα 4.5 αν υποθέσουμε ότι η αρχική θέση του οχήματος ξεκινά στην αρχή των αξόνων – τ0, και ο χρήστης επιθυμεί να παρακολουθήσει το κτήριο που βρίσκεται στο κέντρο του σχήματος, τότε η διαδρομή που θα εισάγουμε στο αρχείο path.json θα έχει την τροχιά όπως φαίνεται στο σχήμα. Δηλαδή θα ξεκινήσει από το t0 και θα ακολουθήσει την διαδρομή t1, t2, t3, t4, t5.

Κατά συνέπεια η διαδρομή του δεύτερου οχήματος θα ακολουθήσει την αντίθετη τροχιά. Ξεκινώντας με κάποιο offset από την θέση t0 θα ακολουθήσει την διαδρομή t4, t3, t2, t1, t4.

T0: 0, 0

T1: 0, -50

T2: 90, -50

T3: 90,50

T4: 0, 50

T5 = T1

Όπως θα δούμε στην συνέχεια τα ύψη των drones καθώς και η αρχική τους θέση σε κάθε διαδρομή εξαρτούνται από το ύψος και την αρχική θέση που δίνεται στην πρώτη διαδρομή και αλλάζουν σε κάθε διαδρομή ούτως ώστε να μην υπάρξει συντριβή μεταξύ των drones.

Αρχείο επεξεργασίας των JSON Files

Το αρχείο επεξεργασίας των JSON files τα οποία έχουμε προαναφέρει, ονομάζεται editSettings.py και βρίσκεται στον ίδιο φάκελο μαζί με τα JSON αρχεία.

Ο κώδικας ο οποίος εμπεριέχεται σε αυτό το αρχείο ουσιαστικά επεξεργάζεται τα JSON αρχεία με βάση τις δύο διαδρομές που έχουν εισαχθεί στο αρχείο paths.json και με βάση το αρχικοποιημένο drone το οποίο υπάρχει στο αρχείο addRobot.json.

editSettings.py*

settingsSetup

```
def settingsSetup(numberOfDrones):
    settingsPath = Path("C:/Users/User/Documents/AirSim/settings.json")
    robotPath=Path("C:/Users/User/Documents/AirSim/addRobot.json")
    with open(settingsPath, 'r') as myfile:
        data=myfile.read()
    with open(robotPath, 'r') as myfile:
        data2=myfile.read()
    settings = json.loads(data)
    robot = json.loads(data2)

    if len(settings['Vehicles'])>1 :
        for element in settings["Vehicles"].copy():
            if element != 'robot1':
                settings["Vehicles"].pop(element, None)
    for i in range (2,numberOfDrones+1):
        settings['Vehicles']['robot'+str(i)]=robot['robot1']
    with open("settings.json", "w") as jsonFile:
        json.dump(settings, jsonFile)

    pathsPath=Path("C:/Users/User/Documents/AirSim/path.json")
    with open(pathsPath, 'r') as myfile2:
        data3=myfile2.read()
    paths=json.loads(data3)
    for element in paths.copy():
        if (int(element)!=1) and (int(element)!=2):
                paths.pop(element,None)
    with open("path.json", "w") as jsonFile:
        json.dump(paths, jsonFile)

    if numberOfDrones>2:
        for i in range(3,numberOfDrones+1,2):
            paths[i]=paths['1']
            paths[i+1]=paths['2']
    with open("path.json", "w") as jsonFile:
        json.dump(paths, jsonFile)
```

Figure 4.6 Delete Previous Settings

Όπως φαίνεται στο σχήμα 4.6 αρχικά διαβάζονται τα αρχεία settings.json και addRobot.json. Στην συνέχεια γίνεται έλεγχος αν υπάρχουν ήδη αρχικοποιημένα οχήματα στο αρχείο settings.json. Αν υπάρχουν ήδη, τότε πρέπει να διαγραφούν έτσι ώστε να γίνει εισαγωγή των drones με βάση τον αριθμό που θα δώσει ο χρήστης ως είσοδο στο πρόγραμμα (Ο αριθμός που θα δώσει ο χρήστης ως είσοδο στο πρόγραμμα φαίνεται ως argument στον ορισμό της συνάρτησης ως numberOfDrones).

Παρομοίως στην συνέχεια, γίνεται έλεγχος στο αρχείο path.json αν υπάρχουν ήδη αρχικοποιημένες διαδρομές πέραν των πρώτων δύο διαδρομών οι οποίες

εισήχθησαν στο αρχείο path.json. Εάν υπάρχουν τότε διαγράφονται με βάση το όνομα της διαδρομής.

Όταν εκτελεστεί αυτός ο κώδικας τα αρχεία πλέον είναι αρχικοποιημένα με ένα drone και δύο διαδρομές τις οποίες έχει δώσει ο χρήστης.

```
with open(pathsPath, 'r') as myfile2:
    data3=myfile2.read()
paths=json.loads(data3)
for i in range(numberOfDrones):
    paths[str(i+1)][str(0)][1]=str(int(paths[str(i+1)][str(0)][1])-3*i)
    for k in range(5):
        paths[str(i+1)][str(k)][2]=str(-7-i)

with open("path.json", "w") as jsonFile:
    json.dump(paths, jsonFile)



with open(settingsPath, 'r') as myfile:
    data=myfile.read()
settings = json.loads(data)

space=3
yaxis=0
for x in range (2,len(settings['Vehicles'])+1):
    settings['Vehicles']['robot'+str(x)]['Y']=space
    space=space+3


with open("settings.json", "w") as jsonFile:
    json.dump(settings, jsonFile)
```

Figure 4.7 Add New Settings

Στην συνέχεια όπως φαίνεται στο σχήμα 4.7 ανάλογα με τον αριθμό των drones τα οποία έχει δώσει ως είσοδο ο χρήστης, θα παραχθεί ο ανάλογος αριθμός διαδρομών ούτως ώστε για κάθε drone να υπάρχει μια διαδρομή να ακολουθήσει. Έτσι έχει αυτοματοποιηθεί η είσοδος των διαδρομών ανάλογα με τον αριθμό των drones. Η ονομασία της κάθε διαδρομής αυξάνεται ακολουθώντας το μοτίβο «1,2,3….». Το ύψος του κάθε drone όπως φαίνεται στο σχήμα ξεκινά από το -7, και σε κάθε διαδρομή αυξάνεται κατά 1. Παρ όλ’ αυτά οι διαδρομές μπορούν να τύχουν επεξεργασίας και από τον ίδιο τον χρήστη μπαίνοντας στο αρχείο και αλλάζοντας τις διαδρομές.

Στην συνέχεια γίνεται επεξεργασία της ονομασίας του κάθε drone καθώς επίσης και της αρχικής του τοποθεσίας στο αρχείο settings.json. Όπως φαίνεται στο

σχήμα το κάθε drone που εισάγεται στο αρχείο, παίρνει ονομασία αυξητικά με την ακολουθία «robot1, robot2, robot3..». Επίσης η αρχική τους τοποθεσία όπως καθορίζεται στο settings['Vehicles']['robot'+str(x)]['Y'] αυξάνεται κάθε φορά κατά 3 μέτρα στον άξονα των Y. Έτσι κατά την απογείωση των drones να μην υπάρχουν οποιεσδήποτε συγκρούσεις μεταξύ τους.

4.2 Ρυθμίσεις Περιβάλλοντος

Όπως έχουμε δει προηγουμένως, το περιβάλλον blocks στο οποίο θα τρέξει η προσομοίωση είναι ένα από περιβάλλον απλό το οποίο περιέχει μερικά αντικείμενα τα οποία, για ευκολία του συστήματος και εξοικονόμηση χώρου, είναι λίγα και διάσπαρτα στον χώρο.

Υποθέτοντας ότι ένα από αυτά τα αντικείμενα είναι ο χώρος (για παράδειγμα η πολυκατοικία) στην οποία θα εφαρμοστεί ο έλεγχος από τα drones, τότε μπορούμε να εφαρμόσουμε την υπόθεση μας σε αυτό.

Προκειμένου όμως να γίνει αυτό, στον χώρο θα χρειαστεί να υπάρχει και ένα αντικείμενο το οποίο να κινείται κοντά ούτως ώστε να γίνει η αναγνώριση από τα drones.

Όπως μπορούμε να διακρίνουμε στο σχήμα 4.5, στο περιβάλλον υπάρχει ήδη ένα αντικείμενο στο χώρο, η πορτοκαλί μπάλα, η οποία είναι αρκετά ευδιάκριτη και βρίσκεται κοντά στο κεντρικό μπλοκ του περιβάλλοντος. Επίσης το συγκεκριμένο αντικείμενο είναι εύκολο να αναγνωριστεί από τον αλγόριθμο YOLO.

Παρ' όλ' αυτά θα χρειαστεί κάποια επεξεργασία στο περιβάλλον ούτως ώστε το περιβάλλον να προσαρμοστεί στην αρχική υπόθεση που έχουμε κάνει, δηλαδή ότι υπάρχει ένα κινούμενο αντικείμενο το οποίο θα εντοπιστεί από τα drones.

Αν πάρουμε πάλι το σχήμα 4.7, η κίνηση η οποία μπορεί να κάνει η μπάλα, είναι γύρω από το κεντρικό μπλοκ του περιβάλλοντος όπως φαίνεται και στο σχήμα με κόκκινη γραμμή. Αυτό μπορεί να υλοποιηθεί με το να γίνει επεξεργασία στο blueprint του αντικειμένου στο παράθυρο Event Graph.

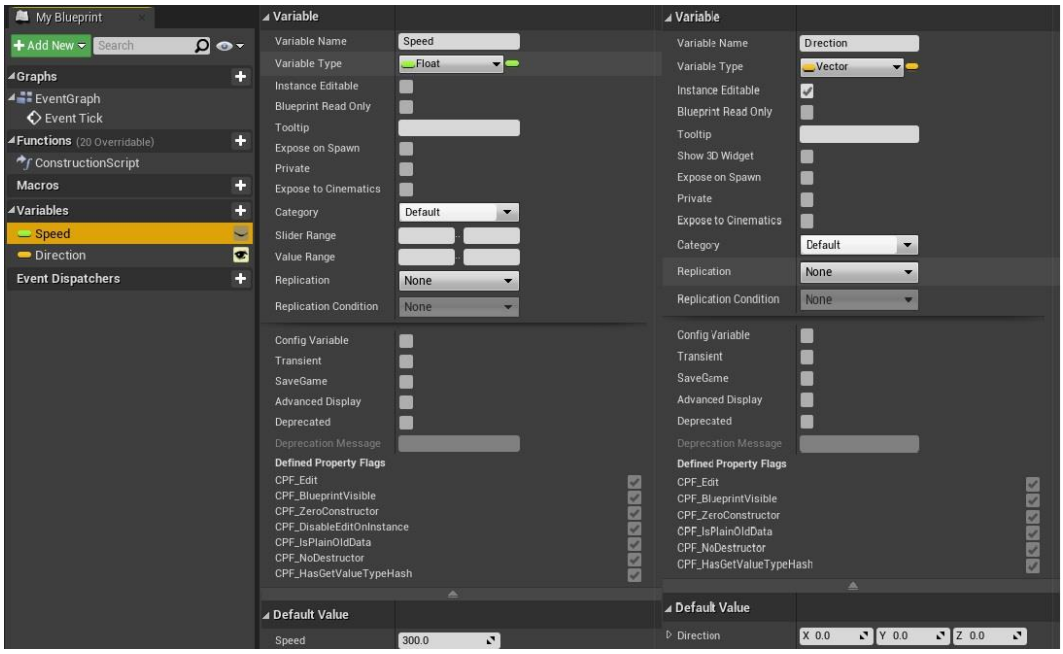


Figure 4.8 Blueprint Ball Variables

Αρχικά, όπως βλέπουμε και στο σχήμα 4.8 θα εισάγουμε δύο variables στο αντικείμενο, Speed και Direction τύπου Float και Vector και ως default values 300 και [0,0,0] αντίστοιχα.

