ΠΟΛΥΤΕΧΝΙΚΕΣ ΣΧΟΛΕΣ

Τμήμα Μηχανολόγων & Αεροναυπηγών Μηχανικών
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Ρομποτική

Τμήματος Μηχανολόγων Μηχανικών

Drones: Υλοποίηση Βασικών Τροχιών, Σύνθετων Περιπτώσεων, Εντοπισμός Προσώπων και Swarming

Ονοματεπώνυμο:	*Α.Μ.:*	*Έτος:*	*Τμήμα:*
Αριστείδης Ανδρουτσόπουλος	1054274	6ο	Η/Υ (CEID)
Ιωάννης Τσικέλης	1067407	4ο	Η/Υ (CEID)
Παναγιώτης Γεωργάκης	1067233	4ο	Μηχ. Μηχ. (MEAD)

03/07/2022

Table of Contents

Εισαγωγή

Σκοπός της παρούσας εργασίας εξαμήνου του μαθήματος της Ρομποτικής, είναι ο προγραμματισμός ενός DJI Tello quadcopter drone για την επίτευξη μιας σειράς λειτουργιών, οι οποίες χωρίζονται σε τρεις θεματολογίες.

1. Η πρώτη θεματολογία αφορά την διάνυση μιας πληθώρας απλών και σύνθετων τροχιών, κάνοντας χρήση εντολών του λογισμικού του drone που βασίζονται σε ήδη γραμμένους ελεγκτές χαμηλού επιπέδου.

2. Η δεύτερη θεματολογία αφορά τον προγραμματισμό ελεγκτών υψηλού επιπέδου για τον εντοπισμό και παρακολούθηση προσώπων στο γειτονικό περιβάλλον του drone.

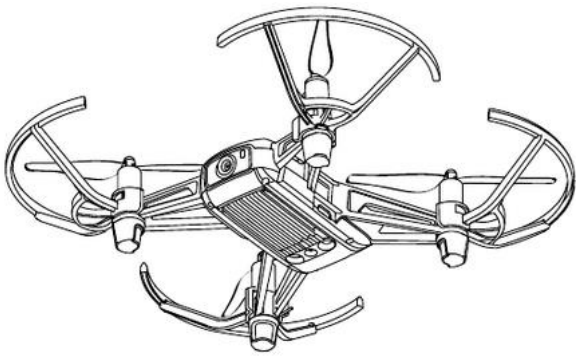
3. Η τρίτη θεματολογία αφορά τον προγραμματισμό ελεγκτών υψηλού επιπέδου και λειτουργίας swarming για την μεταφορά ενός φορτίου από ένα σημείο σε ένα άλλο.

Κατά την περάτωση των παραπάνω θεματολογιών, καταγράφονται παρατηρήσεις για τυχόν αποκλίνουσες συμπεριφορές και διαμορφώνονται τα κατάλληλα συμπεράσματα για την κίνηση, τους αισθητήρες και τον έλεγχο του Tello drone.

1. DJI Tello Drone

1.1. Εισαγωγή

Το DJI Tello (EDU) drone της εταιρείας Ryze Technology είναι ένα μικρών διαστάσεων quadcopter, το οποίο μπορεί να ελέγχεται τόσο μέσω χειριστηρίου ή εφαρμογής κινητού, όσο και μέσω κώδικα υψηλού επιπέδου, υποστηρίζοντας προγραμματισμό σε Scratch, Swift και Python. Ο έλεγχος του Tello κατά το πέρας της εργασίας πραγματοποιείται στο περιβάλλον της Python, λόγω της ευελιξίας της και της πληθώρας open-source βιβλιοθηκών, τις οποίες μπορεί να αξιοποιήσει.



Σχήμα 1: Ισομετρικό σκαρίφημα ενός DJI Tello drone

1.2. Αισθητήρες

Το Tello αξιοποιεί ένα μεγάλο εύρος αισθητήρων για την λειτουργία του, οι οποίοι απαριθμούνται παρακάτω:

1. Μπροστινή κάμερα (έγχρωμη) των 5 MP (*Megapixels*) με δυνατότητα καταγραφής βίντεο ανάλυσης 720p και οπτικού πεδίου (FOV) 82.6ο.

2. Κάμερα (ασπρόμαυρη) με κατεύθυνση προβολής προς τα κάτω για την αναγνώριση του εδάφους και της κάτοψης οποιασδήποτε παρεμβαλόμενης γεωμετρίας αναφοράς.

3. Αισθητήρες IR με κατεύθυνση προβολής προς τα κάτω για την μέτρηση ύψους.

4. Αισθητήρας IMU (*Inertial Measurement Unit*) ο οποίος αξιοποιεί τις μετρήσεις του υποσυστήματος των παρακάτω αισθητήρων:

    4.1. Επιταχυνσιόμετρο (*Accelerometer*) το οποίο μετρά την πραγματική επιτάχυνση του drone ως προς την βαρυτική του επιτάχυνση.

    4.2. Βαρόμετρο (*Barometer*) το οποίο μετράει την ατμοσφαιρική πίεση και συνεπώς το ύψος του drone.

    4.3. Γυροσκόπιο (*Gyroscope*) για την αναγνώριση του προσανατολισμού του drone στους τρεις κύριους άξονες περιστροφής του.

    4.4. Θερμόμετρο (*Thermometer*) για την μέτρηση της θερμοκρασίας του drone.

Όλες οι βαθμίδες του IMU απαιτούν βαθμονόμηση (calibration) στις απαιτούμενες συνθήκες λειτουργίας για ομαλή πτήση και έλεγχο.

1.3. Tello SDK και DJITelloPy

Όπως αναφέρθηκε και Ενότητα 1.1, το Tello ελέγχεται ασύρματα μέσω προγραμματισμού σε Python. Για να επιτευχθεί αυτό γίνεται χρήση του Tello SDK (Software Development Kit), ένα σετ απλοϊκών εντολών που μπορούν να σταλθούν στο Tello μέσω Wi-Fi, οι οποίες δίνουν την δυνατότητα πραγματοποίησης διαφόρων πράξεων. Οι εντολές αυτές αφορούν βασικές λειτουργίες όπως προσγείωση, απογείωση και ενεργοποίηση κάμερας, μεταβολή κάθε μεμονωμένου βαθμού ελευθερίας του Tello (μετατοπίσεις XYZ, γωνίες: RPY), καθώς και πραγματοποίηση βασικών ευθύγραμμων και κυκλικών τροχιών.

Για να γίνει χρήση των εντολών του Tello SDK καθώς και περισσότερων δυνατοτήτων στο περιβάλλον της Python, γίνεται επιπλέον χρήση της βιβλιοθήκης DJITelloPy[1] των **Damià Fuentes Escoté και Jakob Löw**. Η βιβλιοθήκη αυτή προσφέρει όλες τις βασικές εντολές του Tello SDK, ενώ παράλληλα απλοποιεί την συγγραφή κώδικα για πιο περίπλοκες λειτουργίες, όπως την επεξεργασία βίντεο από την κάμερα, την λειτουργία σμήνους drone (swarming) και την υλοποίηση ελεγκτών υψηλού επιπέδου.

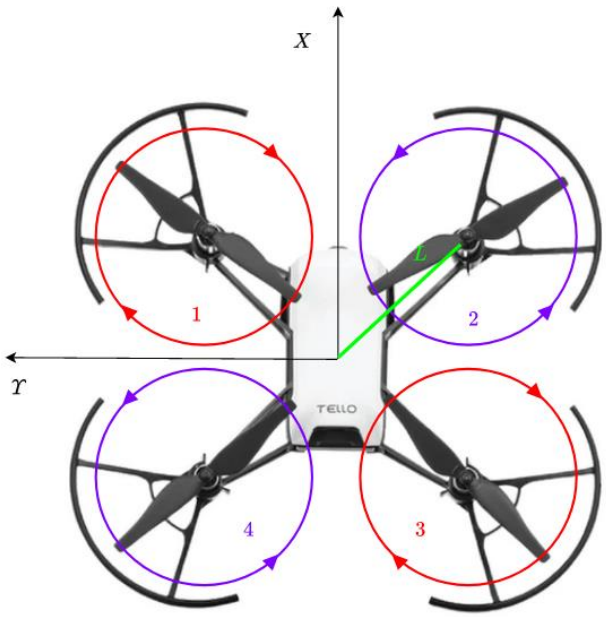
[1] ***DJITelloPy GitHub repo link***: https://github.com/damiafuentes/DJITelloPy

2. Μοντελοποίηση ενός Quadcopter Drone

Για λόγους πληρότητας, αλλά και με σκοπό την γεφύρωση της ύλης του μαθήματος με το αντικείμενο της μηχανικής και του ελέγχου ενός quadcopter drone, στην παρούσα ενότητα παρουσιάζεται μια απόπειρα εύρεσης του κινηματικού και δυναμικού του μοντέλου.

2.1. Αρχή λειτουργίας (θεώρηση Black Box)

Προτού ξεκινήσει η διαδικασία κινηματικής και δυναμικής μοντελοποίησης του quadcopter drone, είναι σκόπιμο να αναλυθεί ο τρόπος με τον οποίο αυτό λειτουργεί. Για αρχή το drone θεωρείται ως Μαύρο Κουτί (Black Box), δηλαδή η φυσική του κινηματικού και του δυναμικού συστήματος παραμένει άγνωστη, ενώ εξετάζεται πως οι είσοδοι του συστήματος (ταχύτητες κινητήρων/ελίκων) επηρεάζουν τις εξόδους (ταχύτητες του drone).
Στο Σχήμα 2 παρουσιάζεται η κάτοψη ενός Tello με αριθμημένο κάθε ένα από τους τέσσερις κινητήρες που χρησιμοποιεί για πτήση.



Σχήμα 2: Κάτοψη ενός Tello quadcopter drone με σημειωμένες τις φορές περιστροφής των ελίκων

Στο Σχήμα 2 παρουσιάζονται οι άξονες αναφοράς ενός Tello, καθώς και οι φορές περιστροφής των ελίκων του. Όπως γίνεται αντιληπτό, η φορά περιστροφής των ελίκων ενός quadcopter είναι αντίστροφη ανά ζευγάρια. Η έλικες 1 και 3 περιστρέφονται ωρολογιακά (CW) ενώ οι έλικες 2 και 4 περιστρέφονται ανθωρολογιακά (CCW). Αυτό συμβαίνει ώστε το quadcopter να έχει μηδενική στροφορμή όταν οι έλικες περιστρέφονται κατά μέτρο με την ίδια γωνιακή ταχύτητα. Προφανώς αυτό προϋποθέτει πως οι έλικες 1 και 3 είναι γεωμετρικά ανεστραμμένες σε σχέση με τις 2 και 4, ώστε να παράγουν και οι τέσσερις ώση (thrust).

Γνωρίζοντας τα παραπάνω, γίνεται εύκολο να εξαχθούν οι συνδυασμοί περιστροφών των ελίκων που απαιτούνται ώστε να επιτευχθούν οι κύριες κινήσεις του quadcopter:

- Εάν η συνολική ώση του quadcopter (άθροισμα ώσεων των τεσσάρων ελίκων) είναι μεγαλύτερη από την δύναμη της βαρύτητας, τότε το quadcopter ανασηκώνεται προς τα πάνω. Αν είναι μικρότερη,

τότε το ρομπότ χαμηλώνει προς τα κάτω. Στην περίπτωση όπου η ώση ισούται με την δύναμη της βαρύτητας, τότε το quadcopter ισορροπεί σε σταθερό ύψος.

- Αν η συνολική ώση των ελίκων 1 και 4 είναι μεγαλύτερη της ώσης των 2 και 3, τότε το quadcopter μετατοπίζεται δεξιά. Αν είναι μικρότερη, τότε μετατοπίζεται αριστερά. Εάν είναι ίσες, το quadcopter δεν μετακινείται στην διεύθυνση του άξονα Y.

- Αν η συνολική ώση των ελίκων 1 και 2 είναι μεγαλύτερη της ώσης των 3 και 4, τότε το quadcopter μετατοπίζεται πίσω. Αν είναι μικρότερη, τότε μετατοπίζεται μπροστά. Εάν είναι ίσες, το quadcopter δεν μετακινείται στην διεύθυνση του άξονα X.

- Αν η συνολική ώση των ελίκων 1 και 3 είναι μεγαλύτερη της ώσης των 2 και 4, τότε το quadcopter περιστρέφεται ανθωρολογιακά. Αν είναι μικρότερη, τότε περιστρέφεται ωρολογιακά. Αν είναι ίσες, τότε δεν υπάρχει περιστροφή ως προς τον κατακόρυφο άξονα Z.

2.2. Κινηματικές εξισώσεις

Αρχικά ορίζονται το διάνυσμα μετατοπίσεων $\mathbf{x} = [x, y, z]^{\boldsymbol{T}}$ και το διάνυσμα περιστροφών RPY (Roll-Pitch-Yaw) $\boldsymbol{\theta} = [\varphi, \theta, \psi]^{\boldsymbol{T}}$.

Για την περιστροφή από το αδρανειακό σύστημα συντεταγμένων στο σύστημα συντεταγμένων του Tello, γίνεται χρήση του πίνακα περιστροφής κατά σύμβαση ZYZ:

$$\mathbf{R} = \begin{bmatrix} c\varphi c\psi - c\theta s\varphi s\psi & -c\varphi s\varphi - c\varphi c\theta s\psi & s\theta s\varphi \\ c\theta c\psi s\varphi & c\varphi c\theta c\psi - s\varphi s\psi & -c\psi s\theta \\ s\varphi s\theta & c\varphi s\theta & c\theta \end{bmatrix} \quad (1)$$

Το διάνυσμα των γωνιακών $\boldsymbol{\omega}$ ταχυτήτων του Tello συνδέεται με το διάνυσμα των περιστροφών μέσω της παρακάτω σχέσης (Ιακωβιανή περιστροφικών συνιστώσεων):

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\varphi & c\theta s\varphi \\ 0 & -s\varphi & c\theta c\varphi \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (2)$$

2.3. Κινητήρες

Οι κινητήρες του Tello είναι συνεχούς ρεύματος (RCGEEK Drone 8520 DC Motors Coreless), η παραγόμενη ροπή των οποίων δίνεται από την σχέση:

$$\tau = k_t(I - I_o) \quad (3)$$

Όπου τ η ροπή του κινητήρα, I το ρεύμα εισόδου στο κινητήρα, I_o το ρεύμα στο κινητήρα δίχως εφαρμογή φορτίου (no load current) και k_t ο συντελεστής αναλογίας ροπής.

Η μετρούμενη τάση ενός brushless DC κινητήρα ισούται με το άθροισμα της αντιηλεκτρεγερτικής δύναμης (back-EMF) και των ενεργειακών απωλειών λόγω αντίστασης:

$$V = IR_m + k_v\omega \tag{4}$$

Όπου V η πτώση τάσης στον κινητήρα, R_m η αντίστασή του, ω η γωνιακή ταχύτητα του κινητήρα και k_v ο συντελεστής αναλογίας που αντιστοιχεί στην παραγόμενη αντιηλεκτρεγερτική δύναμη ανά στροφή (back-EMF/RPM).

Από τις σχέσεις (3) και (4) μπορεί να βρεθεί η ηλεκτρική ισχύς που καταναλώνει ο κινητήρας συναρτήσει της ροπής και της γωνιακής ταχύτητας:

$$\begin{aligned}(3),(4) \rightarrow P_e &= IV \\ &= \frac{(\tau + k_t I_o)(k_t I_o R_m + \tau R_m + k_t k_v \omega)}{k_t^2}\end{aligned} \tag{5}$$

Θεωρώντας πως η αντίσταση R_m του κινητήρα και το ρεύμα δίχως εφαρμογή φορτίου I_o είναι πολύ μικρές ποσότητες (βάσιμη παραδοχή για έναν brushless DC κινητήρα), ο υπολογισμός της ηλεκτρικής ισχύς απλοποιείται μέσω της παρακάτω σχέσης:

$$P_e = \frac{k_v}{k_t}\tau\omega \tag{6}$$

2.4. Δυνάμεις

Σύμφωνα με την αρχή διατήρησης της ενέργειας, η ηλεκτρική ισχύς που καταναλώνει ο κινητήρας πρέπει να ισούται με την μηχανική ισχύ ικανή πτήσεως, δηλαδή την ώση T επί την ταχύτητα του αέρα διαμέσου της έλικας V_h.

$$P_e = TV_h \tag{7}$$

Αν ρ η πυκνότητα του αέρα και A η επιφάνεια διατομής προσρόφησης του αέρα στην έλικα, τότε η ταχύτητα V_h παρουσιάζει την εξής σχέση με την ώση T:

$$V_h = \sqrt{\frac{T}{2\rho A}} \tag{8}$$

Συνδυάζοντας τις σχέσεις (7) και (8), η ηλεκτρική ισχύς του κινητήρα θα ισούται με:

$$(7), (8) \rightarrow P_e = \frac{T^{2/3}}{\sqrt{2\rho A}} \tag{9}$$

Δεδομένου ότι η ροπή του κινητήρα είναι ανάλογη της ώσης ($\tau = k_\tau T$, όπου k_τ ένας συντελεστής που εξαρτάται από την γεωμετρική διαμόρφωση των λεπίδων της έλικας), μέσω των σχέσεων (6) και (10) μπορεί να βρεθεί η εξάρτηση της ώσης μιας έλικας από την γωνιακή ταχύτητα του κινητήρα:

$$(6), (10) \rightarrow T = \left(\frac{k_v k_\tau \sqrt{2\rho A}}{k_t}\right)^2 \omega^2 = k\omega^2 \tag{10}$$

Αθροίζοντας τις ώσεις όλων των κινητήρων, το διάνυσμα της συνολικής ώσης του Tello ως προς το τοπικό του σύστημα συντεταγμένων θα ισούται με:

$$\mathbf{T}_{Tello} = k\left[0\,,\ \ 0\,,\ \ \sum_{i=1}^{4} \omega_i^2\right]^{\boldsymbol{T}} \tag{11}$$

Εκτός από την ωστική δύναμη, πρέπει να μοντελοποιηθεί και η δύναμη της τριβής που επιδρά στο drone λόγω του αέρα. Η τριβή ρευστών μπορεί να απλοποιηθεί μέσω της θεώρησης ιξώδους τριβής και στις τρεις κύριες διευθύνσεις, όπου c ο συντελεστής ιξώδους τριβής:

$$\mathbf{F}_d = -c[\dot{x},\ \ \dot{y},\ \ \dot{z}]^{\boldsymbol{T}} \tag{11}$$

2.5. Ροπές

Αν C_D ο γεωμετρικός συντελεστής οπισθέλκουσας (drag coefficient) και R η ακτίνα της έλικας, τότε η ροπή που παράγεται λόγω της οπισθέλκουσας δύναμης (drag) μιας έλικας ως προς τον άξονα Z δίνεται από την σχέση:

$$\tau_D = \frac{1}{2} R^3 \rho C_D A \omega^2 = b\omega^2 \tag{12}$$

Επομένως, η συνολική ροπή ως προς τον άξονα Z προκύπτει από το άθροισμα των ροπών κάθε έλικας, δίνοντας προσοχή στην φορά περιστροφής:

$$\tau_\psi = b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{13}$$

Οι ροπές γύρω από τους άξονες X και Y προέρχονται εξολοκλήρου από τις ωστικές δυνάμεις στον άξονα Z, οι οποίες δίνονται από την εξίσωση (11). Αν L η απόσταση της έλικας από το κέντρο μάζας του Tello, τότε προκύπτει το διάνυσμα των ροπών:

$$\boldsymbol{\tau}_{Tello} = \begin{bmatrix} \tau_\varphi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} kL(\omega_1^2 - \omega_3^2) \\ kL(\omega_2^2 - \omega_4^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{14}$$

2.6. Εξισώσεις κίνησης

Αρχικά αθροίζονται διανυσματικά όλες οι δυνάμεις που επιδρούν στο Tello σύμφωνα με τον 2ο Νόμο του Newton.:

$$\ddot{\mathbf{x}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R}{m}\mathbf{T}_{Tello} + \frac{1}{m}\mathbf{F}_d \tag{15}$$

Αντίστοιχα γίνεται χρήση των δυναμικών εξισώσεων Euler-Newton ως προς το τοπικό σύστημα συντεταγμένων υποθέτοντας πως το Tello είναι μη-παραμορφώσιμο στερεό:

$$\dot{\boldsymbol{\omega}} = \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \mathbf{I}^{-1}\big(\boldsymbol{\tau}_{Tello} - \boldsymbol{\omega} \times (\mathbf{I} \cdot \boldsymbol{\omega})\big) \tag{16}$$

Για την εύρεση του μητρώου αδράνειας $\mathbf{I}$, το Tello μοντελοποιείται ως δύο λεπτές (αμελητέας διαμέτρου) ομοιόμορφες ράβδοι που τέμνονται στα κέντρα τους με το κέντρο του τοπικού συστήματος συντεταγμένων. Θεωρείται πως όλη η μάζα του Tello είναι ομοιόμορφα κατανεμημένη σημειακά σε κάθε έναν από τους τέσσερεις κινητήρες.

Επομένως το μητρώο αδράνειας αποτελείται εξολοκλήρου από διαγώνια στοιχεία ροπών αδράνειας, ενώ τα μη-διαγώνια στοιχεία γινομένων αδράνειας μηδενίζονται:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{17}$$

Επομένως η εξίσωση που εκφράζει τις περιστροφές γύρω από το τοπικό σύστημα συντεταγμένων ορίζεται ως εξής:

$$\dot{\boldsymbol{\omega}} = \begin{bmatrix} \frac{\tau_\varphi}{I_{xx}} \\ \frac{\tau_\theta}{I_{yy}} \\ \frac{\tau_\psi}{I_{zz}} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix} \tag{18}$$

Με τις εξισώσεις κίνησης γνωστές, τώρα το μοντέλο μπορεί να τεθεί στον Χώρο Κατάστασης (State Space) ώστε να εφαρμοστούν πάνω του νόμοι ελέγχου. Έστω μεταβλητές κατάστασης:

$$\mathbf{z_1} = \mathbf{x}, \qquad \mathbf{z_2} = \dot{\mathbf{x}}, \qquad \mathbf{z_3} = \boldsymbol{\theta}, \qquad \mathbf{z_4} = \dot{\boldsymbol{\omega}}$$

Επομένως, οι εξισώσεις κατάστασης είναι οι εξής:

$$\dot{\mathbf{z}}_\mathbf{1} = \mathbf{z_2}$$

$$\dot{\mathbf{z}}_\mathbf{2} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R}{m} \mathbf{T}_{Tello} + \frac{1}{m} \mathbf{F}_d$$

$$\dot{\mathbf{z}}_\mathbf{3} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\varphi & c\theta s\varphi \\ 0 & -s\varphi & c\theta c\varphi \end{bmatrix} \mathbf{z_4}$$

$$\dot{\mathbf{z}}_\mathbf{4} = \begin{bmatrix} \frac{\tau_\varphi}{I_{xx}} \\ \frac{\tau_\theta}{I_{yy}} \\ \frac{\tau_\psi}{I_{zz}} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix}$$

Με το μοντέλο του Tello στον χώρο κατάστασης, είναι δυνατός ο έλεγχός του μέσω ανάδρασης κατάστασης (Full State Feedback), πράγμα που απαιτεί πλήρη εποπτεία όλων των μεταβλητών κατάστασης με κατάλληλους αισθητήρες. Πρέπει φυσικά να σημειωθεί πως το μοντέλο αυτό αποτελεί απλούστευση της πραγματικότητας, αφού βασίζεται πάνω σε παραδοχές ώστε να προκύψει γραμμικό. Στην πραγματικότητα όμως ένα quadcopter αποτελεί μη-γραμμικό σύστημα οπότε είναι προφανές πως θα υπάρχουν σφάλματα στις αποκρίσεις των καταστάσεων.