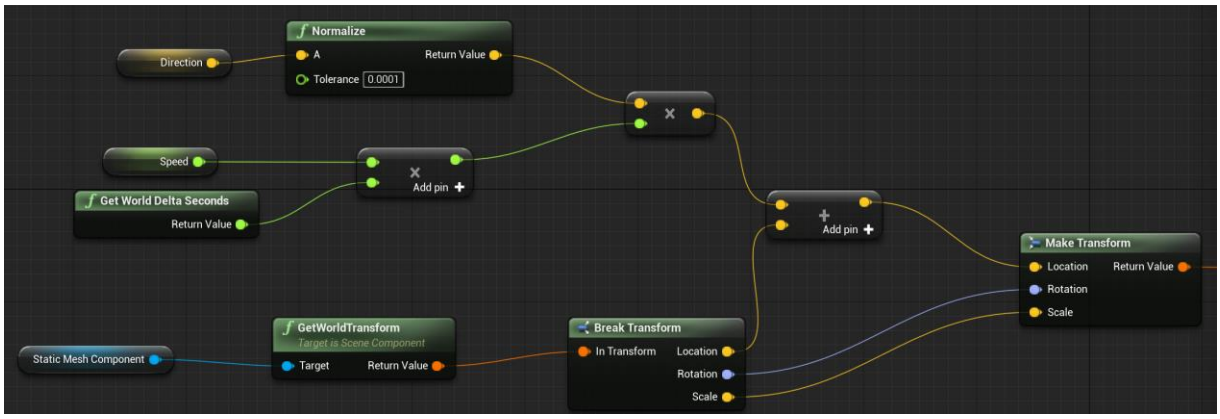


Figure 4.9 Blueprint Ball Event Graph 1

Θα υλοποιήσουμε τώρα τα απαραίτητα βήματα για να λάβουμε τις πληροφορίες που χρειαζόμαστε για να παρέχουμε οδηγίες κίνησης.

Ο πρώτος υπολογισμός που πρέπει να εκτελέσουμε είναι να πάρουμε τη διανυσματική τιμή του direction και να την κάνουμε normalize. Το normalize είναι μια κοινή διαδικασία στα μαθηματικά με vectors, που διασφαλίζει ότι το vector μετατρέπεται σε μήκος μιας μονάδας, κάτι που θα το κάνει συμβατό με τους υπόλοιπους υπολογισμούς μας. Ευτυχώς, υπάρχει ένας κόμβος στο Unreal Editor που το φροντίζει για εμάς, όπως φαίνεται και στο σχήμα 4.9 πάνω αριστερά.

Εισάγουμε την μεταβλητή direction στο κενό χώρο στο Event Graph. Κάνουμε κλικ στον ακροδέκτη εξόδου του κόμβου direction και τον ρίχνουμε στον κενό χώρο γραφήματος. Στην συνέχεια πληκτρολογούμε normalize στο πεδίο αναζήτησης και επιλέγουμε τον κόμβο normalize κάτω από την κατηγορία με ετικέτα vector. Αυτό θα συνδέσει τη μεταβλητή direction με έναν κόμβο που θα κάνει αυτόματα τον υπολογισμό κανονικοποίησης για εμάς.

Για να συσχετίσουμε την μεταβλητή speed στην μεταβλητή direction, θα χρειαστεί πρώτα να την πολλαπλασιάσουμε με χρόνο delta, ο οποίος βασίζεται στο γεγονός ότι ο χρόνος που πάρθηκε μεταξύ των frames του περιβάλλοντος μπορεί να διαφέρουν. Πολλαπλασιάζοντας την ταχύτητα με delta δευτερόλεπτα, διασφαλίζουμε ότι η ταχύτητα με την οποία θα κινείται το αντικείμενο είναι η ίδια ανεξαρτήτως των frames του παιχνιδιού.

Για να το κάνουμε αυτό εισάγουμε την μεταβλητή speed στον κενό χώρο στο Event Graph. Επίσης, με δεξί κλικ ψάχνοντας με την λέξη delta, πατούμε το Get world Delta Seconds. Για τον πολλαπλασιασμό ψάχνουμε με δεξί κλικ το Float*Float και εισάγουμε τα outputs από το Speed και το get World Delta Seconds όπως φαίνεται στο σχήμα.

Τώρα που έχουμε μια κανονικοποιημένο διανυσματικό direction και μια τιμή speed σε σχέση με το χρόνο, πρέπει να πολλαπλασιάσουμε αυτές τις δύο τιμές και στη συνέχεια να τις προσθέσουμε στην τρέχουσα θέση. Αρχικά, βρίσκουμε το στοιχείο Static Mesh Component από τον πίνακα Components και το εισάγουμε στο Event Graph. Αυτό θα δημιουργήσει έναν κόμβο από τον οποίο μπορούμε να εξαγάγουμε δεδομένα που περιέχονται στο αντικείμενο.

Στη συνέχεια, θέλουμε να βρούμε την τοποθεσία του αντικειμένου, από τα transform properties του αντικειμένου εξαγάγουμε την τοποθεσία. Κάνουμε κλικ και σέρνουμε την μπλε καρφίτσα εξόδου σε κενό χώρο και πληκτρολογούμε Get World. Επιλέγουμε την επιλογή Get World Transform για να δημιουργήσουμε τον κόμβο.

Τώρα θέλουμε να αναλύσουμε τον μετασχηματισμό στα συστατικά του μέρη, ώστε να μπορούμε να χρησιμοποιήσουμε μόνο την τοποθεσία στους υπολογισμούς μας, διατηρώντας παράλληλα την περιστροφή και την κλίμακα.

Σέρνουμε το output από το World Transform Node και αναζητούμε τον κόμβο Break Transform και το εισάγουμε στο Event Graph μας.

Τώρα πρέπει να προσθέσουμε τους απαραίτητους κόμβους για να προσθέσουμε τα speed και direction στις πληροφορίες location που μόλις εξαγάγαμε. Κάντε δεξί κλικ στον κενό χώρο πλέγματος και αναζητήστε και επιλέξτε τον κόμβο Make Transform. Ο κόμβος Make Transform έχει τρεις εισόδους, Location Rotation και Scale. Οι είσοδοι rotation και scale πρέπει να συνδέονται με τα αντίστοιχα από τις εξόδους του Break Transform που δημιουργήσαμε νωρίτερα.

Στη συνέχεια, πρέπει να πολλαπλασιάσουμε το direction vector και το speed float που υπολογίσαμε. Αναζητούμε και επιλέγουμε Vector * Float και συνδέουμε τον πράσινο input του, στην έξοδο του κόμβου πολλαπλασιασμού float που χρησιμοποιήσαμε με το speed.

Το τελικό βήμα υπολογισμού μας είναι να προσθέσουμε τα speed και direction στην τρέχουσα τοποθεσία που υπολογίσαμε. Κάνουμε κλικ στην κίτρινη καρφίτσα εξόδου του διανύσματος του νέου κόμβου πολλαπλασιασμού και την σέρνουμε στον κενό χώρο. Κάνουμε αναζήτηση χρησιμοποιώντας + και επιλέγουμε τον κόμβο Vector + Vector. Βεβαιωνόμαστε ότι μια είσοδος αυτού του κόμβου είναι συνδεδεμένη στον προηγούμενο αναφερόμενο κόμβο πολλαπλασιασμού, και συνδέουμε την άλλη είσοδο με τον κόμβο Break Transform. Τέλος, σέρνουμε την έξοδο του κόμβου προσθήκης μας στην είσοδο του κόμβου Make Transform.

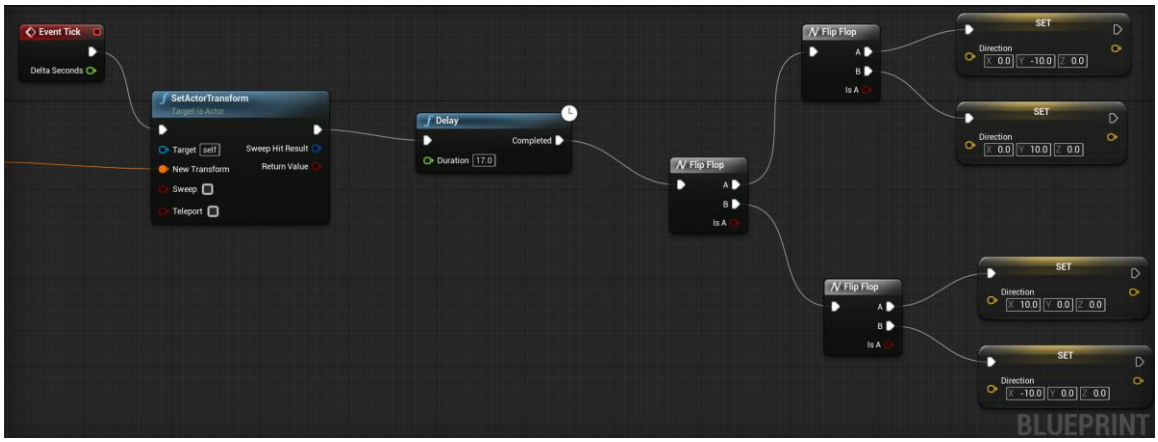


Figure 4.10 Blueprint Ball Event Graph 2

Έχοντας υπολογίσει το transform, τώρα μπορούμε να προσαρμόσουμε την τοποθεσία του αντικειμένου μας χρησιμοποιώντας το Event Tick node για να εκτελεστεί κίνηση σε κάθε frame. Για να το εισάγουμε πατούμε δεξί κλικ και ψάχνουμε το Event Tick. Για να κινηθεί το αντικείμενο, σέρνουμε την έξοδο του event tick και το ενώνουμε με το node Set Actor Transform στην άσπρη είσοδο όπως φαίνεται στο σχήμα 4.10. Στη συνέχεια ενώνουμε την έξοδο του Make Transform από το σχήμα 4.9 στην είσοδο New Transform.

Στην συνέχεια θα χρειαστεί να εισάγουμε flip flops τα οποία θα αλλάζουν την κατεύθυνση του αντικειμένου κάθε λίγα δευτερόλεπτα ούτως ώστε να εκτελέσει την κίνηση την οποία επιθυμούμε.

Όπως φαίνεται και στο σχήμα 4.10 εισάγοντας 3 flip flops και τα 4 sets τα οποία αλλάζουν την κατεύθυνση κάθε 17 δευτερόλεπτα το αντικείμενο θα εκτελεί έναν ατέρμον βρόγχο γύρω από το block στο οποίο αναφερθήκαμε πιο πριν.

4.3 Κώδικας Drone

Ο κώδικας για τα drones βρίσκεται στην τοποθεσία AirSim/PythonClient/Multirotor/hello_drone.py και χωρίζεται σε 3 στάδια. Το πρώτο στάδιο είναι η είσοδος των δεδομένων από τον χρήστη και η αρχικοποίηση των directories. Το δεύτερο στάδιο και το πιο σημαντικό είναι η εκτέλεση των διαδρομών από τα εκάστοτε drones, ενώ το τρίτο στάδιο είναι η δημιουργία του αρχείου traces.yaml.

```
numberOfPaths=int(input("Enter number of paths: "))

jsnPath = input("Enter paths.json location: ") #C:\Users\User\Documents\AirSim\path.json

with open(jsnPath, 'r') as myfile:
    data=myfile.read()

paths = json.loads(data)

client = airsim.MultirotorClient()
client.confirmConnection()

tmp_dir = os.path.join(tempfile.gettempdir(), "airsim_drone")
if os.path.exists(tmp_dir):
    shutil.rmtree(tmp_dir)
os.makedirs(tmp_dir)
images = []
for i in range(numberOfPaths):
    directory = "robot"+str(i+1)
    path = os.path.join(tmp_dir, directory)
    try:
        os.makedirs(path)
    except OSError:
        if not os.path.isdir(path):
            raise

    images.append(0)
```

Figure 4.11 Drone Code User Input

Όπως βλέπουμε και στο σχήμα 4.11 η εκτέλεση του πρώτου βήματος γίνεται με τον εξής τρόπο: Αρχικά ο χρήστης δίνει ως είσοδο από την κονσόλα το path στο οποίο βρίσκεται το αρχείο path.json, το αρχείο το οποίο τροποποιήσαμε όπως έχουμε πει στην αρχή αυτού του κεφαλαίου.

Στην συνέχεια ανοίγουμε το αρχείο και ενεργοποιούμε το Multirotor Client και δημιουργούμε στο temporary directory τον φάκελο airsim_drone αν δεν υπάρχει ήδη. Επίσης, αφού έχουμε από τον χρήστη τον αριθμό των drones τα οποία θα εκτελέσουν την διαδρομή, δημιουργούμε υπο-φακέλους ανάλογα με την είσοδο του χρήστη. Η ονομασία του κάθε φακέλου είναι η ίδια με την ονομασία των drones τα οποία θα έχουμε στην προσομοίωση.

Για παράδειγμα εάν ο αριθμός των drones ο οποίος έδωσε ο χρήστης είναι 3, τότε η δομή των φακέλων θα είναι: ../airsim_drone/robot1, ../airsim_drone/robot2 και ../airsim_drone/robot3.

Τέλος εισάγουμε στην λίστα images τον αριθμό 0 ο οποίος αντιπροσωπεύει τον αριθμό φωτογραφιών οι οποίες έχουν ληφθεί από το drone που έχει ονομασία το ίδιο index (+1).

Ο αλγόριθμος παρακολούθησης χώρου ο οποίος έχει υλοποιηθεί στην εργασία χωρίζεται σε 3 βασικά βήματα:

- Ενεργοποίηση και απογείωση των Drones
- Μονοπάτι και καταγραφή αποτελεσμάτων
- Προσγείωση των drones

Ενεργοποίηση και απογείωση των Drones

```
xPositions = []
yPositions = []
for i in range (numberOfPaths):
    client.enableApiControl(True,"robot"+str(i+1))
    xPositions.append(getX("robot"+str(i+1)))
    yPositions.append(getY("robot"+str(i+1)))

for i in range (numberOfPaths):
    client.armDisarm(True,"robot"+str(i+1))
x1 = []
for i in range (numberOfPaths):
    x1.append(client.takeoffAsync(5,"robot"+str(i+1)))
for i in range (numberOfPaths):
    x1[i].join()
```

Figure 4.12 Activating Drones

Για την ενεργοποίηση των drones και την απογείωση τους όπως φαίνεται και στο σχήμα 4.12 υπάρχουν 3 κλήσεις API και αρχικοποίηση δύο λιστών.

Η πρώτη κλήση είναι η enableApiControl η οποία δέχεται ως είσοδο True/False (στη δική μας περίπτωση θέλουμε να πάρουμε τον έλεγχο του κάθε οχήματος εξ ου και το True) και το όνομα του οχήματος στο οποίο θα πάρουμε τον έλεγχο. Αυτό συμβαίνει διότι για λόγους ασφαλείας ο προεπιλεγμένος έλεγχος API για αυτόνομο Drone δεν είναι ενεργοποιημένος και ο ανθρώπινος χειριστής έχει πλήρη έλεγχο. Αυτή η κλήση πρέπει να πραγματοποιηθεί για να ζητηθεί ο έλεγχος μέσω API για κάθε drone. Επίσης είναι πιθανό ο ανθρώπινος χειριστής του οχήματος να έχει απαγορεύσει τον έλεγχο API, κάτι που σημαίνει ότι αυτή η κλήση δεν θα είχε αποτέλεσμα. Εφόσον ξέρουμε τον αριθμό των Drones τα οποία θα απογειώσουμε (τα έχει δώσει ως είσοδο ο χρήστης στην αρχή του

προγράμματος), και ξέρουμε πως έχουν ονομαστεί τα οχήματα τότε είναι εύκολο να ξέρουμε την ονομασία του κάθε drone για να έχουμε πρόσβαση σε αυτό.

Στην συνέχεια αποθηκεύουμε στις λίστες xPositions και yPositions τις αρχικές θέσεις X και Y του κάθε drone.

```
def getX(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.x_val,3)

def getY(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.y_val,3)

def getZ(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.z_val,3)

def getTrace(robotName):
    return (getX(robotName),getY(robotName),getZ(robotName),round(client.getMultirotorState(robotName).timestamp,3))
```

Figure 4.13 Drone's Position APIs

Για να έχουμε πρόσβαση στις αρχικές θέσεις των drones, υπάρχει υλοποιημένο το API για την κατάσταση του οχήματος, όπως φαίνεται στο σχήμα 4.13. Έχοντας στην διάθεση μας την ονομασία του οχήματος, με την κλήση του getMultirotorState(Όνομα του οχήματος).kinematics_estimated.position, έχουμε πρόσβαση σε ένα vector, το οποίο έχει αποθηκευμένα τα X Y και Z (altitude) του οχήματος. Επίσης η μέθοδος getTrace επιστρέφει παίρνοντας ως είσοδο το όνομα του drone, μια λίστα η οποία έχει την θέση X,Y,Z του drone καθώς επίσης και το timestamp τα οποία θα χρησιμοποιηθούν αργότερα.

Για να οπλιστεί (arm) το κάθε drone χρησιμοποιούμε την κλήση API armDisarm, με τις ίδιες εισόδους, δηλαδή το όνομα του drone κάθε φορά, και True εφόσον θέλουμε να οπλιστεί.

Στην συνέχεια απογειώνουμε το κάθε Drone με την κλήση του API takeoffAsync, και δίνοντας ως είσοδο το ύψος στο οποίο θέλουμε να απογειωθεί κάθετα καθώς επίσης και την ονομασία του drone. Η κάθε κλήση αποθηκεύεται σε μια λίστα ούτως ώστε μετά για κάθε κλήση να καλεστεί η join() η οποία θα τα απογειώσει όλα ταυτόχρονα.

Μονοπάτι και καταγραφή αποτελεσμάτων

```
result=[]
pathsRecorded = []
for i in range(numberOfPaths):
    result.append( client.moveOnPathAsync([airsim.Vector3r(int(paths[str(i+1)][str(0)][0]),int(paths[str(i+1)][str(0)][1]),int(paths[str(i+1)][str(0)][2])),
                            airsim.Vector3r(int(paths[str(i+1)][str(1)][0]),int(paths[str(i+1)][str(1)][1]),int(paths[str(i+1)][str(1)][2])),
                            airsim.Vector3r(int(paths[str(i+1)][str(2)][0]),int(paths[str(i+1)][str(2)][1]),int(paths[str(i+1)][str(2)][2])),
                            airsim.Vector3r(int(paths[str(i+1)][str(3)][0]),int(paths[str(i+1)][str(3)][1]),int(paths[str(i+1)][str(3)][2])),
                            airsim.Vector3r(int(paths[str(i+1)][str(4)][0]),int(paths[str(i+1)][str(4)][1]),int(paths[str(i+1)][str(4)][2]))],
                    paths[str(i+1)]["vel"], paths[str(i+1)]["time-out"],
                    airsim.DrivetrainType.MaxDegreeOfFreedom, airsim.YawMode(True,paths[str(i+1)]["degPerSec"]), -1, 1,"robot"+str(i+1)))
    pathsRecorded.append([])

start = time.time()
time.perf_counter()
elapsed = 0
blacklist=[]
while elapsed < 170:

    for i in range(numberOfPaths):
        if (i+1) not in blacklist:
            pathsRecorded[i].append(getTrace('robot'+str(i+1)))
            pos=serverConnection(takeImage('robot'+str(i+1)))
            if (pos[0]!=0) or (pos[1]!=0):
                client.cancelLastTask('robot'+str(i+1))
                blacklist.append(i+1)
    oldElapsed=elapsed
    elapsed = time.time() - start
    oneIterationTime=elapsed-oldElapsed
    print(oneIterationTime)
    if len(blacklist) == numberOfPaths:
        break
```

Figure 4.14 Drone's Path Execution

Για την εκτέλεση των διαδρομών όπως έχουν δοθεί από τον χρήστη χρησιμοποιούμε το API moveOnPathAsync το οποίο παίρνει σαν είσοδο τις πληροφορίες οι οποίες έχουν εισαχθεί στο αρχείο path.json. Συγκεκριμένα όπως βλέπουμε και στο σχήμα 4.14, εκτελούμε την εντολή για το κάθε ένα από τα drones δίνοντας σε κάθε drone μια διαδρομή να ακολουθήσει. Η κάθε διαδρομή όπως έχουν εισαχθεί στο αρχείο path.json περιέχει 5 σημεία τα οποία θα ακολουθήσει διαδοχικά το drone όπως φαίνεται στο πιο πάνω σχήμα. Το API παίρνει επίσης ως είσοδο την ταχύτητα με την οποία θα εκτελεστεί η διαδρομή καθώς επίσης και τον χρόνο στον οποίο θα εκτελεστεί η διαδρομή αλλιώς η εντολή θα κάνει time-out. Κάποιες άλλες παράμετροι οι οποίες φαίνονται πιο πάνω είναι οι εξής:

Drivetrain Type: Max Degree of Freedom
Yaw Mode: True , degrees per second
Look ahead/Adaptive look ahead: -1, 1

Υπάρχουν δύο τρόποι με τους οποίους μπορούμε να πετάξουμε το όχημα: η παράμετρος Drivetrain έχει ρυθμιστεί στο airsim.DrivetrainType.**ForwardOnly** ή airsim.DrivetrainType.**MaxDegreeOfFreedom**.
Όταν καθορίζεται το **ForwardOnly**, λέμε ότι το μπροστινό μέρος του οχήματος πρέπει πάντα να δείχνει προς την κατεύθυνση που εκτελεί κίνηση το drone. Έτσι, εάν θέλουμε το drone να στρίψει αριστερά, τότε θα περιστραφεί πρώτα ώστε τα μπροστινά σημεία να βλέπουν προς τα αριστερά. Το **MaxDegreeOfFreedom**

σημαίνει ότι δεν μας ενδιαφέρει που δείχνει το μπροστινό μέρος. Έτσι, όταν στρίψουμε αριστερά το drone, απλά κινείται αριστερά σαν καβούρι. Στην δική μας περίπτωση χρειαζόμαστε το **MaxDegreeOfFreedom** αφού ανεξαρτήτως διαδρομής, εμάς μας ενδιαφέρει η παρακολούθηση του χώρου. Αν ο χώρος βρίσκεται στα δεξιά μας για παράδειγμα, θέλουμε το drone να βλέπει δεξιά προς τον χώρο και όχι την διαδρομή που ακολουθεί.

Το yaw_mode είναι μια δομή YawMode με δύο πεδία, yaw_or_rate και is_rate. Εάν το πεδίο is_rate είναι True τότε το πεδίο yaw_or_rate ερμηνεύεται μοίρες / δευτερόλεπτο που σημαίνει ότι θέλουμε το όχημα να περιστρέφεται συνεχώς γύρω από τον άξονά του σε αυτήν τη γωνιακή ταχύτητα ενώ κινείται. Εάν το is_rate είναι False τότε το yaw_or_rate ερμηνεύεται ως γωνία σε μοίρες, πράγμα που σημαίνει ότι θέλουμε το όχημα να περιστρέφεται σε συγκεκριμένη γωνία (δηλ. Yaw) και να διατηρεί αυτή τη γωνία ενώ κινείται. Στην δική μας περίπτωση η οποία είναι ένας τετράγωνος χώρος και θα εκτελέσουμε την ανάλογη διαδρομή, μας ενδιαφέρει να περιστρέφεται συνεχώς γύρω από τον άξονα του με κάποια γωνιακή ταχύτητα. Έτσι στον κώδικα όπως βλέπουμε στο σχήμα 4.14 το YawMode έχει καθοριστεί ως True, και η γωνιακή ταχύτητα ερμηνεύεται από το αρχείο path.json.

Όταν ζητάμε από το όχημα να ακολουθήσει ένα μονοπάτι, το AirSim χρησιμοποιεί "carrot following algorithm". Αυτός ο αλγόριθμος λειτουργεί κοιτάζοντας μπροστά στη διαδρομή και προσαρμόζοντας το διάνυσμα ταχύτητάς του. Οι παράμετροι για αυτόν τον αλγόριθμο καθορίζονται από το look ahead και το adapive look ahead. Στη δική μας περίπτωση θέλουμε τις default τιμές οι οποίες είναι -1 και 1 αντίστοιχα.

Η τελευταία παράμετρος του API είναι η ονομασία του drone η οποία θα ακολουθήσει την εκάστοτε διαδρομή.

Στο τέλος αυτής της εντολής για το κάθε drone, εισάγουμε μια κενή λίστα στην λίστα pathsRecorded. Το index της λίστας pathsRecorded αντιπροσωπεύει το κάθε drone. Για παράδειγμα αν θα εκτελεστεί η προσομοίωση με 2 drones τότε :

pathsRecorded [0]: Λίστα από τα traces του drone με ονομασία “robot1”

pathsRecorded [1]: Λίστα από τα traces του drone με ονομασία “robot2”

Όταν όλα τα drones ξεκινήσουν το καθένα την δική του διαδρομή, ξεκινά ένα for loop το οποίο έχει ως τερματική περίπτωση τον χρόνο ο οποίος χρειάζεται για να

εκτελέσει το κάθε drone την διαδρομή, ή αλλιώς το time-out. Στην δική μας περίπτωση, το κάθε drone χρειάζεται περίπου 170 δευτερόλεπτα για να εκτελέσει την διαδρομή.

Εντός του loop, κάθε φορά εκτελούμε τα εξής βήματα:

Για κάθε drone παίρνουμε το trace του χρησιμοποιώντας την μέθοδο getTrace από το σχήμα 4.13 και το αποθηκεύουμε στην λίστα pathsRecorded.

Στην συνέχεια χρησιμοποιούμε τις μεθόδους serverConnection και takeImage χρησιμοποιώντας το όνομα του εκάστοτε drone.

```
def takeImage(robotName):
    global images
    png_image=client.simGetImage("high_res", airsim.ImageType.Scene,robotName)
    n=robotName.replace('robot','')
    dir=robotName+"\\"+ str(images[int(n)-1])
    filename = os.path.join(tmp_dir, dir )
    images[int(n)-1]=images[int(n)-1]+1
    airsim.write_file(os.path.normpath(filename + '.png'), png_image)
    pngDirectory=os.path.normpath(filename + '.png')
    return pngDirectory
```

Figure 4.15 Take Image Method

Αρχικά κάνουμε χρήση του API simGetImage χρησιμοποιώντας το όνομα της cameras “high_res” όπως εμείς την έχουμε ορίσει στο αρχείο settings.json, με τύπο φωτογραφίας το Scene και την ονομασία του drone με το οποίο θα γίνει λήψη της φωτογραφίας.