Λόγω των περιορισμών του Tello η χρήση ελεγκτών που αξιοποιεί το παραπάνω μοντέλο καθίσταται αρκετά δύσκολη. Κατά το πέρας της εργασίας, οι ελεγκτές οι οποίοι θα προγραμματιστούν είναι απλοί PID και κάνουν χρήση των έτοιμων εντολών των βιβλιοθηκών που αναφέρθηκαν στην Ενότητα 1.3 δίχως να υπάρχει εποπτεία του υποκείμενου συστήματος.

3. Τεχνολογίες Υλοποίησης Αλγοριθμικών Συστάδων

3.1. Localisation

Για την καλύτερη σύνθεση κινήσεων καθώς και τροχιών στο χώρο με την μέγιστη δυνατή ακρίβεια, τίθεται πλέον ανάγκη για προσαρμογή της ικανότητας του συστήματος να αναγνωρίζει την θέση του στο χώρο, μέσα σε ρεαλιστικά πλαίσια, μέσω ενός συστήματος αναφοράς. Η διαδικασία αυτή, γνωστή και ως Localisation είναι μια εμπλοκή σύνθετων λειτουργιών που επιτρέπει την δυνατότητα να γνωρίζει το ιπτάμενο ρομπότ την σχετική του θέση στο χώρο σε μορφή χρήσιμη για το σύστημα. Η δυνατότητα αυτή χαρακτηρίζει με την σειρά της μια διαφορετική προσέγγιση στην τοποθέτηση του μέσα στο περιβάλλον του και επιτρέπει μέγιστη επιρροή στην προσαρμογή του κατά την διάρκεια της κίνησης του μέσα στο περιβάλλον στο οποίο βρίσκεται.

Πλέον, με την χρήση ενός υποσυστήματος Localisation, το ρομπότ μπορεί να πάρει αποφάσεις εν γνώση των αλλαγών που αυτές θα θέσουν στο ίδιο ενώ η συνεχή γνώση, η οποία μπορεί να παρομοιαστεί με μία ροή πληροφοριών, δίνει χώρο για μια συνεχώς εξελικτική διαδικασία που καλυτερεύει σε μεγάλο βαθμό την ικανότητα του συστήματος να κινηθεί με μεγάλες ακρίβειες στο χώρο. Συγκεκριμένα για τα drones, με μια γενίκευση ίσως, ένα τέτοιο υποσύστημα βασίζεται σε κάθε μέθοδο ανάδρασης σε ερεθίσματα του περιβάλλοντος. Η χρήση αισθητήρων για την καλύτερη κατανόηση της θέσης είναι σαφής και δεδομένη. Αισθητήρες όπως απόστασης και κίνησης -IMU- μπορούν να χρησιμοποιηθούν μέσα από μια προεπεξεργασία ένωσης των δεδομένων κίνησης, γνωστό και ως Sensors Fusion, έτσι ώστε να αυξηθεί κατά πολύ η ακρίβεια της απόφασης ακριβής τοποθεσίας στον χώρο.

Για την περίπτωση του TELLO, η διαδικασία του να κατανοεί που βρίσκεται στον χώρο είναι ιδιαιτέρως σύνθετη, όπως είναι στην μεγαλύτερη γκάμα επιλογών Drones. Ο λόγος είναι κυρίως η πληθώρα των βαθμών ελευθερίας που έχουν ενώ παράλληλα μεγάλο ρόλο έχει και η έλλειψη άμεσης μεσολάβησης αυτόνομα για τον κάθε βαθμό ελευθερίας -Under-Actuated System-. Το TELLO συγκεκριμένα χρησιμοποιεί πολλαπλές μεθόδους για την αναπαράσταση της θέσης του στο χώρο με την κυριότερη να είναι η ασπρόμαυρη κάμερα που έχει στο κάτω μέρος του σώματος του. Παράλληλα με την χρήση της κάμερας αυτής, γίνεται χρήση του αισθητήρα Time of Flight -απόστασης- για την ακριβή μέτρηση του

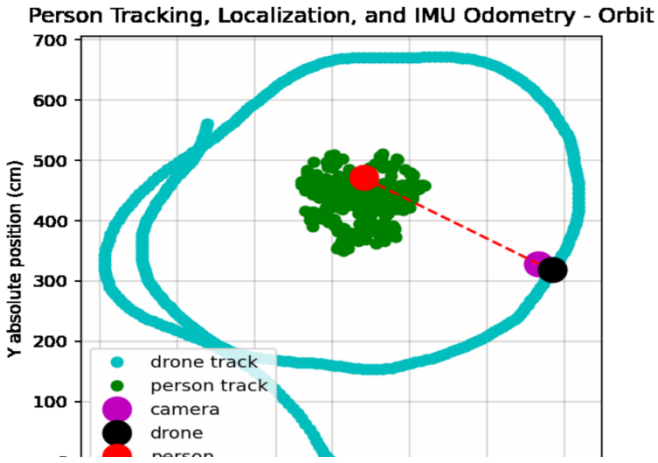
Εικόνα 1: Παράδειγμα Scew-off σε Localization με το TELLO. Η λευκή γραμμή είναι η πραγματική θέση του κατά την κίνηση ενώ η μπλέ η θέση που θεωρεί ότι έχει μέσω του συμπλέγματος όλων των αισθητήρων.

ύψους που βρίσκεται σε σχέση με την επιφάνεια που βρίσκεται κάθετα ως προς αυτό, ενώ για την αίσθηση της κίνησης του στο χώρο γίνεται η χρήση του IMU του που περιέχει 6 βαθμούς ελευθερίας κίνησης. Η ασπρόμαυρη κάμερα δημιουργεί την αίσθηση της τοποθεσίας μέσω της σύγκρισης της κάθε εικόνας που παράγεται για πιθανά κοινά σημεία με την ακριβώς προηγούμενη εικόνα που ήταν διαθέσιμη. Ο IMU με την σειρά του δίνει την δυνατότητα να οριστούν αποστάσεις μέσα από επιταχύνσεις -Accelerometer- και περιστροφές -Gyroscope-.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η πρακτική σύσταση ενός τέτοιου συστήματος τοποθέτησης στο χώρο. Παίρνοντας σαν παράδειγμα το TELLO, ενώ τόσες μέθοδοι είναι συνεχώς εν χρήση, η ακρίβεια του είναι πάρα πολύ μικρή. Ο κύριος λόγος είναι σαφώς ο θόρυβος που παρουσιάζεται στις μεθόδους ανίχνευσης θέσεις καθώς και στον αλγόριθμο για την εύρεση της ενώ παράλληλα υπάρχουν μέγιστα προβλήματα που ορίζουν σχεδόν αδύνατη την ακριβή τοποθέτηση του στον χώρο. Ένα εμφανές παράδειγμα είναι ότι από μόνο του ένα chip IMU 6 βαθμών ελευθερίας ανιχνεύει την περιστροφή της γης γύρω από τον εαυτό της -γνωστό και ως Drifting- θέτοντας έτσι συνεχή θόρυβο σε μορφή διανύσματος το οποίο δεν μπορεί να μοντελοποιηθεί με ακρίβεια και δεν γίνεται να φιλτραριστεί με ένα σύστημα 6 βαθμών ελευθερίας. Με την σειρά της και η κάμερα μπορεί να επιστρέψει εμφάνιση ομοιότητας σε δυο σημεία της εικόνας που δεν είναι τα ίδια αλλά μοιάζουν σε αποδεκτό βαθμό το οποίο θα επιφέρει έντονο θόρυβο στο σύστημα απόφασης της θέσης.

Στην υλοποίηση των θεμάτων του παρόντος πρότζεκτ το ζήτημα της έλλειψης ακρίβειας στον προσδιορισμό της θέσης γίνεται ακόμη πιο έντονο. Το TELLO δεν επιτρέπει τον έλεγχο και την παραμετροποίηση των built-in μεθόδων localisation ενώ αποτρέπει την χρήση έξυπνων αλγορίθμων επαναληπτικού χαρακτήρα με μικρό $\Delta\tau$ διακριτοποίησης λόγω της αρχιτεκτονικής σχεδίασης του API του που απαγορεύει κινήσεις μικρότερες των 20 εκατοστών. Το πρόβλημα γίνεται ακόμη πιο έντονο όταν τεθεί προσπάθεια για οριοθέτηση κινήσεων στον χώρο με τη δοσμένη βιβλιοθήκη εντολών μέσω της οποίας μπορεί κάποιος να θέσει ως μέσο αναφοράς την αρχική του θέση από την οποία σηκώθηκε ή κάποιο pad από τα δοσμένα της έκδοσης EDU. Δυστυχώς και οι δύο περιπτώσεις είναι άκρως ανακριβείς ενώ η συσσώρευση των πολλαπλών λαθών των αισθητήρων αυξάνεται εκθετικά με τον χρόνο πτήσης. Για τροχιές με πολλαπλά σημεία αναφοράς κατά την διάρκεια της, η επανατοποθέτηση τους στον χώρο γίνεται με τυφλές διαδικασίες με έντονη στοχαστικότητα με αποτέλεσμα το σύμπλεγμα όλων των κινήσεων να δείχνει πρόχειρο και νευρικό.

Έγινε εκτενής έρευνα για την επίλυση αυτών των προβλημάτων καθώς και διάφορες προσπάθειες να υλοποιηθούν κάποιες από αυτές. Με αισθητή διαφορά, η καλύτερη λύση ήταν η χρήση και της μπροστά κάμερας για τον προσδιορισμό της θέσης σύμφωνα με στοιχεία που αναγνώριζε ως σταθερές βάσης για την επιρροή τους στον προσδιορισμό αυτόν, όπως θα φανεί σε επόμενο σημείο της αναφοράς. Παράλληλα έγινε χρήση controllers υψηλού επιπέδου μέσω του ελέγχου ταχυτήτων σε τρεις άξονες που επιτρέπει να προσπεραστεί ο περιορισμός της αρχιτεκτονικής με έναν έμμεσο τρόπο, όπως θα φανεί σε επόμενο κεφάλαιο. Είναι πάντως επιθυμητό να υπάρχει πολύ καλός προσδιορισμός της θέσης ενός ιπτάμενου συστήματος για



Εικόνα 2: Αναφορά θέσης με την βοήθεια της κάμερας, Localization σε TELLO γραμμένο σε Python

την καλύτερη προσαρμογή του σε συνεχής κινήσεις βασισμένες σε τροχιές στον χώρο και ενώ έχουν εμφανιστεί διάφοροι τρόποι αντιμετώπισης στην βιβλιογραφίας, ακόμα παραμένει ένα σχετικά άλυτο πρόβλημα των συστημάτων αυτών με μεθοδολογίες όπως State Estimation και Φίλτρα στοχαστικότητας να έχουν το πάνω χέρι. Ενδιαφέρον θα παρουσίαζε δε η χρήση ενός συστήματος προσδιορισμού τρίτου προσώπου που ορίζει με μεγάλη ακρίβεια ιπτάμενα ρομπότ, όταν βρίσκονται εντός ενός συγκεκριμένου χώρου, άσχετα με την πληροφορία που αυτά παράγουν.