Στην συνέχεια παίρνουμε την ονομασία του drone, για να έχουμε πρόσβαση στον ανάλογο φάκελο στα directories που δημιουργήσαμε προηγουμένως, και τον αριθμό του drone, για να έχουμε πρόσβαση στο index της λίστας images η οποία κρατά τον αριθμό των φωτογραφιών οι οποίες υπάρχουν ήδη στον συγκεκριμένο φάκελο, δηλαδή οι φωτογραφίες που έχουν φωτογραφηθεί ήδη από το drone, και μετέπειτα ανανεώνουμε τον αριθμό των φωτογραφιών στην λίστα με το ανάλογο index. Τέλος αφού η φωτογραφία είναι έτοιμη με την σωστή ονομασία, την αποθηκεύουμε στον φάκελο και τελειώνοντας η συνάρτηση επιστρέφει πίσω το path στο οποίο έχει αποθηκευτεί η φωτογραφία.

```
def serverConnection(path):
    client2=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client2.connect(("127.0.0.1",8080))
    img=cv2.imread(path)
    f = io.BytesIO()
    np.savez_compressed(f,frame=img)
    f.seek(0)
    out = f.read()
    client2.sendall(out)
    from_server=(client2.recv(1024)).decode('utf-8')
    pos=json.loads(from_server)
    return pos
```

Figure 4.16 Server Connection Method

Έχοντας ως είσοδο την τοποθεσία της φωτογραφίας που θέλουμε να αναλύσουμε, αρχικά εγκαθιδρύουμε σύνδεση με τον server ο οποίος ακούει στο port όπως φαίνεται πιο πάνω και φορτώνουμε την φωτογραφία. Στην συνέχεια στέλνουμε την φωτογραφία μέσω του δικτύου στον server και μετά λαμβάνουμε πίσω απάντηση από τον server.

Τέλος επιστρέφουμε από την μέθοδο την απάντηση του server κωδικοποιώντας την πρώτα σε json format. Όπως θα δούμε πιο μετά η απάντηση του server καθορίζει κατά πόσο υπάρχει το αντικείμενο στην φωτογραφία που έχει αναλύσει ή όχι.

Επιστρέφοντας στο σχήμα 4.14, αφού έχει καλεστεί η μέθοδος serverConnection, συνεπώς και η μέθοδος takeImage, έχουμε απάντηση από τον server σε μορφή json, κατά πόσο υπάρχει το αντικείμενο στην φωτογραφία ή όχι, κάτι το οποίο ελέγχει το if statement αμέσως μετά. Εάν υπάρχει το αντικείμενο τότε σταματάμε το drone από το να συνεχίσει την διαδρομή και επιβλέπει το συγκεκριμένο σημείο στο οποίο έχει εντοπίσει το αντικείμενο. Ταυτόχρονα εισάγουμε στην λίστα blacklist τον αριθμό του drone ουσιαστικά το οποίο έχει κάνει τον εντοπισμό, δηλαδή εάν έχει εντοπιστεί από το robot2, τότε αποθηκεύουμε τον αριθμό 2. Αυτή η λίστα ελέγχεται αν περιέχει τον αριθμό του εκάστοτε drone κάθε φορά που πάει να γίνει μια λήψη φωτογραφίας ούτως ώστε τα drones τα οποία έχουν ήδη εντοπίσει το αντικείμενο να μην συνεχίσουν να βγάζουν φωτογραφίες.

Τέλος γίνεται ο έλεγχος εάν όλα τα drones έχουν εντοπίσει το αντικείμενο, και συνεπώς το μέγεθος της λίστας blacklist είναι το ίδιο με τον αριθμό των drones, ούτως ώστε να σταματήσει η εκτέλεση του αλγορίθμου.

Προσγείωση των drones

```
x2 = []
for i in range (numberOfPaths):
    x2.append(client.moveToPositionAsync(getX("robot"+str(i+1)),getY("robot"+str(i+1)),-25-i,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
for i in range (numberOfPaths):
    x2[i].join()
x3 = []
for i in range (numberOfPaths):
    x3.append(client.moveToPositionAsync(xPositions[i],yPositions[i],-25-i,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
for i in range (numberOfPaths):
    x3[i].join()
x4= []
for i in range (numberOfPaths):
    x4.append(client.moveToPositionAsync(xPositions[i],yPositions[i],0,3,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
for i in range (numberOfPaths):
    x4[i].join()
for i in range (numberOfPaths):
    client.enableApiControl(False,"robot"+str(i+1))


return pathsRecorded
```

Figure 4.17 Landing the drones

Αφού έχει εκτελεστεί η διαδρομή από το κάθε drone και έχει γίνει η ανάλογη αναγνώριση, θέλουμε να επιστρέψουμε τα drones πίσω στην αρχική τους θέση. Λόγω του ότι το κάθε drone βρίσκεται σε διαφορετική θέση, και σε κάθε θέση μπορεί να υπάρχει οποιοδήποτε αντικείμενο το οποίο μπορεί να εμποδίζει την επιστροφή του, μια λύση είναι το κάθε drone πρώτα να αυξήσει το ύψος του στην θέση στην οποία βρίσκεται ήδη ούτως ώστε να μην υπάρχουν εμπόδια σε αυτό το ύψος, και μετά να επιστρέψει στις X, Y αρχικές του θέσεις ούτως ώστε να προσγειωθεί.

Ακολουθώντας την πιο πάνω λογική καλούμε το API moveToPositionAsync για το κάθε drone 3 φορές. Οι παράμετροι είναι παρόμοιες όπως και στο moveOnPathAsync που έχουμε δει προηγουμένως: X Y Z συντεταγμένες του drone, ταχύτητα του drone, χρόνος time-out, DriverainType, Yaw mode, look ahead, adaptive look ahead, όνομα του drone.

Την πρώτη φορά, κρατάμε σταθερά τα X και Y των drones, καλώντας τις μεθόδους getX και getY ενώ αυξάνουμε το ύψος τους στα 25 μέτρα.

Στην συνέχεια αφού όλα τα drones βρίσκονται στο ίδιο ύψος, καλούμε ξανά το ίδιο API, χρησιμοποιώντας αυτή την φορά τις αρχικές τους συντεταγμένες X και Y τις οποίες έχουμε αποθηκευμένες στις λίστες xPositions και yPositions.

Τέλος αφού όλα τα drones βρίσκονται στην αρχική τους θέση αλλά με το ύψος 25, καλούμε το ίδιο API αλλάζοντας μόνο το ύψος αυτή τη φορά.

Όταν όλα τα drones έχουν προσγειωθεί απενεργοποιούμε τον έλεγχο των drones μέσω API ξανακαλώντας το enableApiControl ορίζοντας αυτή την φορά την Boolean παράμετρο σε false.

Βγαίνοντας από την συνάρτηση επιστρέφουμε την λίστα από τα traces του κάθε drone τα οποία έχουν καταγραφεί κατά την προσομοίωση.

Μετά το τέλος της προσομοίωσης φτάνουμε στο τελικό βήμα του κώδικα του drone το οποίο είναι η δημιουργία του αρχείου traces.yaml. Το μόνο που χρειάζεται για την δημιουργία αυτού του αρχείου είναι οι διαδρομές οι οποίες έχουν καταγραφεί και επιστραφεί από την μέθοδο pathsExecution την οποία έχουμε αναλύσει προηγουμένως.

```
def createYamlFile(pathsRecorded):
    #Starting point of drone 0 lat lon on maps. 5th decimal is meters
    latMap=35.14500
    lonMap=33.40960

    maxLength = max(len(x) for x in pathsRecorded)
    ymlObj={
        "scenarios":[{
            "name":"mobility_scenario",
            "actions":[]
            }]
        }

    counter=0
    for t in range(maxLength):
        for dronesNum in range(len(pathsRecorded)):
            if len(pathsRecorded[dronesNum])>t:
                newlat=round(pathsRecorded[dronesNum][t][0]/100000,7)+latMap
                newlon=round(pathsRecorded[dronesNum][t][1]/100000,7)+lonMap
                reverseAlt=pathsRecorded[dronesNum][t][2]*(-1)
                dictionaryLatLonAlt={"network":"drone-network","lat": newlat, "lon": newlon,"alt":reverseAlt}
                moveDict={"type":"move","parameters": dictionaryLatLonAlt.copy()}
                dictionaryOutter={"time":t,"position":counter,"instance_type":"robot"+str(dronesNum+1),"instances":1,"action":moveDict}
                ymlObj["scenarios"][0]["actions"].append(dictionaryOutter.copy())
                counter+=1



    with open(r"C:\Users\User\Desktop\traces.yaml", 'w') as f:
        data = yaml.dump(ymlObj,f,sort_keys=False)
```

Figure 4.18 Creating Traces File

Όπως φαίνεται και στο σχήμα 4.18, αρχικά έχουμε ορίσει κάποιες συντεταγμένες τις οποίες μπορούμε να πάρουμε από το Google Maps[11]. Στην δική μας περίπτωση έχουμε ορίσει ως σημείο του χάρτη την κεντρική πλατεία του Πανεπιστημίου Κύπρου. Έτσι μετατρέποντας τις συντεταγμένες από την προσομοίωση μας μπορούμε να τις αντιστοιχίσουμε σε πραγματικές συντεταγμένες στον παγκόσμιο χάρτη.

Για την δημιουργία του αντικειμένου yaml, το αρχικοποιούμε με την δομή που φαίνεται πιο κάτω. Κάτω από τις ενέργειες (actions), θα ξεκινήσουμε να εισάγουμε μια λίστα από ενέργειες οι οποίες βασίζονται στον χρόνο. Συγκεκριμένα θα έχουμε τις θέσεις και τις πληροφορίες για κάθε drone τον εκάστοτε χρόνο.

Για παράδειγμα, σε μια προσομοίωση που συμπεριλαμβάνει 3 drones, κατά τον χρόνο 0, θα δημιουργηθούν οι 3 θέσεις στη λίστα actions όπως φαίνεται στο σχήμα 4.19.

```yaml
- time: 0
  position: 0
  instance_type: robot1
  instances: 1
  action:
    type: move
    parameters:
      network: drone-network
      lat: 35.145
      lon: 33.4096
      alt: 0.86
- time: 0
  position: 1
  instance_type: robot2
  instances: 1
  action:
    type: move
    parameters:
      network: drone-network
      lat: 35.145
      lon: 33.4096383
      alt: 5.232
- time: 0
  position: 2
  instance_type: robot3
  instances: 1
  action:
    type: move
    parameters:
      network: drone-network
      lat: 35.1449786
      lon: 33.4095018
      alt: 2.482
```

Figure 4.19 Traces File Example

Στην συνέχεια δημιουργούμε το αρχείο traces.yaml, αν δεν υπάρχει ήδη, στην τοποθεσία την οποία επιθυμούμε. Στην δική μας περίπτωση στο desktop. Το trace file είναι οι τροχιές από την μοντελοποίηση του Fogify για mobility emulation των Drones. Με αυτό το αρχείο και το containerization καταφέραμε να ενσωματώσουμε την προσομοίωση με το Fogify emulation.

4.4 Κώδικας Server

Ο κώδικας server αποτελείται από 2 στοιχεία. Τον ίδιο τον server ο οποίος ουσιαστικά είναι μια διαδικασία που μετά από το bind με κάποια διεύθυνση αποτελείται από ένα socket το οποίο περιμένει από ένα client να ενωθεί μαζί του.

```python
print("starting")
serv=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serv.bind(("0.0.0.0",8080))
serv.listen()
print("listening")
while True:
    conn, addr=serv.accept()
    ultimate_buffer=bytearray()
    cond=False
    while True and (not cond):
        receiving_buffer = conn.recv(1024)
        if (len(receiving_buffer)<1024):
            cond=True
        ultimate_buffer+= receiving_buffer
    final_image=np.load(io.BytesIO(ultimate_buffer))['frame']
    pointA=0
    pointB=0
    try:
        pointA,pointB=detection_image_file(final_image, yolo_weights, yolo_cfg, coco_names, confidence_threshold, nms_threshold)
        response="["+str(pointA)+","+str(pointB)+"]"
        conn.send(response.encode('utf-8'))
    except:
        print("Could not load image")
        conn.close()
        break
    conn.close()
    print("client disconnected")
```

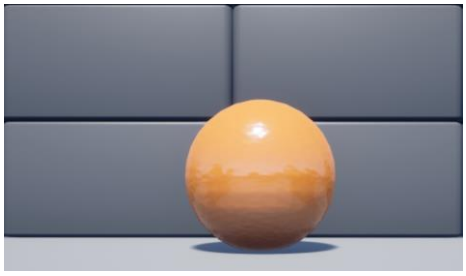
Figure 4.20 Server Code

Όπως φαίνεται και στο σχήμα 4.20 μόλις ο client, ο οποίος είναι ο κώδικας του drone που αναλύσαμε πιο πριν, ενωθεί, στέλνει στον server την φωτογραφία η οποία χρειάζεται να αναλυθεί. Μόλις γίνει η λήψη της φωτογραφίας γίνεται η ανάλυση χρησιμοποιώντας τον αλγόριθμο YOLO. Μόλις γίνει η ανάλυση ο αλγόριθμος θα επιστρέψει πίσω 2 σημεία τα οποία είναι ουσιαστικά το κέντρο του box το οποίο σχεδιάστηκε από τον αλγόριθμο και το οποίο περιέχει μέσα το αντικείμενο. Αν δεν έχει εντοπιστεί το αντικείμενο τότε τα pointA και pointB παραμένουν να είναι 0,0 που σημαίνει ότι το αντικείμενο δεν έχει εντοπιστεί. Όταν γίνει αυτή η διαδικασία ο server επιστρέφει στον client τα pointA και pointB.

```
def draw_boxes(image_path, boxes, confidences, class_ids, classes, img, colors, confidence_threshold, NMS_threshold):

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, confidence_threshold, NMS_threshold)

    FONT = cv2.FONT_HERSHEY_SIMPLEX
    pointA=0
    pointB=0
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            color = colors[i]
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
            text = "{}: {:.4f}".format(classes[class_ids[i]], confidences[i])
            cv2.putText(img, text, (x, y - 5), FONT, 0.5, color, 2)
            if classes[class_ids[i]]=="orange":
                pointA=int((x+(x+w))/2)
                pointB=int((y+(y+h))/2)

    return pointA,pointB
```

Figure 4.21 Draw Boxes Addition

Ο αλγόριθμος ο οποίος χρησιμοποιείται όπως έχουμε προαναφέρει [22] εντοπίζει το αντικείμενο προσθέτωντας στον κώδικα τον οποίο υλοποιεί τα boxes στα αντικείμενα που εντοπίζει τις εξής γραμμές όπως φαίνονται στο σχήμα 4.21. Συγκεκριμένα κατά την εισαγωγή των boxes στην φωτογραφία, εάν εντοπιστεί αντικείμενο το οποίο έχει την ονομασία “orange” τότε υπολογίζουμε το κεντρικό pixel του box και το επιστρέφουμε σε μορφή συντεταγμένης όπως φαίνεται στο σχήμα 4.22.



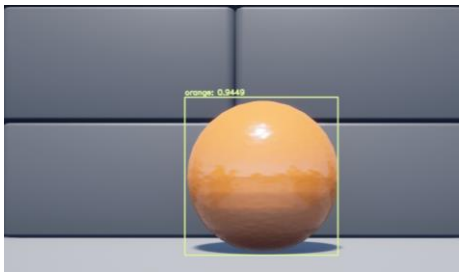


Figure 4.22 Detection Example

Ο αλγόριθμος χρησιμοποιεί 3 files:

Yolov3.weights: Τα weights του αλγορίθμου

Coco.names: Οι ονομασίες των αντικειμένων στα οποία είναι εκπαιδευμένος ο αλγόριθμος να εντοπίζει.

yolov3.cfg: Configuration File του αλγορίθμου.

4.5 Κώδικας Workload Generator

Σε αυτό το σημείο και μετά την ολοκλήρωση της εκάστοτε προσομοίωσης, έχουμε τις φωτογραφίες από τα drones καθώς επίσης και το αρχείο traces.yaml το οποίο δημιουργήθηκε στο τέλος. Αυτό μας δίνει την δυνατότητα να προσομοιώσουμε το ίδιο αποτέλεσμα χωρίς να χρειαστεί το πρόγραμμα unreal engine, αλλά μόνο τα αρχεία traces.yaml, οι φωτογραφίες, και ο server, όπως φαίνεται και στο σχήμα 4.23. Ουσιαστικά αυτό το αρχείο έχοντας ως είσοδο το αρχείο traces.yaml και τις φωτογραφίες θα παράξει τα ίδια δεδομένα που είχαν γίνει capture κατά την εκτέλεση της προσομοίωσης. Αυτό το κομμάτι είναι υλοποιημένο για να υπάρχει η δυνατότητα offline εκτέλεσης του πειράματος και γίνεται πριν το containerization και την εφαρμογή στο Fogify.

```python
def connect2(image):
    client=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(("127.0.0.1",8080))
    f = io.BytesIO()
    np.savez_compressed(f,frame=image)
    f.seek(0)
    out = f.read()
    client.sendall(out)
    from_server=(client.recv(1024)).decode('utf-8')
    client.close()
    return from_server


numberOfPaths=int(input("Enter number of drones: "))
parentPath = input("Enter files location: ") #C:\Users\User\Desktop\3drones1server
tracePath=parentPath+"/traces.yaml"
with open(tracePath) as f:
    traceData = yaml.safe_load(f)


counter=0
imagesListNumber=[]
for i in range(numberOfPaths):
    eachPath=parentPath+"/robot"+str(i+1)
    path, dirs, files = next(os.walk(eachPath))
    file_count = len(files)
    imagesListNumber.append(file_count)

print(imagesListNumber)

#Correlate counter with positions in yaml file because each position is corelated only once, and that is how the data are appended to the file
for i in range (max(imagesListNumber)):
    start = time.time()
    time.perf_counter()
    elapsed = 0
    for numberOfDrones in range(numberOfPaths):
        if imagesListNumber[numberOfDrones]>i:
            pathToImage=parentPath+"/robot"+str(numberOfDrones+1)+"/"+str(i)+".png"
            img=cv2.imread(pathToImage)
            response=connect2(img)
            alt=traceData["scenarios"][0]["actions"][counter]["action"]["parameters"]["alt"]
            lat=traceData["scenarios"][0]["actions"][counter]["action"]["parameters"]["lat"]
            lon=traceData["scenarios"][0]["actions"][counter]["action"]["parameters"]["lon"]
            print("robot"+str(numberOfDrones+1)+": lat: "+str(lat)+" lon: "+str(lon)+" alt: "+str(alt)+" time "+str(i)+" : "+response)
            counter+=1
    elapsed = time.time() - start
    print(round(elapsed,2))
    print("\n")
```

Figure 4.23 Workload Generator

Αρχικά η μέθοδος connect2 έχοντας ως παράμετρο την ίδια την φωτογραφία η οποία θα τύχει ανάλυσης, εκτελεί ακριβώς την ίδια εργασία που εκτελούσε και η μέθοδος serverConnection στον κώδικα του drone, όπως φαίνεται στο σχήμα 4.16.

Τοποθετώντας τις φωτογραφίες κάτω από ένα φάκελο, καθώς επίσης και το αρχείο traces.yaml στον ίδιο φάκελο είμαστε σε θέση να εκτελέσουμε την ίδια προσομοίωση.

Ζητώντας από τον χρήστη τον αριθμό των drones και την τοποθεσία του φακέλου του οποίου δημιουργήσαμε, το script αρχικά μετρά τις φωτογραφίες οι οποίες λήφθηκαν από το κάθε drone. Έχοντας τον αριθμό και των φωτογραφιών, μπορούμε να συσχετίσουμε την κάθε φωτογραφία με την τοποθεσία και timestamp του εκάστοτε drone την συγκεκριμένη χρονική στιγμή στην οποία έγινε η λήψη της.

Παίρνοντας τον μεγαλύτερο αριθμό φωτογραφιών από τους φακέλους, μπορούμε να δημιουργήσουμε ένα for loop το οποίο για κάθε χρονική στιγμή θα ελέγχει αν υπάρχει η συγκεκριμένη φωτογραφία για το εκάστοτε drone, και αν ναι να

τυπώνει την τοποθεσία του, και αφού γίνει η σύνδεση και ο εντοπισμός από τον server, να εκτυπώνει κατά πόσο έχει εντοπιστεί το αντικείμενο ή όχι.

Για παράδειγμα έχοντας 3 drones τα οποία έχουν και τα 3 εντοπίσει το αντικείμενο την χρονική στιγμή t0, το Workload Generator θα εμφανίσει τα παρακάτω όπως φαίνεται στο σχήμα 4.24.

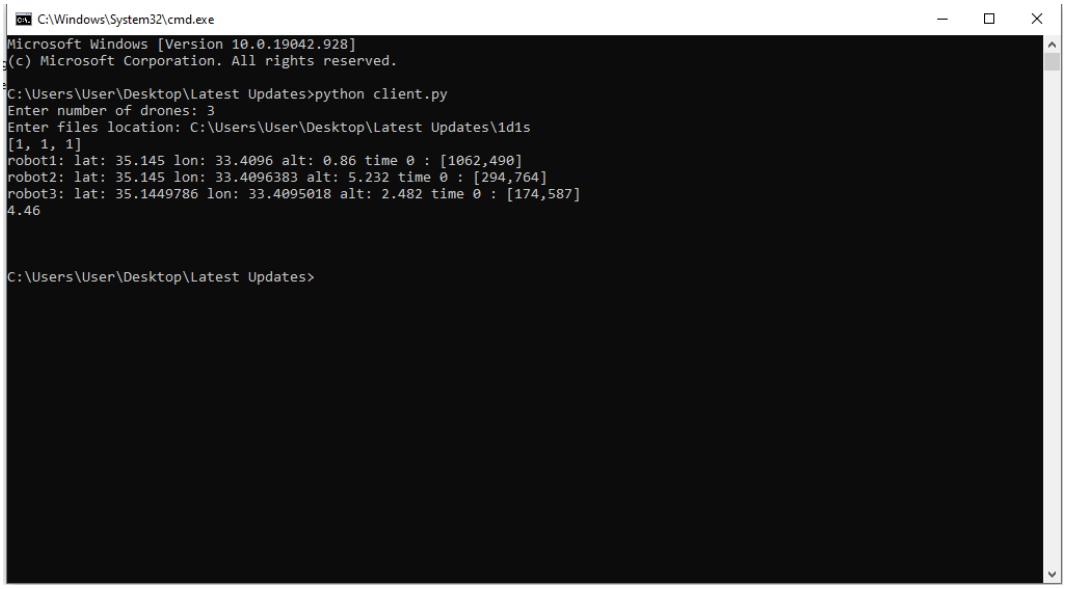


Figure 4.24 Workload Generator Execution

4.6 Κώδικας των Dockerfiles

Σε αυτό το σημείο είμαστε σε θέση να δημιουργήσουμε τα docker images. Σχετικά με αυτό το κομμάτι έχουμε δύο dockerfiles τα οποία δημιουργούν τα αντίστοιχα images για τον server καθώς επίσης και για τον workload generator.

```
# Pull base image of latest Python 3.7.x
FROM python:3.6

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN git clone https://github.com/opsengine/cpulimit/
WORKDIR cpulimit
RUN make && cp src/cpulimit /usr/bin

RUN apt-get update ##[edited]
RUN apt-get install ffmpeg libsm6 libxext6  -y

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt


EXPOSE 8080
# cron tab
ADD ./code/ /code/
ADD ./models/coco.names /code/
ADD ./models/yolov3.cfg /code/
ADD ./models/yolov3.weights /code/
#ADD crontab /etc/cron.d/crontab_file
# RUN chmod 0644 /code/cronjobs

#try to keep logs
# RUN echo "cron.*                          /var/log/cron.log" >> /etc/rsyslog.conf
# RUN service rsyslog start
#ADD code/entrypoint.sh /code/
#COPY code/entrypoint.sh /code/entrypoint.sh
RUN chmod +x /code/entrypoint.sh
CMD ["/bin/bash", "/code/entrypoint.sh"]
```

```
# Pull base image of latest Python 3.7.x
FROM python:3.6

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN git clone https://github.com/opsengine/cpulimit/
WORKDIR cpulimit
RUN make && cp src/cpulimit /usr/bin

RUN apt-get update ##[edited]
RUN apt-get install ffmpeg libsm6 libxext6  -y

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
EXPOSE 8080
ADD ./code/ /code/
ADD ./1drone1server /code/1drone1server
ADD ./3drones1server /code/3drone1server
RUN chmod +x /code/entrypoint.sh
CMD ["/bin/bash", "/code/entrypoint.sh"]
```

Figure 4.25 Dockerfiles

Όπως φαίνεται και στο σχήμα 4.25 αριστερά διακρίνεται ο κώδικας του dockerfile για τον server μαζί με τα αρχεία που χρειάζεται ο server για να τρέξει ενώ στα δεξιά είναι ο κώδικας του dockerfile για το workload generator μαζί με τα αρχεία που εισάγονται ως παράδειγμα για το emulation.