3.2. Perception – Recognition

Σε συνέχεια της προσπάθειας επίλυσης του προηγούμενου προβλήματος τίθεται το εξής ερώτημα: Πως μπορεί το σύστημα να θέση τον εαυτό του στο χώρο ενώ παράλληλα να μπορεί να αναγνωρίσει το περιβάλλον του και να κινηθεί σε τροχιές με πιθανά εμπόδια; Η ανάγκη για την αποφυγή εμποδίων καθώς και την ολοκλήρωση τροχιών και επίλυση προβλημάτων κίνησης, πάντα σε σχέση με τον περιβάλλον του, απαιτεί την κατανόηση του περίγυρου με τρόπο που είναι κατανοητό από το σύστημα ανίχνευσης του ρομπότ καθώς και χρήσιμο από το σύστημα αποφάσεων του. Η ικανότητα να ανιχνεύει η ακόμα και να αναγνωρίζει στοιχεία του περιβάλλοντος -γνωστό και ως Recognition and Perception- επιτρέπει την επίλυση προβλημάτων μέσω καλώς ορισμένων παραδοχών κίνησης σε σχέση με τα στοιχεία αυτά. Μάλιστα, μπορεί να τεθεί και ως επιχείρημα προόδου πως ιπτάμενα ρομπότ έχουν απόλυτη ανάγκη της ικανότητας αυτής για την χρήση τους σε πολύπλοκες διαδικασίες που είναι πραγματικά χρήσιμες καθώς και σημαντικές πολλές φορές για την επίλυση σημαντικών προβλημάτων της ανθρώπινης κοινωνίας. Η ικανότητα τους να αντιληφθούν το περιβάλλον τους με τρόπο χρήσιμο για τις διαδικασίες απόφασης και κίνησης, δίνει την δυνατότητα για on-the-fly προσαρμογές πορείας και τροχιάς, αποφάσεων με γνώμονα και κριτήρια όπως ενεργειακές απολαβές ή πορείες μικρότερης απόστασης και ούτω καθεξής.

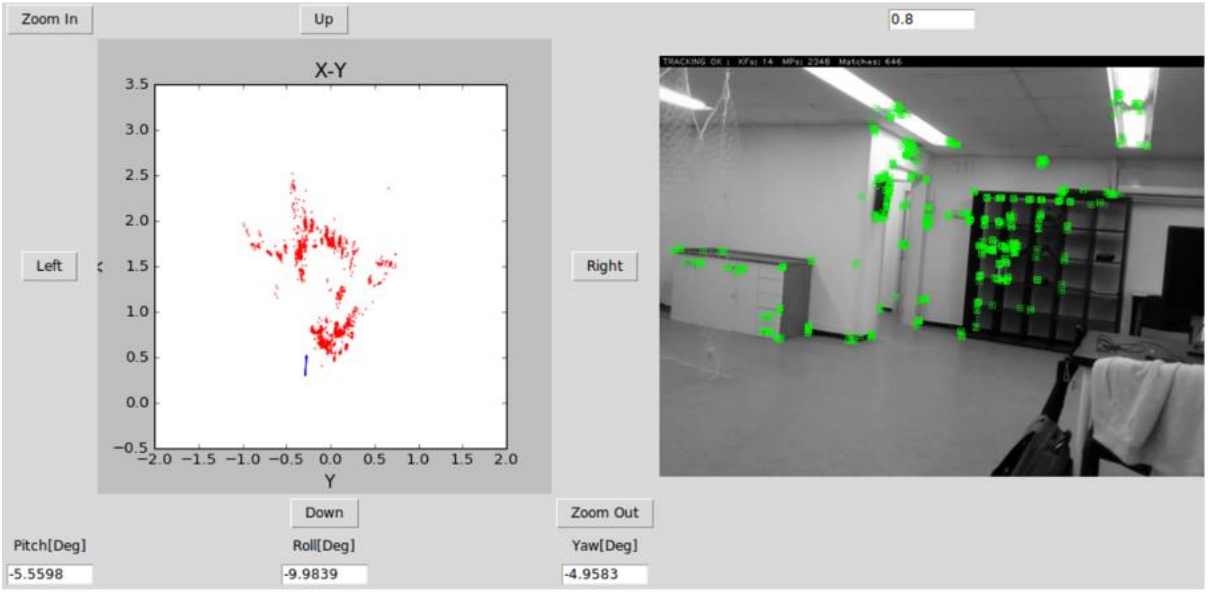
Για την ειδική περίπτωση όπου υπάρχει διαθέσιμο ένα σύμπλεγμα από αισθητήρες μεγάλης ταχύτητας, η προσέγγιση του τρόπου αναγνώρισης ορίζεται πλήρως από την τμηματοποίηση του χώρου γύρω του με τρόπο που κάθε αισθητήρας προσθέτει είτε νέα πληροφορία, είτε ενισχύει υπάρχουσα πληροφορία για την καλυτέρευση προσπάθεια ένωσης όλων των ροών πληροφορίας με σκοπό την σύνθεση ενός μοντέλου του χώρου. Με τον τρόπο αυτό μπορεί πολύ εύκολα να αποφύγει συγκεκριμένες καταστάσεις, εμπόδια καθώς και συγκρούσεις. Παρόλαυτα όμως, η χρήση μέσων με μεγαλύτερο through-put πληροφορίας γίνεται όλο και περισσότερο η προτιμητέα μέθοδος αναγνώρισης του περιβάλλοντος. Η χρήση κάμερας είτε κανονικής, αισθητήρων φωτός, είτε βάθους, είτε ακμών -πολύ καινούργια τεχνολογικά, πρώτη εφαρμογή το 2020-, επιτρέπει την επεξεργασία μεγάλης επιφάνειας χώρου με σχετικά γρήγορα αποτελέσματα και καλή ακρίβεια για την προσαρμογή τριχιάς είτε σχεδίου πορείας. Ο συνδυασμός και των δύο μεθόδων επιτρέπουν συστήματα που είναι environment-gnostic ενώ παράλληλα μπορούν να αντιδράσουν σε επικίνδυνες

Εικόνα 3: Παράδειγμα Recognition σε DJI TELLO.

καταστάσεις με εκπληκτικές ταχύτητες, κύριο παράδειγμα η αποφυγή σύγκρουσης μέσω αισθητήρα απόστασης ενώ η κάμερα δεν ήταν προσαρμοσμένη στην σωστή γωνία θέασης.

Στην περίπτωση του TELLO είναι προφανές ότι ο μόνος τρόπος αναγνώρισης του περιβάλλοντος μπορεί να γίνει μόνο μέσω κάμερας και αισθητήρα απόστασης για το ύψος. Δυστυχώς δεν υπάρχει πρόσβαση στην κάτω κάμερα με αποτέλεσμα να γίνεται χρήση μεθόδων που δεν μπορούν να αλλαχθούν. Παρόλαυτα με σωστή επεξεργασία, η μπροστά κάμερα επιτρέπει την αναγνώριση του περιβάλλοντος υπό συγκεκριμένη γωνία θέασης ενώ με μια συστάδα αλγορίθμων γίνεται να συμβεί σύνθεση του περιβάλλοντος με αρκετά καλή ακρίβεια, κάτι το οποίο όμως είναι πέρα από τις ανάγκες της συγκεκριμένης προσπάθειας.

Έγινε προσπάθεια να συντεθεί ένας βασικός κώδικας που να επιτρέπει την χρήση της μπροστινής κάμερας, όπως φαίνεται και σε επόμενο κεφάλαιο ενώ μέσω της βιβλιοθήκης OpenCV γίνεται αναγνώριση προσώπου για λόγους Localisation και Recognition. Μεγάλο πρόβλημα προέκυψε λόγω της αργής απόκρισης της κάμερας σε σχέση με το stream και το visualisation σε κάθε frame επιστροφής ενώ θα ήταν ιδιαιτέρως πιο σκόπιμο αν μπορούσε να γίνει επεξεργασία της εικόνας σε πολλαπλά επίπεδα ώστε να αποκτηθεί κάθε δυνατή χρήσιμη πληροφορία, κάτι που αφήνεται για μελλοντική υλοποίηση. Μεγάλο θέμα παρουσίασε η έλλειψη ενός βάσιμου στοιχείου αναγνώρισης καθώς και η πλήρως στοχαστική συμπεριφορά σε σχέση με τα Pads της μονάδας EDU τα οποία ενώ δεν υπάρχει τρόπος αλλαγής του τρόπου αναγνώρισης τους λόγω της αρχιτεκτονικής της TELLO, δεν έχουν καλό ιστορικό αναγνώρισης και τρόπου συμπεριφοράς σε σχέση με αυτά. Κάθε προσπάθεια για χρήση αυτών είχε δύστυχο αποτέλεσμα και σε κάποιες περιπτώσεις χειρότερα σε σύγκριση με την ελεύθερη πτήση, χωρίς τα pads.



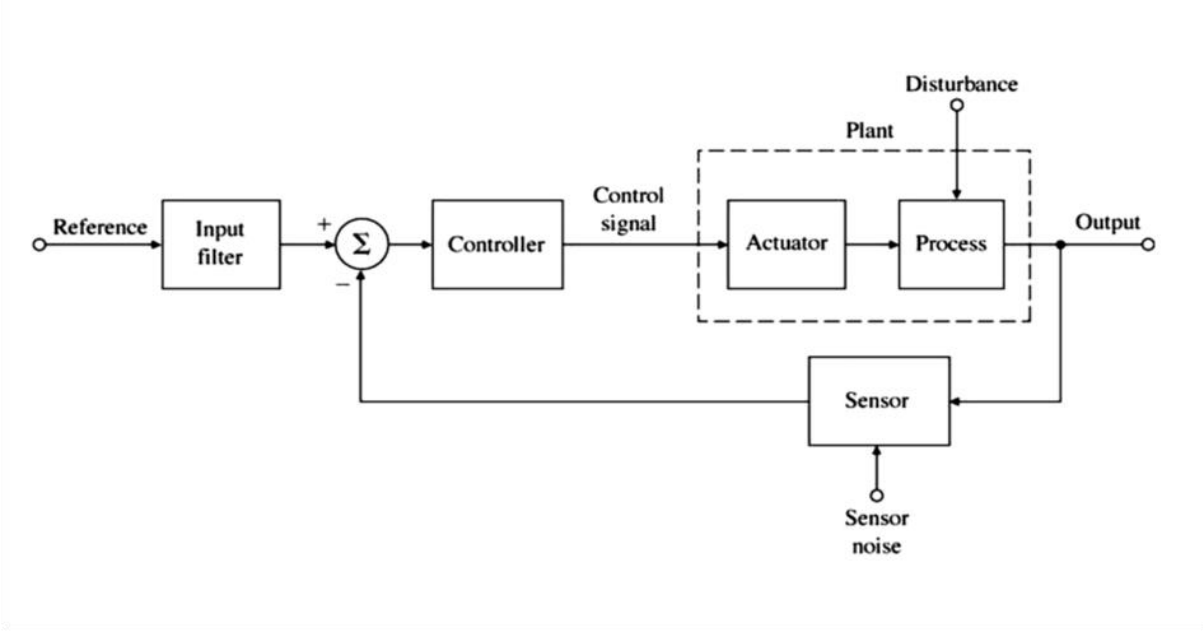
Εικόνα 4: Παράδειγμα Ανακατασκευής του χώρου μέσω της κάμερας με DJI TELLO.

3.3. Έλεγχος και Συγκλίσεις

Αναφέρεται με προφανή σκοπό, στην συγκεκριμένη εργασία, πως δεν χρειάζονται όλα τα προβλήματα λύσεις με αυτόνομο χαρακτήρα καθώς οι ιδιαιτέρως απλές τροχιές μπορούν με σχετική ευκολία να λυθούν μέσω ανοιχτών βρόγχων, αν και κάπως παράδοξα καθώς τέτοια λύση έχει έντονο στοχαστικό χαρακτήρα. Παρόλαυτα, σύνθετα θέματα δεν μπορούν να επιλυθούν μέσω αυτής της μεθόδου καθώς πολλά στοιχεία γίνονται μεταβλητά και η αντίστοιχη ανάδραση είναι απαραίτητη. Η πρόσβαση σε μία σύνθεση πληροφορίας που προσαρμόζεται με ανάδραση είναι από του σημαντικότερους τρόπους για την αυτοματοποίηση

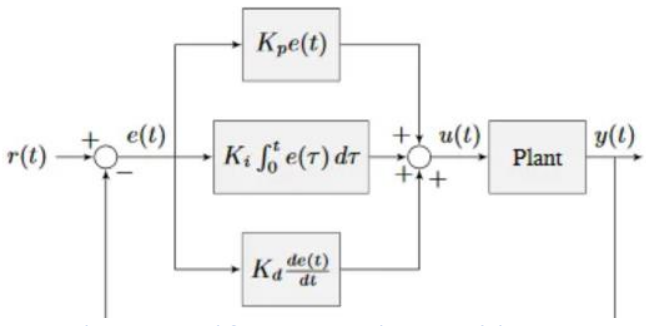
συστημάτων γενικής χρήσης ενώ επιτρέπει την αυτόνομη επιλογή αποφάσεων και στόχων σε σχέση με την συνεχή ροή πληροφορίας που παράγεται από τα αισθητήρια υποσυστήματα. Μέσω της σωστής χρήσης των ροών αυτών μπορεί κάποιος να θέσει υποσυστήματα σύγκλισης που να επιτρέπουν την συνεχή και καλά οριοθετημένη ανάδραση σε στοιχεία του περιβάλλοντος τα οποία εμφανίζονται τυχαία ή αλλάζουν την μορφή τους με την πάροδο του χρόνου.

Συστήματα ανάδρασης με κλειστούς βρόγχους μπορούν να θέσουν την επιθυμητή συμπεριφορά σε διάφορες καταστάσεις, σε ένα σύστημα που είναι environment agnostic ενώ παράλληλα μπορεί να λύσει προβλήματα όπου η συνεχή ροή δεδομένων, με σκοπό διορθώσεις και περιορισμό κινήσεων για λόγους εκτεταμένης προσοχής, να είναι η μόνη επιλογή. Η διαδικασία της ανάδρασης μπορεί να θέσει εφικτές καταστάσεις στις οποίες το σύστημα χρησιμοποεί την πληροφορίας του υποσυστήματος localisation και recognition με σκοπό την καλύτερη προσαρμογή του στο περιβάλλον του αλλά και για λόγους ακρίβειας και άμεσης απόκρισης. Εστιασμένα, μπορεί κάποιος να θεωρήσει ότι τέτοιου είδους συμπεριφορά είναι αποδεκτή σε κάθε πιθανή κατάσταση και για κάθε πιθανή χρήση, αν και η πολυπλοκότητα που προσθέτει είναι αισθητή. Παρόλαυτα, η χρήση σύγκλισης σφάλματος ή αλλιώς ελαχιστοποίηση σφαλμάτων μέσω ενός controller, είναι άκρως επιθυμητή για πολύπλοκες διαδικαςίες, όπως φαίνεται και στα επόμενα κεφάλαια.



Εικόνα 5: Απλό παράδειγμα γενίκευσης ενός συστήματος ελέγχου.

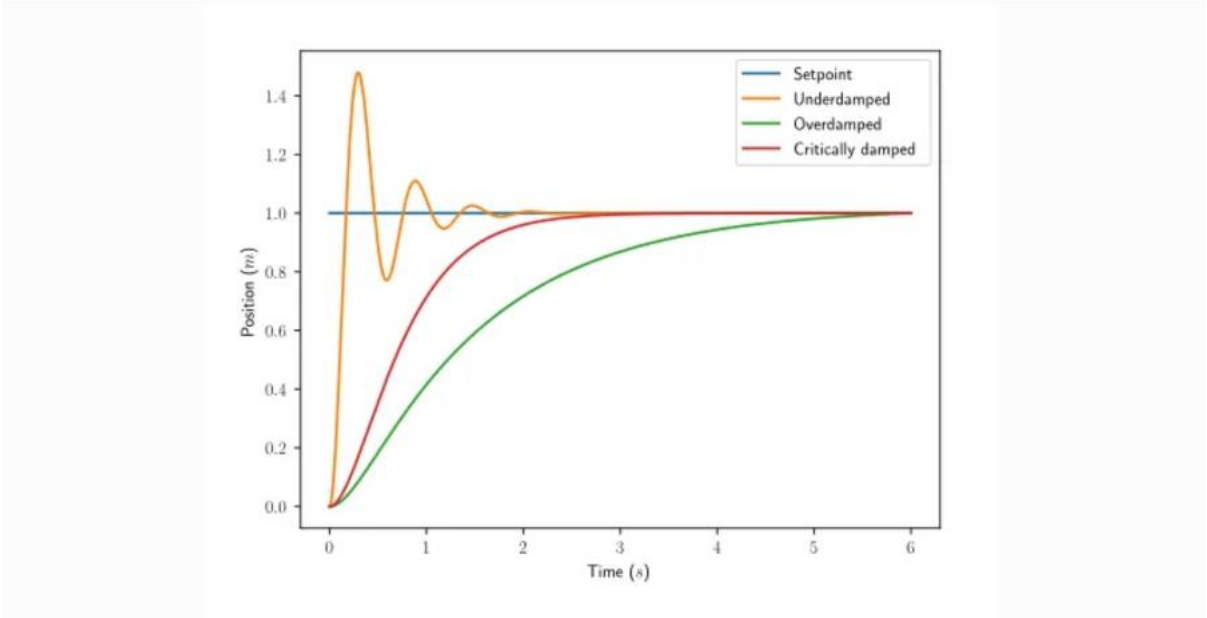
Στην περίπτωση του TELLO, δυστυχώς η χρήση τέτοιων μεθόδων είναι σχεδόν αδύνατη. Η έλλειψη πρόσβασης σε χαμηλού επιπέδου κώδικα συμπεριφοράς, θέτει αδύνατη την παραμετροποίηση της συμπεριφοράς με ανάδραση, πέρα από τις ήδη υπάρχουσες εργοστασιακές εκδοχές. Ακόμα χειρότερα, η χρήση υψηλού επιπέδου συστήματα ελέγχου είναι σχεδόν αδύνατη λόγω της αδυναμίας του TELLO να



Εικόνα 6: Παράδειγμα συστήματος ελέγχου PID.

δεχθεί κινήσεις μικρότερες από 20 εκατοστά, μέγεθος που θέτει σχεδόν άχρηστη την οποιαδήποτε ανάδραση σε κλειστό χώρο ή σε μεγάλες ταχύτητες. Υπάρχει ωστόσο η δυνατότητα να γίνει έλεγχος του drone μέσω εντολών που παρομοιάζουν τις εντολές ενός χειριστηρίου ελέγχου, κατά βάση εντολές που ελέγχουν την ταχύτητα σε 3 άξονες, με 6 βαθμούς ελευθερίας -πάνω κάτω, δεξιά αριστερά, μπροστά πίσω, περιστροφή δεξιά αριστερά- που μπορεί να επιφέρει ένα είδος ανάδρασης μέσω ταχυτήτων.

Συγκεκριμένα για την παρούσα εργασία, η προσέγγιση που ακολουθήθηκε ήταν η χρήση σφαλμάτων ως προς κάποιο σημείο αναφοράς για την σύγκλιση του drone σε μια συγκεκριμένη, μεταβλητή θέση με την χρήση των εντολών ταχύτητας RC. Θεωρώντας κάθε φορά κατάλληλο σύστημα ελέγχου, δημιουργήθηκαν 4 controllers για τα δυο από τα τρία σύνθετα θέματα που επιλέξαμε να επιλύσουμε. Στην μία περίπτωση, η χρήση της απόστασης καθώς και γωνίας από το αποτέλεσμα της επεξεργασίας και αναγνώρισης εικόνας χρησιμοποιείται ως σφάλματα ενώ στην άλλη, η δύναμη που ασκείται σταθερά από το βάρος του payload που έχει σηκώσει η συστάδα των drones. Σε κάθε περίπτωση γίνεται χρήση PD αρχιτεκτονικής ελέγχου για να ελαχιστοποιηθεί το αντίστοιχο σφάλμα. Για την καλύτερη λειτουργία των συστημάτων ελέγχου απαιτείται και εύρεση κερδών μέσω εμπειρικών διαδικασιών με γνώμονα την τάξη μεγέθους της ανάδρασης που απαιτείται σε κάθε περίπτωση. Η μοντελοποίηση του σφάλματος θεωρεί γραμμική συμπεριφορά ως προς την σύγκλιση και θεωρούμε ότι αλλάζει μόνο ο τρόπος που χειραγωγεί την σύγκλιση ο controller PD.



Εικόνα 7: Είδη πιθανών τρόπων σύγκλισης ενός PID Controller.

4. Θεματολογίες κώδικα

Έχοντας αναλύσει τα χαρακτηριστικά του DJI Tello drone το οποίο μελετάται, στην παρούσα ενότητα αναλύεται ο κώδικας που χρησιμοποιήθηκε για την υλοποίηση των λειτουργιών που αναφέρθηκαν κατά την Εισαγωγή.

Οι κώδικες και των τριών θεματολογιών έχουν αναρτηθεί στο ακόλουθο GitHub repository:

GitHub repo link: https://github.com/itsikelis/tello-robotics-project

4.1. Περιορισμοί

Πριν την ανάλυση του κώδικα της κάθε θεματολογίας, είναι σκόπιμο να αναφερθούν μερικοί περιορισμοί που παρουσιάζει το Tello τόσο από άποψη λογισμικού, όσο και από άποψη λειτουργίας.

- Οι κάμερες καθώς και το ζευγάρι IR αισθητήρων που χρησιμοποιεί το Tello για localization είναι εξαιρετικά επιρρεπείς σε μεταβολές φωτεινότητας στον χώρο εργασίας. Αυτό έχει ως αποτέλεσμα λειτουργίες οι οποίες δουλεύουν επαρκώς σε ένα σημείο ενός δωματίου να αποκλίνουν από τα επιθυμητά αποτελέσματα σε ένα διαφορετικό σημείο, λόγω διαφοράς στον φωτισμό.

- Το IMU του Tello συχνά αποπροσανατολίζεται, είτε λόγω περιβαλλοντικών συνθηκών, είτε λόγω ανάγκης βαθμονόμησης. Αυτό έχει ως αποτέλεσμα οι αποκρίσεις του Tello κατά την επαναληπτική διεξαγωγή της ίδιας λειτουργίας και υπό τις ίδιες συνθήκες να αποκλίνουν σημαντικά μεταξύ τους, καθιστώντας την συμπεριφορά του Tello στοχαστική.

- Το Tello με τις βασικές του εντολές μπορεί να διανύσει καμπυλόγραμμες τροχιές μονάχα κυκλικής γεωμετρίας, αφού πάντα τις υπολογίζει με την βοήθεια μιας ακτίνας και δύο σημείων. Εάν ο χρήστης επιθυμεί την υλοποίηση τροχών με περίπλοκες γεωμετρίες οι οποίες δεν μπορούν να εκφραστούν ως σύμπλεγμα ημικυκλίων (πχ. πολυωνυμικές τροχιές) τότε θα πρέπει να δημιουργήσει έναν ξεχωριστό ελεγκτή για αυτόν τον σκοπό.

- Το εσωτερικό λογισμικό του Tello επιτρέπει μόνο μετατοπίσεις μεγαλύτερες των 20 cm, και κυκλικές τροχιές ακτίνας μεγαλύτερης των 50 cm. Οι περιορισμοί αυτοί καθιστούν την δημιουργία ελεγκτών αρκετά πιο σύνθετη, διότι δεν γίνεται να χρησιμοποιηθούν οι βασικές εντολές μετακίνησης του Tello για μικρά τμήματα. Ο χρήστης αναγκαστικά περιορίζεται στην χρήση μεθόδων άμεσου ελέγχου ταχύτητας.