Κεφάλαιο 5

Πειράματα

5.1 Επεξεργαστική Δύναμη Σταθμού Βάσης

Στο πρώτο πείραμα θέλαμε να δούμε αν και πως η επεξεργαστική δύναμη του σταθμού βάσης επηρεάζει το χρόνο εκτέλεσης του μοντέλου μηχανικής μάθησης. Για να το πετύχουμε αυτό, εκτελέσει δύο φορές το ίδιο πείραμα αλλά με διαφορετικές δυνατότητες στο σταθμό βάσης. Συγκεκριμένα και στις δύο φορές είχαμε ένα drone που έκανε την διαδρομή που εξαγάγαμε από το airsim. Για μοντέλο στο ασύρματο δίκτυο είχαμε το **γραμμικό** και για **επεξεργαστική ισχύ** του σταθμού βάσης αρχικά είχαμε **έναν εξυπηρετητή με 2 πυρήνες στα 1.2Ghz και 2 GB μνήμη** ενώ στη δεύτερη εκτέλεση είχαμε **4 πυρήνες στα 1.2Ghz και 4 GB μνήμη**.

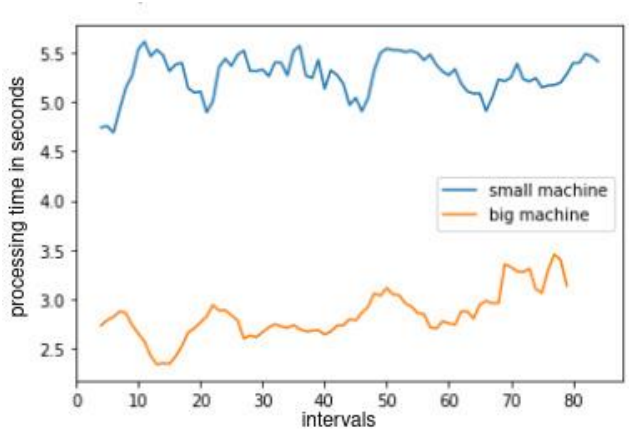


Figure 5.1 Small Vs Big Machine

Όπως μπορούμε να δούμε στο διάγραμμα 5.1 ο χρόνος εκτέλεσης του αλγορίθμου μηχανικής μάθησης είναι ανάλογος με την υπολογιστική ισχύ του σταθμού βάσης.

5.2 Μοντέλο Δικτύου

Σε αυτό το πείραμα δοκιμάσαμε διαφορετικά μαθηματικά μοντέλα για ασύρματου δικτύου για να δούμε πως επηρεάζει το καθένα την επίδοση της εφαρμογής. Έχοντας λοιπόν καταλήξει από το προηγούμενο πείραμα ότι είναι καλύτερα να έχουμε ένα μεγάλο εξυπηρετητή στο σταθμό βάσης, αφού η επεξεργαστική ισχύς επηρεάζει την απόδοση του συστήματος, κρατήσαμε αυτό σε όλα τα παρακάτω πειράματα. Σε αυτό το παράδειγμα χρησιμοποιήσαμε ένα drone και την τροχιά του.

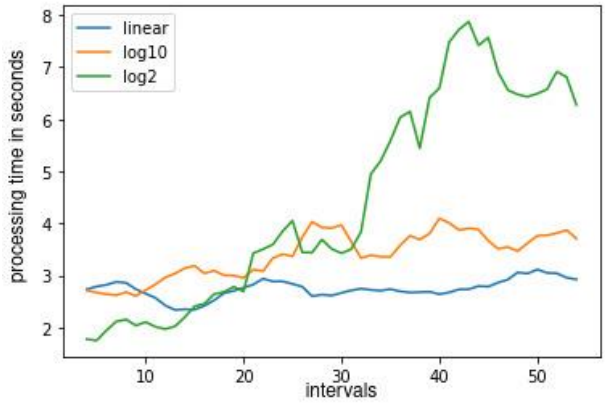


Figure 5.2 Network Models

Στο διάγραμμα 5.2 βλέπουμε πως επηρέασαν τα διαφορετικά μοντέλα το χρόνο εκτέλεσης του αλγορίθμου. Συγκεκριμένα το γραμμικό μοντέλο είναι το καλύτερο και επηρεάζει ελάχιστα το χρόνο εκτέλεσης, ενώ το λογαριθμικό με βάση το δύο φαίνεται να είναι λίγο καλύτερο για κοντινές αποστάσεις αλλά αρκετά χειρότερο για μακρινές. Τέλος το λογαριθμικό μοντέλο με βάση το δέκα έχει πολύ καλή επίδοση σε πολύ κοντινές αποστάσεις αλλά είναι πολύ χειρότερο σε μακρινές. Μπορεί να φαίνεται ότι το γραμμικό μοντέλο είναι καλύτερο αλλά στην πραγματικότητα τα ασύρματα δίκτυα ακολουθούν λογαριθμική κλίμακα οπότε για τα υπόλοιπα πειράματα χρησιμοποιούμε το λογαριθμικό μοντέλο με βάση το δύο.

5.3 Αλλαγή Παραμέτρων

Εκτός από τις αλλαγές στις παραμέτρους του περιβάλλοντος, επιλέξαμε να αλλάξουμε και παραμέτρους στην ίδια την εκτέλεση του προγράμματος. Συγκεκριμένα αλλάξαμε την **εκτέλεση της αποστολής εικόνων** στον εξυπηρετητή.

Στην αρχική υλοποίηση ο εξυπηρετητής έπρεπε να εκτελέσει τον αλγόριθμο μηχανικής μάθησης για τον εντοπισμό του αντικειμένου και να επιστρέψει κάποιο αποτέλεσμα. Αλλάξαμε αυτό το κομμάτι για να γίνεται η εκτέλεση ασύγχρονα, δηλαδή το drone στέλνει την εικόνα και ο εξυπηρετητής απλά απαντά ότι την δέχτηκε, χωρίς να έχει εκτελέσει τον αλγόριθμο μηχανικής μάθησης ακόμα. Εκείνη τη στιγμή, ο εξυπηρετητής ξεκινάει ένα καινούριο νήμα εκτέλεσης και εκτελεί ασύγχρονα τον αλγόριθμο.

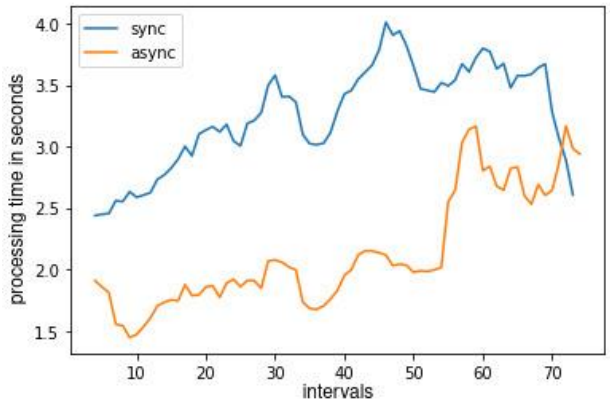


Figure 5.3 Sync Vs Async Image Recognition

Στο διάγραμμα 5.3 βλέπουμε ότι υπάρχει μία αισθητή διαφορά στο χρόνο απόκρισης και αυτό είναι λογικό καθώς πλέον το σύστημα επηρεάζεται κυρίως από το δίκτυο. Ωστόσο η διαφορά αν και είναι σημαντική, μπορούμε να πούμε ότι ακόμα και με ασύγχρονη εκτέλεση ο χρόνος εκτέλεσης δεν είναι αμελητέος (>1.5s).

Μια ακόμα παράμετρος που αλλάξαμε είναι η συμπίεση των εικόνων από την υπηρεσία που τρέχει πάνω στο drone. Συγκεκριμένα, στη βασική υλοποίηση του αλγορίθμου, πριν στείλει το drone τη φωτογραφία στο σταθμό βάσης την συμπιέζει για να μειώσει τον όγκο των δεδομένων που μεταφέρονται στο δίκτυο. Καθώς η συμπίεση θα μπορούσε να καταναλώνει επεξεργαστική ισχύ αποφασίσαμε να δοκιμάσουμε το σύστημα χωρίς αυτή.

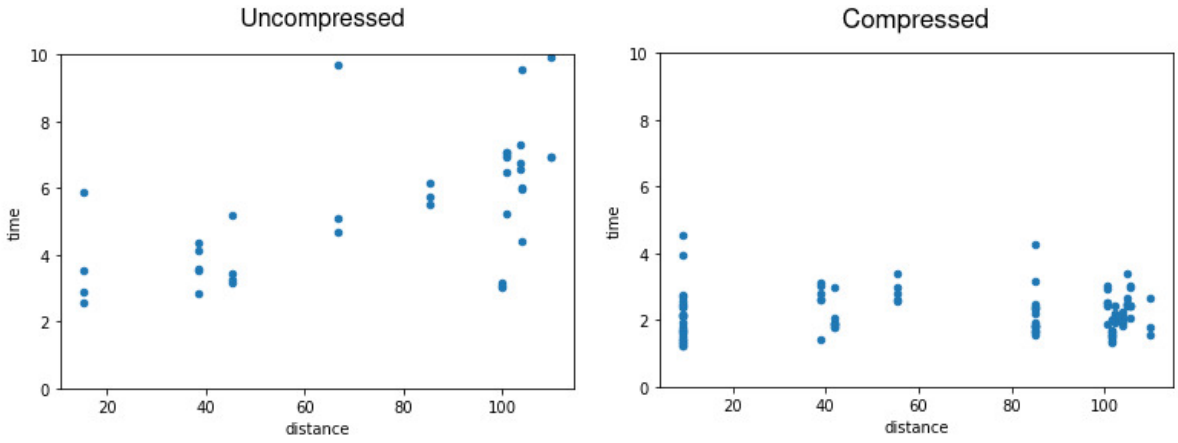


Figure 5.4 Uncompressed Vs Compressed Image transfer

Όπως βλέπουμε στην πιο πάνω εικόνα 5.4 τα δύο scatter plots δείχνουν το πώς επηρεάζει η απόσταση το χρόνο εκτέλεσης με και χωρίς συμπίεση της αρχικής εικόνας. Είναι ξεκάθαρο ότι η συμπίεση στη πλευρά του drone βοηθάει πολύ στη μείωση του χρόνου εκτέλεσης ειδικά στις μεγάλες αποστάσεις από το σταθμό βάσης. Κάτι ακόμα ενδιαφέρον είναι ότι όταν δεν έχουμε συμπίεση στα δεδομένα η απόσταση φαίνεται να είναι συσχετισμένη με το χρόνο εκτέλεσης σε αντίθεση με το όταν έχουμε συμπίεση.

Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα

Λαμβάνοντας υπόψιν τα πειράματα τα οποία έγιναν στο κεφάλαιο 5 μπορούμε να καταλήξουμε σε μερικά συμπεράσματα όσον αφορά το σύστημα το οποίο υλοποιήσαμε.

Όσο αφορά το κομμάτι της επεξεργαστικής ισχύς όπως είδαμε και από το πρώτο από τα πειράματα, είναι ο βασικός παράγοντας καθυστέρησης του συστήματος. Συγκεκριμένα ο χρόνος εκτέλεσης του αλγορίθμου μηχανικής μάθησης είναι ευθέως ανάλογος με την υπολογιστική ισχύ του σταθμού βάσης. Αν το σύστημα μας τρέχει σε μηχανή η οποία υπολείπεται επιταχυντή κάρτας γραφικών τότε ο χρόνος επεξεργασίας αυξάνεται αισθητά.

Στο επόμενο πείραμα στο οποίο δοκιμάσαμε την ασύγχρονη εκτέλεση του αλγορίθμου ανεξάρτητα του drone, παρατηρήσαμε ότι η διαφορά είναι σημαντική σε σχέση με την εκτέλεση του αλγορίθμου παράλληλα με την εκτέλεση της διαδρομής του drone. Παρόλη την μείωση του χρόνου εκτέλεσης όμως βλέποντας και από το σχήμα 5.2 μπορούμε να πούμε ότι πάλι ο χρόνος απόκρισης δεν είναι αμελητέος. Επομένως υπάρχει η βελτίωση με αυτή την εκδοχή του πειράματος αλλά βρισκόμαστε αρκετά μακριά από τον βέλτιστο χρόνο.

Στο τελευταίο μας πείραμα στο οποίο συγκρίναμε τον όγκο των δεδομένων τα οποία μεταφέρονται στο δίκτυο με την συμπίεση και μη συμπίεση των εικόνων. Φτάσαμε στο συμπέρασμα ότι η συμπίεση των εικόνων από την μεριά του drone

πριν την αποστολή της εικόνας επηρεάζει τον χρόνο εκτέλεσης ειδικά στις μεγάλες αποστάσεις από το σταθμό βάσης. Επίσης όταν δεν έχουμε συμπίεση στα δεδομένα η απόσταση φαίνεται να είναι συσχετισμένη με το χρόνο εκτέλεσης σε αντίθεση με το όταν έχουμε συμπίεση.

Σε γενικές γραμμές μετά από τα πειράματά μας μπορούμε να καταλήξουμε στο συμπέρασμα ότι όταν η μεταφορά δεδομένων είναι μεγάλη τότε πραγματικά η ποιότητα της σύνδεσης επηρεάζει πάρα πολύ το χρόνο επεξεργασίας σε edge δίκτυα.

6.2 Μελλοντική Δουλειά

Έχοντας υπόψιν το σύστημα το οποίο υλοποιήσαμε, με τα συγκεκριμένα εργαλεία υπάρχουν πολλοί τομείς οι οποίοι μπορούν να βελτιωθούν ή ακόμα και να αλλάξουν για να συνεχιστούν τα πειράματα σε αυτό το κομμάτι ενώ παράλληλα θα μπορούσε επίσης να προχωρήσει ένα βήμα παραπέρα.

Όσο αφορά το κομμάτι στο οποίο εργαστήκαμε θα μπορούσαν κάποια εργαλεία να βελτιωθούν και να αλλάξουν ούτως ώστε να έχουμε πιο ακριβής αποτελέσματα. Αρχικά θα μπορούσε να αλλάξει ο αλγόριθμος εντοπισμός αντικειμένου σε κάτι το οποίο θα ήταν υπολογιστικά πιο φτηνό και να μην έχει τόσο μεγάλη επίπτωση στον χρόνο απόκρισης. Επίσης θα μπορούσε να βελτιωθεί το περιβάλλον στο οποίο τρέχει το σύστημά μας εισάγοντας πιο περίπλοκα αντικείμενα στον χώρο για παράδειγμα έχοντας ένα αυτοκίνητο να κινείται σε μια διαδρομή, ή βάζοντας το σύστημα να τρέξει σε ένα πιο περίπλοκο περιβάλλον όπως σε αυτό μιας πόλης. Προσομοιώνοντας όμως το πιο περίπλοκο περιβάλλον και επιβαρύνοντας την μηχανή με το σύστημά μας, ανάλογα με τις προδιαγραφές της, η μηχανή θα είχε ανάλογη επίπτωση στον χρόνο και στην απόδοσή της.

Ένα άλλο κομμάτι που θα μπορούσε να υλοποιηθεί στα πλαίσια αυτής της εργασίας μελλοντικά αφού γίνουν αλλαγές στο unreal engine, είναι το containerization του όλου συστήματος για να μπορούν να γίνουν πιο εξονυχιστικά τα πειράματα στο fogify για να χρησιμοποιούνται τα δεδομένα real time και όχι στοιχεία τα οποία έχουν ήδη εξαχθεί από το AirSim και Unreal Engine.

Επιπρόσθετα μια υλοποίηση θα μπορούσε να είναι η ανάπτυξη αλγορίθμου για δυναμική δημιουργία μονοπατιού του drone για να μπορέσει μετά τον εντοπισμό του αντικειμένου να μπορέσει να ακολουθήσει το αντικείμενο. Αυτό θα μπορούσε

να υλοποιηθεί όμως μετά από αρκετή βελτιστοποίηση των χρόνων εκτέλεσης όπως είδαμε και από τα πειράματά μας αφού χρειάζεται ακόμα αρκετή βελτιστοποίηση για να μπορέσει να ακολουθήσει αποτελεσματικά το drone ένα αντικείμενο που κινείται αυθαίρετα.

Το τελευταίο κομμάτι το οποίο θα μπορούσε μεταγενέστερα να υλοποιηθεί μετά από βελτιστοποίηση όσων προαναφέραμε, είναι η υλοποίηση του συστήματος σε πραγματικό εξοπλισμό με σκοπό την παρακολούθηση πραγματικών χώρων πειραματικά. Αφού το AirSim έχει αναπτυχθεί με σκοπό την επέκταση του σε πραγματικά drones, τότε το σύστημα μας που αφορά την ανάλυση εικόνων και εντοπισμό αντικειμένου θα μπορούσε να εφαρμοστεί και να δοκιμαστεί σε πραγματικά drones.

Βιβλιογραφία

[1] AirSim Documentation. Available:

https://microsoft.github.io/AirSim/

[2] settings.json Documentation. Available:

https://microsoft.github.io/AirSim/settings/

[3] Unreal Engine Documentation. Available:

https://docs.unrealengine.com/en-US/index.html

[4] Blueprint Documentation. Available:

https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/index.html

[5] AirSim APIs Documentation. Available:

https://microsoft.github.io/AirSim/api_docs/html/

[6] Unreal Environments. Available:

https://www.unrealengine.com/marketplace/en-US/content-cat/assets/environments?count=20&sortBy=effectiveDate&sortDir=DESC&start=0

[7] Socket Documentation. Available:

https://docs.python.org/3/library/socket.html

[8] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis and M. D. Dikaiakos, "Fogify: A Fog Computing Emulation Framework," *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 42-54, doi: 10.1109/SEC50012.2020.00011.

[9] Blocks Environment. Available:

https://microsoft.github.io/AirSim/unreal_blocks/

[10] Type of Images:

https://microsoft.github.io/AirSim/image_apis/

[11] Google Maps Position:

https://www.google.com/maps/place/35%C2%B008'42.0%22N+33%C2%B024'34.6%22E/@35.1448932,33.4095854,19.75z/data=!4m5!3m4!1s0x0:0x0!8m2!3d35.145!4d33.4096

[12] DroneKit Documentation. Available:
https://dronekit-python.readthedocs.io/en/latest/

[13] Webots Documentation. Available:
https://cyberbotics.com/doc/guide/mavic-2-pro

[14] Grinberg, M. (2018). Flask web development: developing web applications with python. " O'Reilly Media, Inc."

[15] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[16] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," Transactions on Emerging Telecommunications Technologies, vol. 29, no. 11, 2018.

[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, 2011.

[18] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso, "Fogbed: A rapidprototyping emulation environment for fog computing," in 2018 IEEE International Conference on Communications (ICC), May 2018.

[19] Y. Zeng, M. Chao, and R. Stoleru, "Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments," in 2019 IEEE International Conference on Fog Computing (ICFC), 2019, pp. 153–164.

[20] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, “Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures,” CoRR, vol. abs/1709.07563, 2017.

[21] J. Hasenburg, M. Grambow, E. Grunewald, S. Huk, and D. Bermbach, ¨ “Mockfog: Emulating fog computing infrastructure in the cloud,” in 2019 IEEE International Conference on Fog Computing (ICFC), June 2019, pp. 144–152.

[22] Yolo Algorithm Documentation. Available: https://pjreddie.com/darknet/yolo/

[23] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

Appendix A

Κώδικας Drone

```
import airsim
import numpy as np
import os
import tempfile
import pprint
import cv2
import time
import socket
import time
import math
import argparse
import pyautogui
import json
from pathlib import Path
import matplotlib.pyplot as py
import yaml
import io
import shutil

imagesDepth=0
yaw=0

def createYamlFile(pathsRecorded):
    #Starting point of drone 0 lat lon on maps. 5th decimal is meters
    latMap=35.14500
    lonMap=33.40960

    maxLength = max(len(x) for x in pathsRecorded)
    ymlObj={
        "scenarios":[{
            "name":"mobility_scenario",
            "actions":[]
            }]
        }

    counter=0
    for t in range(maxLength):
        for dronesNum in range(len(pathsRecorded)):
            if len(pathsRecorded[dronesNum])>t:
                newlat=round(pathsRecorded[dronesNum][t][0]/100000,7)+latMap
                newlon=round(pathsRecorded[dronesNum][t][1]/100000,7)+lonMap
                reverseAlt=pathsRecorded[dronesNum][t][2]*(-1)
                dictionaryLatLonAlt={"network":"drone-network","lat": newlat, "lon": newlon,"alt":reverseAlt}
                moveDict={"type":"move","parameters": dictionaryLatLonAlt.copy()}
                dictionaryOutter={"time":t,"position":counter,"instance_type":"robot"+str(dronesNum+1),"instances":1,"action":moveDict}
                ymlObj["scenarios"][0]["actions"].append(dictionaryOutter.copy())
                counter+=1


    with open(r"C:\Users\User\Desktop\traces.yaml", 'w') as f:
        data = yaml.dump(ymlObj,f,sort_keys=False)

def getX(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.x_val,3)

def getY(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.y_val,3)

def getZ(robotName):
    pos=client.getMultirotorState(robotName).kinematics_estimated.position
    return round(pos.z_val,3)

def getTrace(robotName):
    return (getX(robotName),getY(robotName),getZ(robotName),round(client.getMultirotorState(robotName).timestamp,3))


def serverConnection(path):
    client2=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client2.connect(("127.0.0.1",8080))
    img=cv2.imread(path)
    f = io.BytesIO()
    np.savez_compressed(f,frame=img)
    f.seek(0)
    out = f.read()
    client2.sendall(out)
    from_server= from_server=(client2.recv(1024)).decode('utf-8')
    pos=json.loads(from_server)
    return pos
```

```
def landDrone(startX,startY,robotName):
    client.moveToPositionAsync(getX(robotName),getY(robotName),-25,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,robotName).join()
    client.moveToPositionAsync(startX,startY,-25,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,robotName).join()
    client.moveToPositionAsync(startX,startY,0,3,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,robotName).join()
    client.enableApiControl(False,robotName)


def takeImage(robotName):
    global images
    png_image=client.simGetImage("high_res", airsim.ImageType.Scene,robotName)
    n=robotName.replace('robot','')
    dir=robotName+"\\"+ str(images[int(n)-1])
    filename = os.path.join(tmp_dir, dir )
    images[int(n)-1]=images[int(n)-1]+1
    airsim.write_file(os.path.normpath(filename + '.png'), png_image)
    pngDirectory=os.path.normpath(filename + '.png')
    return pngDirectory

def pathsExecution(paths,numberOfPaths):
    xPositions = []
    yPositions = []
    for i in range (numberOfPaths):
        client.enableApiControl(True,"robot"+str(i+1))
        xPositions.append(getX("robot"+str(i+1)))
        yPositions.append(getY("robot"+str(i+1)))

    for i in range (numberOfPaths):
        client.armDisarm(True,"robot"+str(i+1))
    x1 = []
    for i in range (numberOfPaths):
        x1.append(client.takeoffAsync(5,"robot"+str(i+1)))
    for i in range (numberOfPaths):
        x1[i].join()

    result=[]
    pathsRecorded = []
    for i in range(numberOfPaths):
        result.append( client.moveOnPathAsync([airsim.Vector3r(int(paths[str(i+1)][str(0)][0]),int(paths[str(i+1)][str(0)][1]),int(paths[str(i+1)][str(0)][2])),
                                airsim.Vector3r(int(paths[str(i+1)][str(1)][0]),int(paths[str(i+1)][str(1)][1]),int(paths[str(i+1)][str(1)][2])),
                                airsim.Vector3r(int(paths[str(i+1)][str(2)][0]),int(paths[str(i+1)][str(2)][1]),int(paths[str(i+1)][str(2)][2])),
                                airsim.Vector3r(int(paths[str(i+1)][str(3)][0]),int(paths[str(i+1)][str(3)][1]),int(paths[str(i+1)][str(3)][2])),
                                airsim.Vector3r(int(paths[str(i+1)][str(4)][0]),int(paths[str(i+1)][str(4)][1]),int(paths[str(i+1)][str(4)][2]))],
                        paths[str(i+1)]["vel"], paths[str(i+1)]["time-out"],
                        airsim.DrivetrainType.MaxDegreeOfFreedom, airsim.YawMode(True,paths[str(i+1)]["degPerSec"]), -1, 1,"robot"+str(i+1)))
        pathsRecorded.append([])

    start = time.time()
    time.perf_counter()
    elapsed = 0
    blacklist=[]
    while elapsed < 170:

        for i in range(numberOfPaths):
            if (i+1) not in blacklist:
                pathsRecorded[i].append(getTrace('robot'+str(i+1)))
                pos=serverConnection(takeImage('robot'+str(i+1)))
                if (pos[0]!=0) or (pos[1]!=0):
                    client.cancelLastTask('robot'+str(i+1))
                    blacklist.append(i+1)
        oldElapsed=elapsed
        elapsed = time.time() - start
        oneIterationTime=elapsed-oldElapsed
        print(oneIterationTime)
        if len(blacklist) == numberOfPaths:
            break



    x2 = []
    for i in range (numberOfPaths):
        x2.append(client.moveToPositionAsync(getX("robot"+str(i+1)),getY("robot"+str(i+1)),-25-i,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
    for i in range (numberOfPaths):
        x2[i].join()
    x3 = []
    for i in range (numberOfPaths):
        x3.append(client.moveToPositionAsync(xPositions[i],yPositions[i],-25-i,5,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
    for i in range (numberOfPaths):
        x3[i].join()
    x4= []
    for i in range (numberOfPaths):
        x4.append(client.moveToPositionAsync(xPositions[i],yPositions[i],0,3,30,airsim.DrivetrainType.MaxDegreeOfFreedom,airsim.YawMode(False,0),-1,1,"robot"+str(i+1)))
    for i in range (numberOfPaths):
        x4[i].join()
    for i in range (numberOfPaths):
        client.enableApiControl(False,"robot"+str(i+1))


    return pathsRecorded


numberOfPaths=int(input("Enter number of paths: "))

jsnPath = input("Enter paths.json location: ") #C:\Users\User\Documents\AirSim\path.json

with open(jsnPath, 'r') as myfile:
    data=myfile.read()

paths = json.loads(data)

client = airsim.MultirotorClient()
client.confirmConnection()

tmp_dir = os.path.join(tempfile.gettempdir(), "airsim_drone")
if os.path.exists(tmp_dir):
    shutil.rmtree(tmp_dir)
os.makedirs(tmp_dir)
images = []
for i in range(numberOfPaths):
    directory = "robot"+str(i+1)
    path = os.path.join(tmp_dir, directory)
    try:
        os.makedirs(path)
    except OSError:
        if not os.path.isdir(path):
            raise

    images.append(0)



pathsRecorded=pathsExecution(paths,numberOfPaths)
createYamlFile(pathsRecorded)




airsim.wait_key('Press any key to reset to original state')
client.armDisarm(False)
client.reset()
client.enableApiControl(False)
```