4.2. Υλοποίηση τροχιών

Η παρούσα θεματολογία αφορά τον προγραμματισμό του Tello με σκοπό την διάνυση μιας πληθώρας τροχιών με αυξανόμενη κλίμακα δυσκολίας. Οι τροχιές οι οποίες επιλέχθηκαν προς υλοποίηση αναγράφονται παρακάτω:

- Ευθύγραμμη οδήγηση σε ένα σημείο Α και επιστροφή στην βάση απογείωσης.

- Οδήγηση σε τετραγωνική τροχιά.
- Οδήγηση σε κυκλική τροχιά.
- Οδήγηση σε τροχιά σχήματος οχτώ.

Ο φάκελος στον οποίο είναι αποθηκευμένοι οι κώδικες των τροχιών στο GitHub repository ονομάζεται Basic_Splines. Προτού αναλυθεί η υλοποίηση των τροχιών, είναι σκόπιμη η επεξήγηση των σημαντικότερων εντολών που χρησιμοποιήθηκαν από την βιβλιοθήκη DJITelloPy.

Εντολή:	*Περιγραφή λειτουργίας:*
Tello.connect()	Σύνδεση στην διεύθυνση του Tello από τον υπολογιστή.
Tello.enable_mission_pads()	Ενεργοποίηση βάσεων του Tello προς αναγνώριση.
Tello.takeoff(), Tello.land()	Απογείωση, προσγείωση.
Tello.go_xyz_speed((x, y, z), speed)	Γραμμική τροχιά που ξεκινά από την αρχική τοποθεσία του Tello (σημείο αναφοράς) προς το σημείο (x, y, z) με ταχύτητα speed.
Tello.go_xyz_speed_mid((x, y, z), speed, mid)	Γραμμική τροχιά με αρχική τοποθεσία την βάση αναφοράς με ταυτότητα mid. Π.χ η βάση με αριθμό "1" θα έχει ταυτότητα mid = 1.
Tello.curve_xyz_speed((x1,y1,z1), (x2,y2,z2), speed)	Κυκλική τροχιά που ξεκινά από την αρχική τοποθεσία του Tello και διαπερνά τα σημεία (x1,y1,z1) και (x2,y2,z2) με ταχύτητα speed.
Tello.curve_xyz_speed_mid((x1,y1,z1), (x2,y2,z2), speed, mid)	Κυκλική τροχιά με αρχική τοποθεσία την βάση αναφοράς με ταυτότητα mid.

Σημειώνεται πως όλες οι εντολές μετρούν μεταφορές σε cm και γωνίες σε μοίρες.

4.2.1. . Οδήγηση σε σημείο Α και επιστροφή στην βάση

Έστω σημείο Α με συντεταγμένες (40, 40, 30) cm από την βάση αναφοράς mid = 1. Ο κώδικας της τροχιάς από την βάση 1 στο σημείο Α και πίσω παρουσιάζεται παρακάτω:

```
from time import sleep
from djitellopy import Tello
from dataclasses import dataclass

@dataclass      # Point(x,y,z) in 3D space
class Point:
    x: float
    y: float
    z: float

tello = Tello()
tello.connect()
tello.enable_mission_pads()
tello.takeoff()

A = Point(40, 40, 30)

# Move to point A with mission pad 1 as reference.
tello.go_xyz_speed_mid(A.x, A.y, A.z, 50, 1)

# Return to mission pad 1 with A as reference.
tello.go_xyz_speed(-A.x, -A.y, 0, 30)

# Mission pad 1 identification loop. When the drone recognizes mission pad 1
# (20 tries total) then it moves over it and then proceeds to land.
if(tello.get_mission_pad_id() == 1):
    i = 0
    while i < 20:
        tello.go_xyz_speed_mid(0, 0 , 30, 30, 1)
        tello.land()
        i += 1
```

Παρατηρήσεις:

Ο λόγος που η εύρεση της βάσης 1 επαναλαμβάνεται εντός βρόχου while είναι διότι η διαδικασία εντοπισμού βάσεων αποδείχτηκε δύσκολη και πολλές φορές απρόβλεπτη. Ο εντοπισμός βάσεων πραγματοποιείται μέσω είτε της μπροστινής, είτε της κατακόρυφης κάμερας, οι οποίες παρουσιάζουν σημαντική ευαισθησία στις συνθήκες φωτεινότητας του χώρου όπου διεξάγεται το πείραμα.

Για αυτόν τον λόγο πολλές φορές καθίσταται προτιμότερη η χρήση ενός σημείου στον χώρο σαν σημείο αναφοράς, έναντι μιας βάσης.

4.2.2. . Οδήγηση σε τετραγωνική τροχιά

Έστω τετράγωνο ακμής 50 cm που αποτελεί την ζητούμενη τετραγωνική τροχιά. Η τροχιά αυτή μπορεί να αναλυθεί σε τέσσερις απλούστερες ευθύγραμμες τροχιές μετά από κατάλληλη ορθή περιστροφή. Ο κώδικας που την υλοποιεί παρουσιάζεται παρακάτω:

```
from time import sleep
```

```python
from djitellopy import Tello

tello = Tello()
tello.connect()
tello.takeoff()

# Square path execution loop.
n = 1   # Number of square paths to be executed.
for i in range(n):
    # The drone moves forward and then rotates 90 degrees clockwise.
    # By repeating this action 4 times, a square path is formed.
    for j in range(4):
        tello.move_forward(50)
        tello.rotate_counter_clockwise(90)

tello.land()
```

Παρατηρήσεις:

Για την τετραγωνική τροχιά επιλέχτηκε μια επανάληψη ευθύγραμμης μεταφοράς του Tello κατά την ακμή του τετραγώνου (50 cm) ακολουθούμενη από ωρολογιακή περιστροφή κατά 90ο. Παρατηρείται πως η περιστροφή του Tello δημιουργεί μικρές αποκλίσεις από την τετραγωνική τροχιά, οι οποίες συσσωρεύονται, με αποτέλεσμα η τοποθεσία απογείωσης να διαφέρει σε ένα βαθμό από την τοποθεσία προσγείωσης.

Δοκιμάζοντας την εκτέλεση τετραγωνικής τροχιάς δίχως περιστροφές, οι αποκλίσεις αυτές εμφανίζονται σε μικρότερο βαθμό. Αυτό σημαίνει πως όσο μεγαλύτερος είναι ο αριθμός των προγραμματισμένων βασικών κινήσεων από τις οποίες κατασκευάζεται μια τροχιά, τόσο μεγαλύτερη θα είναι η απόκλιση από την αυτήν την τροχιά.

4.2.3. . Οδήγηση σε κυκλική τροχιά

Έστω κύκλος ακτίνας 50 cm που αποτελεί την ζητούμενη κυκλική τροχιά. Η τροχιά αυτή μπορεί να αναλυθεί σε δύο απλούστερες ημικυκλικές τροχιές επιλέγοντας τα κατάλληλα ενδιάμεσα και τελικά σημεία. Ο κώδικας που την υλοποιεί παρουσιάζεται παρακάτω:

```python
from djitellopy import Tello
from dataclasses import dataclass

@dataclass      # Point(x,y,z) in 3D space
class Point:
    x: float
    y: float
    z: float

tello = Tello()
tello.connect()
tello.takeoff()

## Define points
#  Points of Semi-Circle 1
```

```
A1 = Point(0, 0, 20)
B1 = Point(50, 50, 0)
C1 = Point(100, 0, 0)

#  Points of Semi-Circle 2
A2 = Point(0, 0, 0)
B2 = Point(-50, -50, 0)
C2 = Point(-100, 0, 0)

tello.go_xyz_speed(A1.x, A1.y, A1.z, 40)   # Initialize height
tello.curve_xyz_speed(B1.x, B1.y, B1.z, C1.x, C1.y, C1.z, 40)    # Semi-Circle
1
tello.curve_xyz_speed(B2.x, B2.y, B2.z, C2.x, C2.y, C2.z, 40)    # Semi-Circle
2

tello.land()
```

Παρατηρήσεις:

Είναι σημαντικό να αναφερθεί ότι το αδρανειακό σύστημα συντεταγμένων που αντιλαμβάνεται το Tello αλλάζει από τροχιά σε τροχιά. Αυτό συμβαίνει γιατί κάθε τροχιά αξιοποιεί το δικό της σημείο αναφοράς, είτε αυτό είναι το σημείο εκκίνησής της, είτε είναι η βάση αναφοράς. Στην περίπτωση αξιοποίησης του σημείου εκκίνησης για αναφορά, το Tello αποτυπώνει το τοπικό του σύστημα συντεταγμένων στο σημείο εκκίνησης και το θεωρεί ως αδρανειακό σύστημα συντεταγμένων μέχρι η τροχιά να σταματήσει. Το τελικό σημείο της τροχιάς γίνεται το αρχικό της επόμενης, όπου και αποτυπώνεται εκ νέου ο προσανατολισμός του Tello για να δημιουργηθεί το νέο αδρανειακό σύστημα αναφοράς για αυτήν την τροχιά. Για τον λόγο αυτό απαιτείται προσοχή στον ορισμό των σημείων των επόμενων διαδοχικών τροχιών.

4.2.4. . Οδήγηση σε τροχιά σχήματος οχτώ

Η τροχιά σχήματος οχτώ μπορεί να θεωρηθεί ότι αποτελείται από δύο εφαπτόμενους κύκλους, έστω και πάλι ακτίνας 50 cm. Βάσει της υλοποίησης της κυκλικής τροχιάς της Ενότητας 3.1.3, με παρόμοια λογική πραγματοποιείται και η τροχιά σχήματος οχτώ, η οποία αποτελείται από τέσσερις απλούστερες ημικυκλικές τροχιές. Ο κώδικας που την υλοποιεί παρουσιάζεται παρακάτω:

```
## Define points
#  Points of Semi-Circle 1
A1 = Point(0, 0, 30)
B1 = Point(50, 50, 0)
C1 = Point(100, 0, 0)

#  Points of Semi-Circle 2
A2 = Point(0, 0, 0)
B2 = Point(50, -50, 0)
C2 = Point(100, 0, 0)

#  Points of Semi-Circle 3
A3 = Point(0, 0, 0)
B3 = Point(-50, 50, 0)
C3 = Point(-100, 0, 0)
```

```
#  Points of Semi-Circle 4
A4 = Point(0, 0, 0)
B4 = Point(-50, -50, 0)
C4 = Point(-100, 0, 0)

## Trajectories
tello.go_xyz_speed(A1.x, A1.y, A1.z, 40)   # Initialize height
tello.curve_xyz_speed(B1.x, B1.y, B1.z, C1.x, C1.y, C1.z, 40)    # Semi-Circle
1
tello.curve_xyz_speed(B2.x, B2.y, B2.z, C2.x, C2.y, C2.z, 40)    # Semi-Circle
2
tello.curve_xyz_speed(B3.x, B3.y, B3.z, C3.x, C3.y, C3.z, 40)    # Semi-Circle
3
tello.curve_xyz_speed(B4.x, B4.y, B4.z, C4.x, C4.y, C4.z, 40)    # Semi-Circle
4

tello.land()
```

Παρατηρήσεις:
Κατά τις δοκιμές των τροχιών γίνεται αντιληπτό πως, σε σύγκριση με την κυκλική τροχιά, η τροχιά σχήματος οχτώ παρουσιάζει πολύ μεγαλύτερη απόκλιση από την ιδανική. Το σημείο απογείωσης διαφέρει σε μεγάλο βαθμό από το σημείο προσγείωσης και αυτό οφείλεται στον μεγάλο αριθμό κυκλικών κινήσεων οι οποίες πρέπει να πραγματοποιηθούν. Η συνεχής αλλαγή συστήματος συντεταγμένων, η ευαισθησία της κυκλικής τροχιάς σε σχέση με την ευθύγραμμη καθώς και η μεγάλη έκταση της τροχιάς (ο χώρος του εργαστηρίου είναι μη-ομοιόμορφος ως προς την φωτεινότητα) αυξάνουν δραματικά την πιθανότητα σφάλματος στην διάνυση της τροχιάς.

4.3. Εντοπισμός και παρακολούθηση προσώπων (face tracking)

Η θεματολογία αυτή αφορά τον προγραμματισμό υψηλού επιπέδου ελεγκτή για την συνεχή παρακολούθηση ενός προσώπου από το Tello.

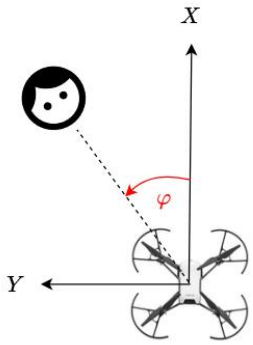
Το σύστημα αυτομάτου ελέγχου το οποίο κατασκευάζεται ελέγχει ουσιαστικά την περιστροφή του Tello γύρω από τον κατακόρυφο άξονα Z, δηλαδή την εκτροπή (yaw) ώστε αυτό να κοιτά μετωπικά το πρόσωπο του ατόμου το οποίο εντοπίζει. Ο ελεγκτής ο οποίος χρησιμοποιείται για τον έλεγχο της εκτροπής (yaw controller) επιλέγεται του τύπου PID.

Πρωταγωνιστικό ρόλο στον σχεδιασμό ενός τέτοιου ελεγκτή παίζει η βιβλιοθήκη Υπολογιστικής Οράσεως OpenCV. Μέσω των εντολών που αυτή προσφέρει πραγματοποιείται συνεχής επεξεργασία κάθε καρέ (frame) που λαμβάνει η μπροστινή κάμερα του Tello ώστε να εντοπίζονται πρόσωπα εντός του οπτικού της πεδίου. Το GitHub της OpenCV, στο οποίο περιέχονται όλα τα πρότυπα αρχεία για την αναγνώριση προσώπων, αλλά και άλλων μελών του σώματος αναγράφεται παρακάτω:

OpenCV GitHub repo link: https://github.com/opencv/opencv

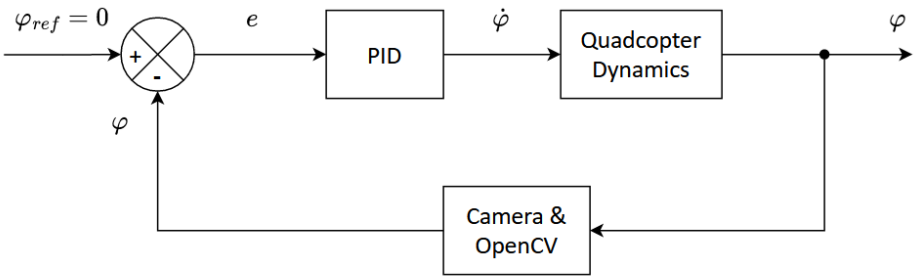
4.3.1. Προκαταρκτικός σχεδιασμός νόμου ελέγχου

Η αρχή λειτουργίας του συστήματος παρακολούθησης προσώπων περιγράφεται ως εξής:



Σχήμα 3: Σχηματική αναπαράσταση παρακολούθησης προσώπου

Σκοπός του συστήματος είναι η μετωπική ευθυγράμμιση του Tello με το εντοπιζόμενο πρόσωπο, πράγμα που σημαίνει ότι για δεδομένη μετατόπιση του προσώπου στο χώρο, το Tello πρέπει να είναι σε θέση να περιστραφεί ως προς τον άξονα Z κατά μια αντίστοιχη γωνία εκτροπής φ. Για ευθυγράμμιση πρέπει λοιπόν η εκτροπή αναφοράς να ισούται με το 0 ($\varphi_{ref} = 0$). Το απλοποιημένο σύστημα κλειστού βρόχου παρουσιάζεται στο Σχήμα 4:



Σχήμα 4: Σύστημα κλειστού βρόχου για τον έλεγχο της γωνίας εκτροπής

Το Tello μέσω της κάμερας μπορεί με έμμεσο τρόπο να μετράει την γωνία εκτροπής σε σχέση με το εντοπιζόμενο πρόσωπο. Η μετρούμενη γωνία εκτροπής συγκρίνεται με την εκτροπή αναφοράς, και το σφάλμα τους εισάγεται στον ελεγκτή PID, ο οποίος ελέγχει τον ρυθμό μεταβολής της γωνίας εκτροπής. Η ταχύτητα εκτροπής οδηγεί σε μια νέα γωνία εκτροπής μέσω ολοκλήρωσης κατά την δυναμική του συστήματος όπως αυτή αναλύθηκε στην Ενότητα 2. Έτσι ο βρόχος κλείνει και η ανάδραση συνεχίζεται.

4.3.2. Οργάνωση κώδικα

Ο κώδικας υλοποίησης του παραπάνω συστήματος ελέγχου χωρίζεται για λόγους σαφήνειας σε δύο αρχεία. Το πρώτο αρχείο με τίτλο functions.py είναι μη-εκτελέσιμο και αποτελεί το πεδίο συγγραφής όλων των απαραίτητων συναρτήσεων για τον εντοπισμό προσώπου και τον έλεγχο της γωνίας εκτροπής. Το δεύτερο αρχείο ονομάζεται facetrack.py, είναι εκτελέσιμο και αξιοποιεί τις συναρτήσεις του πρώτου για την σύνθεση του συστήματος κλειστού βρόχου που αναλύθηκε στην Ενότητα 3.2.1. Και τα δύο περιέχονται στον φάκελο Face_Tracking του repository στο GitHub.

4.3.3. Συναρτήσεις εντοπισμού και παρακολούθησης (facetrack.py)

Στην παρούσα υπο-ενότητα περιγράφονται μεμονωμένα οι συναρτήσεις που υλοποιούν τόσο τον εντοπισμό των προσώπων όσο και την συνεχή παρακολούθησή τους από το Tello μέσω ελεγκτή.

Η συνάρτηση initializeTello() είναι υπεύθυνη για την σύνδεση με το Tello, την αρχικοποίηση των μεταβλητών κατάστασης και του ανοίγματος της κάμερας.

```
from djitellopy import Tello
import cv2
import numpy as np

def initializeTello():
    tello = Tello()
    tello.connect()
    tello.for_back_velocity = 0
    tello.left_right_velocity = 0
    tello.up_down_velocity = 0
    tello.yaw_velocity = 0
    tello.speedYaw = 0
    tello.streamon()    # Begin front camera streaming
    return tello
```

Η συνάρτηση telloGetFrame() λαμβάνει τα καρέ του βίντεο που καταγράφει η κάμερα και επιπλέον τους μεταβάλει τις διαστάσεις όπως ορίζει ο χρήστης.

```
def telloGetFrame(tello, w, h):
    telloFrame = tello.get_frame_read()
    telloFrame = telloFrame.frame
    img = cv2.resize(telloFrame, (w, h))
    return img
```

Η συνάρτηση findFace() είναι υπεύθυνη για τον εντοπισμό προσώπων στα καρέ που λαμβάνει ως όρισμα από την κάμερα. Αξιοποιεί την κλάση της OpenCV που ονομάζεται CascadeClassifier, η οποία λαμβάνει ως όρισμα ένα πρότυπο αρχείο .xml που αντιστοιχεί στην πληροφορία ενός προσώπου (προκύπτει μέσω μηχανικής μάθησης όμως για του σκοπούς της εργασίας λαμβάνεται έτοιμο). Το αρχείο αυτό ονομάζεται haarcascade_frontalface_default.xml και παρέχεται στο GitHub της OpenCV για τον σκοπό του face detection.

Αφού το καρέ του βίντεο μετατραπεί ως προς το χρώμα του σε ασπρόμαυρο μέσω της εντολής cvtColor(img, cv2.COLOR_BGR2GRAY), εισάγεται στην συνάρτηση detectMultiScale(), η οποία εντοπίζει πρόσωπα διαφόρων μεγεθών στο ασπρόμαυρο καρέ και τα επιστρέφει ως λίστες ορθογωνίων περιοχών.

Τα εντοπιζόμενα πρόσωπα ύστερα περιβάλλονται από ορθογώνια με σκοπό να ποσοτικοποιηθεί η τοποθεσία τους σε σχέση με το Tello. Ανάλογα με το πόσο μικρή ή μεγάλη είναι η επιφάνεια του περιβάλλοντος ορθογωνίου ορίζεται το πόσο κοντά ή μακριά βρίσκεται το πρόσωπο από την κάμερα. Επιπλέον οι οριζόντιες και κατακόρυφες συντεταγμένες του κέντρου του ορθογωνίου αποτελούν προσέγγιση της τοποθεσίας του προσώπου εντός των ορίων του καρέ. Έτσι λαμβάνονται συνεχώς μετρήσεις των επιφανειών και συντεταγμένων των ορθογωνίων που περιβάλλουν τα εκάστοτε πρόσωπα στο καρέ, και επιλέγονται το ορθογώνιο με την μέγιστη επιφάνεια οι αντίστοιχες συντεταγμένες του.

```
def findFace(img):
    faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    scaleFactor = 1.2
    minNeighbors = 8
    faces = faceCascade.detectMultiScale(imgGray, scaleFactor, minNeighbors)
```

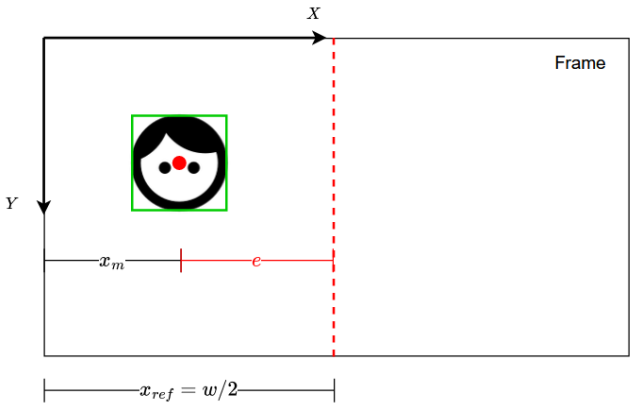
```
    faceListCenter = []         # Initialize face center coordinate list
    faceListArea = []           # Initialize face rectangle area list

    for (x, y, w, h) in faces:
      # Rectangle around face
        cv2.rectangle(img, (x, y), (x+w, y+h), (0,0,255), 2)
        cx = x + w // 2          # Center coordinates must be integers
        cy = y + h // 2
        area = w * h             # Frame area
      # Circle at center of rectangle
        cv2.circle(img, (cx, cy), 5, (0,255,0), cv2.FILLED)
        faceListArea.append(area)
        faceListCenter.append([cx, cy])

    if len(faceListArea) != 0:
        i = faceListArea.index(max(faceListArea))
        return img, [faceListCenter[i], faceListArea[i]]
    else:
        return img, [[0, 0], 0]
```

Τέλος, η συνάρτηση trackFace() είναι υπεύθυνη για τον έλεγχο του συστήματος, δηλαδή την συνεχή παρακολούθηση του προσώπου όταν αυτό μετακινείται ανά καρέ.

Για τον έλεγχο της εκτροπής επιλέγεται ελεγκτής PD. Η αναλογική συνιστώσα P συμβάλει στην γρήγορη απόκριση του συστήματος και η διαφορική συνιστώσα D καθιστά το σύστημα ευσταθές. Ο λόγος που επιλέχτηκε να αμεληθεί η ολοκληρωτική I συνιστώσα του ελεγκτή, είναι διότι ένα quadcopter αποτελεί ισχυρά ασταθές σύστημα, ενώ ο ολοκληρωτικός έλεγχος οδηγεί σε κινδύνους αστάθειας και αργής απόκρισης.