Appendix B

Κώδικας Server

```
import cv2
import argparse
import numpy as np

from flask import Flask, request

# np_load_old = np.load

# modify the default parameters of np.load
# np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

def yolov3(yolo_weights, yolo_cfg, coco_names):
    net = cv2.dnn.readNet(yolo_weights, yolo_cfg)
    classes = open(coco_names).read().strip().split("\n")
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    return net, classes, output_layers


def perform_detection(net, img, output_layers, w, h, confidence_threshold):
    blob = cv2.dnn.blobFromImage(img, 1 / 255., (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    layer_outputs = net.forward(output_layers)

    boxes = []
    confidences = []
    class_ids = []

    for output in layer_outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            # Object is deemed to be detected
            #if confidence > confidence_threshold:
            if confidence > 0:
                # center_x, center_y, width, height = (detection[0:4] * np.array([w, h, w, h])).astype('int')
                center_x, center_y, width, height = list(map(int, detection[0:4] * [w, h, w, h]))
                # print(center_x, center_y, width, height)

                top_left_x = int(center_x - (width / 2))
                top_left_y = int(center_y - (height / 2))

                boxes.append([top_left_x, top_left_y, width, height])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    return boxes, confidences, class_ids


def draw_boxes(image_path, boxes, confidences, class_ids, classes, img, colors, confidence_threshold, NMS_threshold):

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, confidence_threshold, NMS_threshold)

    FONT = cv2.FONT_HERSHEY_SIMPLEX
    pointA=0
    pointB=0
    if len(indexes) > 0:
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            color = colors[i]
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
            text = "{}: {:.4f}".format(classes[class_ids[i]], confidences[i])
            cv2.putText(img, text, (x, y - 5), FONT, 0.5, color, 2)
            if classes[class_ids[i]]=="orange":
                pointA=int((x+(x+w))/2)
                pointB=int((y+(y+h))/2)

    return pointA,pointB



def detection_image_file(image_path, yolo_weights, yolo_cfg, coco_names, confidence_threshold, nms_threshold):
    #img, h, w = load_input_image(image_path)
    img=image_path
    h, w, _ = img.shape
    net, classes, output_layers = yolov3(yolo_weights, yolo_cfg, coco_names)
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    boxes, confidences, class_ids = perform_detection(net, img, output_layers, w, h, confidence_threshold)
    return draw_boxes(image_path, boxes, confidences, class_ids, classes, img, colors, confidence_threshold, nms_threshold)
```

```
if __name__ == '__main__':
    ap = argparse.ArgumentParser()
    ap.add_argument('--image', help='Path to the test images', default='img.png')
    ap.add_argument('--weights', help='Path to model weights', type=str, default='yolov3.weights')
    ap.add_argument('--configs', help='Path to model configs',type=str, default='yolov3.cfg')
    ap.add_argument('--class_names', help='Path to class-names text file', type=str, default='coco.names')
    ap.add_argument('--conf_thresh', help='Confidence threshold value', default=0.5)
    ap.add_argument('--nms_thresh', help='Confidence threshold value', default=0.4)
    args = vars(ap.parse_args())
    #image_path = args['image']
    yolo_weights, yolo_cfg, coco_names = args['weights'], args['configs'], args['class_names']
    confidence_threshold = args['conf_thresh']
    nms_threshold = args['nms_thresh']
    #detection_image_file(image_path, yolo_weights, yolo_cfg, coco_names, confidence_threshold, nms_threshold)

    app = Flask(__name__)


    @app.route('/', methods=['POST'])
    def upload_file():
        if request.method == 'POST':
            if 'image' not in request.files:
                return 'there is no image in form!'
            image = request.files['image']
            final_image = np.load(image, allow_pickle=True)['frame']
            # try:
            pointA, pointB = detection_image_file(final_image, yolo_weights, yolo_cfg, coco_names,
                                                  confidence_threshold, nms_threshold)
            response = "[" + str(pointA) + "," + str(pointB) + "]"
            print("client disconnected")
            return response
            # except Exception as ex:
            #     print("Could not load image")
            return '{"error": "something goes wrong"}'

    print("starting")
    app.run("0.0.0.0",8080)
```

Appendix C

Κώδικας Edit Settings

```
import numpy as np
import matplotlib.pyplot as py
import os
import cv2
from pathlib import Path
import json


def settingsSetup(numberOfDrones):
    settingsPath = Path("C:/Users/User/Documents/AirSim/settings.json")
    robotPath=Path("C:/Users/User/Documents/AirSim/addRobot.json")
    with open(settingsPath, 'r') as myfile:
        data=myfile.read()
    with open(robotPath, 'r') as myfile:
        data2=myfile.read()
    settings = json.loads(data)
    robot = json.loads(data2)



    if len(settings['Vehicles'])>1 :
        for element in settings["Vehicles"].copy():
            if element != 'robot1':
                settings["Vehicles"].pop(element, None)
    for i in range (2,numberOfDrones+1):
        settings['Vehicles']['robot'+str(i)]=robot['robot1']
    with open("settings.json", "w") as jsonFile:
        json.dump(settings, jsonFile)


    pathsPath=Path("C:/Users/User/Documents/AirSim/path.json")
    with open(pathsPath, 'r') as myfile2:
        data3=myfile2.read()
    paths=json.loads(data3)
    for element in paths.copy():
        if (int(element)!=1) and (int(element)!=2):
                paths.pop(element,None)
    with open("path.json", "w") as jsonFile:
        json.dump(paths, jsonFile)

    if numberOfDrones>2:
        for i in range(3,numberOfDrones+1,2):
            paths[i]=paths['1']
            paths[i+1]=paths['2']
    with open("path.json", "w") as jsonFile:
        json.dump(paths, jsonFile)


with open(pathsPath, 'r') as myfile2:
    data3=myfile2.read()
paths=json.loads(data3)
for i in range(numberOfDrones):
    paths[str(i+1)][str(0)][1]=str(int(paths[str(i+1)][str(0)][1])-3*i)
    for k in range(5):
        paths[str(i+1)][str(k)][2]=str(-7-i)


with open("path.json", "w") as jsonFile:
    json.dump(paths, jsonFile)




    with open(settingsPath, 'r') as myfile:
        data=myfile.read()
    settings = json.loads(data)

    space=3
    yaxis=0
    for x in range (2,len(settings['Vehicles'])+1):
        settings['Vehicles']['robot'+str(x)]['Y']=space
        space=space+3


    with open("settings.json", "w") as jsonFile:
        json.dump(settings, jsonFile)




numberOfDrones=int(input("Enter number of paths: "))
settingsSetup(numberOfDrones)
```

Appendix D

Κώδικας Workload Generator

```
import socket
import cv2
import time
import os
import numpy as np
import io
import yaml
import requests

server_ip = os.getenv("SERVER_IP", "127.0.0.1")
server_port = int(os.getenv("SERVER_PORT", "8080"))
numberOfPaths = int(os.getenv("NUMBER_OF_DRONES"))
parentPath = f"/code/{numberOfPaths}drone1server" #input("Enter files location: ") #C:\Users\User\Desktop\3drones1server
tracePath = parentPath+"/traces.yaml"
numberOfDrones = os.getenv("DRONE_NUM", 1)
FOGIFY_SERVER = os.getenv("FOGIFY_SERVER")

def connect(image):
    url = f'http://{server_ip}:{server_port}'
    # files = {'media': open('test.jpg', 'rb')}

    client=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_ip,server_port))
    f = io.BytesIO()
    np.savez_compressed(f,frame=image)
    f.seek(0)
    out = f.read()
    from_server=requests.post(url, files={'image': out})
    # client.sendall(out)

    # from_server=(client.recv(1024)).decode('utf-8')
    # client.close()
    return from_server.text




# imagesListNumber=[]
eachPath=parentPath+"/robot"+str(numberOfDrones)
path, dirs, files = next(os.walk(eachPath))
file_count = len(files)
    # imagesListNumber.append(file_count)

def get_location():
    if FOGIFY_SERVER:
        return requests.get(f"http://{FOGIFY_SERVER}:5600/network/drone_network/robot1").json()
    else:
        with open(tracePath) as f:
            traceData = yaml.safe_load(f)
            return dict(alt=traceData["scenarios"][0]["actions"][i]["action"]["parameters"]["alt"],
                        lat=traceData["scenarios"][0]["actions"][i]["action"]["parameters"]["lat"],
                        lon=traceData["scenarios"][0]["actions"][i]["action"]["parameters"]["lon"])
        return {}
print(file_count)
print("robot,count,lat,lon,alt,time,response")
#Correlate counter with positions in yaml file because each position is corelated only once, and that is how the data are appended to the file
while(True):
    for i in range(file_count):
        start = time.time()
        time.perf_counter()
        elapsed = 0
        pathToImage=parentPath+"/robot"+str(numberOfDrones)+"/"+str(i)+".png"
        img=cv2.imread(pathToImage)
        response=connect(img)
        try:
            location = get_location()
        except Exception as e:
            location = {}
        lat = location.get('lat')
        lon = location.get('lon')
        alt = location.get('alt')
        elapsed = time.time() - start
        print(f"{numberOfDrones},{i},{lat},{lon},{alt},{elapsed},{response}")
        time.sleep(5)
```

Appendix E

Κώδικας Dockerfiles

Workload Generator Dockerfile

```
# Pull base image of latest Python 3.7.x
FROM python:3.6

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN git clone https://github.com/opsengine/cpulimit/
WORKDIR cpulimit
RUN make && cp src/cpulimit /usr/bin

RUN apt-get update ##[edited]
RUN apt-get install ffmpeg libsm6 libxext6  -y

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
EXPOSE 8080
ADD ./code/ /code/
ADD ./1drone1server /code/1drone1server
ADD ./3drones1server /code/3drone1server
RUN chmod +x /code/entrypoint.sh
CMD ["/bin/bash", "/code/entrypoint.sh"]
```

Server Dockerfile

```
# Pull base image of latest Python 3.7.x
FROM python:3.6

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN git clone https://github.com/opsengine/cpulimit/
WORKDIR cpulimit
RUN make && cp src/cpulimit /usr/bin

RUN apt-get update ##[edited]
RUN apt-get install ffmpeg libsm6 libxext6  -y

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt


EXPOSE 8080
# cron tab
ADD ./code/ /code/
ADD ./models/coco.names /code/
ADD ./models/yolov3.cfg /code/
ADD ./models/yolov3.weights /code/
#ADD crontab /etc/cron.d/crontab_file
# RUN chmod 0644 /code/cronjobs

#try to keep logs
# RUN echo "cron.*                          /var/log/cron.log" >> /etc/rsyslog.conf
# RUN service rsyslog start
#ADD code/entrypoint.sh /code/
#COPY code/entrypoint.sh /code/entrypoint.sh
RUN chmod +x /code/entrypoint.sh
CMD ["/bin/bash", "/code/entrypoint.sh"]
```