Σχήμα 5: Ενδεικτικό σκαρίφημα λειτουργίας facetracking και χαρακτηριστικά μεγέθη

Ο ελεγκτής της εκτροπής λαμβάνει ως είσοδο το σφάλμα μεταξύ της τετμημένης αναφοράς του κέντρου του καρέ, και της αντίστοιχης μετρούμενης τετμημένης του κέντρου του ορθογωνίου που περιβάλλει το εντοπιζόμενο πρόσωπο. Ανάλογα με το πρόσημο του σφάλματος (δηλαδή αν το πρόσωπο βρίσκεται στα αριστερά ή στα δεξιά του άξονα συμμετρίας του καρέ) ο ελεγκτής δίνει μια ταχύτητα εκτροπής $\dot{\varphi}$ ωρολογιακά ή ανθωρολογιακά, με μέγιστη τιμή κατά μέτρο τις 80o/sec.

```
def trackFace(tello, info, w, pidYaw, pErrorYaw):
    x_m, y_m = info[0]          # measured x, y coordinates
    # Yaw angle PID
```

```
    errorYaw = x_m - w // 2         # deviation between center value and
measurement
    # PD Conroller Implementation
    speedYaw = pidYaw[0] * errorYaw + pidYaw[2] * (errorYaw - pErrorYaw)
    speedYaw = int(np.clip(speedYaw, -80, 80))     # yaw speed saturation
control

    if x_m != 0:                                # yaw != 0 -> spin cw/ccw
        tello.yaw_velocity = speedYaw
    else:                                       # yaw = 0 -> no spin
        tello.for_back_velocity = 0
        tello.left_right_velocity = 0
        tello.up_down_velocity = 0
        tello.yaw_velocity = 0
        errorYaw = 0                    # yaw error refresh

    if tello.send_rc_control:
        tello.send_rc_control(tello.left_right_velocity,
                              tello.for_back_velocity,
                              tello.up_down_velocity,
                              tello.yaw_velocity)
    return errorYaw
```

4.3.4. Εκτέλεση κλειστού βρόχου (facetrack.py)

Εφόσον έχουν οριστεί όλες οι συναρτήσεις που έχουν να κάνουν με τον εντοπισμό και την παρακολούθηση προσώπων ανά καρέ, δεν μένει μόνο παρά να χρησιμοποιηθούν για την δόμηση του κλειστού βρόχου.

Αρχικά ορίζονται οι διαστάσεις των καρέ, τα κέρδη k_P και k_D του ελεγκτή PD (προφανώς δίχως ακριβές μοντέλο επιλέγονται μέσω trial and error) και αρχικοποιούνται οι μεταβλητές κατάστασης του Tello καθώς και το σφάλμα εκτροπής.

```
from functions import *
import cv2

w, h = 640, 480             # Frame width, height

pidYaw = [0.19, 0, 0.60]      # Yaw PID controller gains kP, kI, kD

pErrorYaw = 0               # Initialize yaw error
startCounter = 0            # 0: flight, 1: no flight (for camera testing
purposes)

tello = initializeTello()
```

Ο κλειστός βρόχος του συστήματος αυτομάτου ελέγχου της εκτροπής ορίζεται πλέον εύκολα μέσω των συναρτήσεων του αρχείου functions.py.

```
while True:     # Control Loop
    # Flight
    if startCounter == 0:
```

```
        tello.takeoff()
        tello.go_xyz_speed(0, 0, 100, 40)
        startCounter = 1

    img = telloGetFrame(tello, w, h)

    img, info = findFace(img)

    pErrorYaw = trackFace(tello, info, w, pidYaw, pErrorYaw)

    cv2.imshow('Image', img)
    if cv2.waitKey(1) & 0xFF == ord('q'):   # End loop when 'q' key is pressed
        tello.land()
        break
```

Παρατηρήσεις:

Η απόκριση του Tello στην ακολουθία του εντοπιζόμενου προσώπου αν και ακριβής, παρουσιάζει περιστασιακές υπερακοντίσεις, οι οποίες οφείλονται τόσο σε θέματα ευαισθησίας σε φωτισμό, όσο και στο ότι τα κέρδη του ελεγκτή PD δεν αποτελούν βέλτιστα.

Σημείωση:

Εκτός από τον ελεγκτή ο οποίος ελέγχει την γωνία εκτροπής για την παρακολούθηση προσώπου, πραγματοποιήθηκε επίσης απόπειρα δημιουργίας ελεγκτή απόστασης από το εντοπιζόμενο πρόσωπο.

Η απόσταση μετριέται με έμμεσο τρόπο, μέσω του εμβαδού του ορθογωνίου που περιβάλλει το εντοπιζόμενο πρόσωπο, όπως αναφέρθηκε κατά την ανάλυση της συνάρτησης findFace() στην Ενότητα 3.2.3. Αφού οριστεί μια ζώνη επιτρεπόμενων τιμών εμβαδών (δηλαδή επιτρεπόμενων τιμών απόστασης), ορίζεται η επιφάνεια αναφοράς ως το μέσο της ζώνης αυτής. Αφαιρώντας από την επιφάνεια αναφοράς την μετρούμενη επιφάνεια του περιβάλλοντος ορθογωνίου, εξάγεται το σφάλμα μετατόπισης, το οποίο εισάγεται στον ελεγκτή.

```
def trackFace(tello, info, w, pidYaw, pErrorYaw, pidFB, pErrorFB):
    x_m, y_m = info[0]          # measured x, y coordinates
    area = info[1]
    area_range = [6000, 7000]   # minimum and maximum detected face areas

    # Forward - Backward position PID
    # deviation between measured frame area and mean reference freame area
    errorFB = (area_range[0] + area_range[1]) // 2 - area
    # PD Controller Implementation
    speedFB = pidFB[0] * errorFB + pidFB[2] * (errorFB - pErrorFB)
    # forward/backwards speed saturation control
    speedFB = int(np.clip(speedFB, -30, 30))

    # area outside limits -> move forward/backwards
    if area < area_range[0] or area > area_range[1]:
        tello.for_back_velocity = speedFB
```

```
    # area within limits -> no movement
    else:
        tello.for_back_velocity = 0
        tello.left_right_velocity = 0
        tello.up_down_velocity = 0
        tello.yaw_velocity = 0
        errorFB = 0                    # forward/backward position error
refresh

    if tello.send_rc_control:
        tello.send_rc_control(tello.left_right_velocity,
                              tello.for_back_velocity,
                              tello.up_down_velocity,
                              tello.yaw_velocity)
    return errorFB
```

Ο ελεγκτής λειτούργησε μερικώς, όμως παρουσιάζει ταλάντωση γύρω την μόνιμη κατάσταση, πράγμα που μπορεί να οφείλεται σε λανθασμένη επιλογή κερδών του ελεγκτή, καθώς και σε σφάλμα στην επιλογή συνθήκης σύγκλισης.

4.4. Χειρισμός σμήνους - Swarming

Μέρος της υλοποίησης αποτέλεσε και η απόπειρα προσομοίωσης ενός σμήνος drones, μέσω των δυνατοτήτων που μας έδινε η πλατφόρμα Tello. Εμπνευσμένοι από παρουσιάσεις σμηνών πολύ μεγαλύτερης κλίμακας σε drone shows και αποστολές έρευνας και διάσωσης (Search and Rescue), επιχειρήσαμε να εκτελέσουμε μια από κοινού ακολουθία κινήσεων με τα δύο διαθέσιμα drones του Εργαστηρίου με σκοπό την εκτέλεση ορισμένων αερο-ακροβατικών όπως και τη διαγραφή καθορισμένων τροχιών μετέπειτα. Για την επίτευξη αυτού, έγινε εκμετάλλευση της swarming ιδιότητας των Tello drones, η οποία λειτουργεί μέσω σύνδεσης του κάθε drone σε ένα κοινό δίκτυο WLAN (Wireless Local Area Network) και η αποστολή πακέτων πρωτοκόλλου UDP ταυτοχρόνως και στα δύο συμμετέχοντα ιπτάμενα ρομπότ.

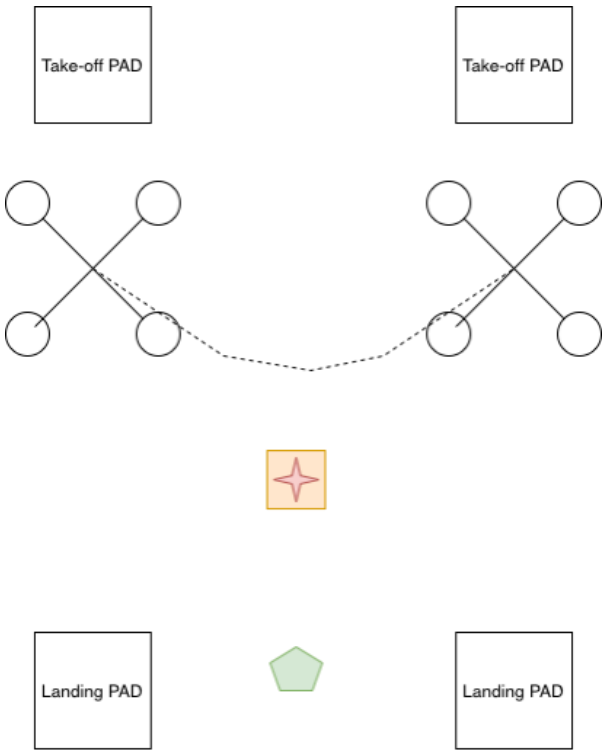
4.4.1. Swarming για Ακολουθία βασικών τροχιών

Για τους σκοπούς της παρουσίασης και της αξιολόγησης των δυνατοτήτων του swarming πρωτοκόλλου στα Tello, δημιουργήθηκε η εξής ακολουθία κινήσεων: κατ΄ αρχάς , τα drones του σμήνους, αφού απογειωθούν, εκτελούν δύο κυβιστήσεις (flips), μία προς τα εμπρός και μία προς τα πίσω ακολουθούμενες από τη διαγραφή μιας τετραγωνικής τροχιάς 700x700 mm. Το σενάριο λήγει με την προσγείωση και των δυο drones στην αρχική θέση απογείωσής τους.

Κατά την εκτέλεση του παραπάνω σεναρίου κίνησης του σμήνους, παρατηρήθηκαν ορισμένοι περιορισμοί, οι οποίοι είχαν αντίκτυπο στην ομοιότητα των κινήσεων που εκτελούσαν τα δύο drones. Συγκεκριμένα, λόγω της πολύ ευαίσθητης, σε εξωτερικούς παράγοντες φωτισμού, ικανότητας localisation των drones, η ακρίβεια με την οποία διόρθωναν τα σφάλματα θέσης τους διέφερε, με αποτέλεσμα την συσσώρευση ενός εύλογου σφάλματος μεταξύ των επιθυμητών και πραγματικών θέσεών τους. Αυτό φυσικά θα ήταν αποφεύξιμο, στον βαθμό του δυνατού, αν υπήρχε επαρκέστερος φωτισμός για να αυξηθεί η αξιοπιστία των μη προσβάσιμων εσωτερικών ελεγκτών, ή τρόπος παρέμβασης στους ελεγκτές αυτούς καθ΄ αυτούς, μέσω του SDK της DJI.

4.4.2. Swarming για Σύνθετη Χρήση, Εύρεση και Ανύψωση βαριδίου από το έδαφος

Η δυνατότητα να σηκωθεί στον αέρα ένα αντικείμενο σταθερού βάρους με την βοήθεια μιας συστάδας από drones μπορεί να φανεί χρήσιμο σε πολλαπλές περιπτώσεις. Για την προσομοίωση ενός τέτοιου ενδεχόμενου έγινε χρήση δύο Tello EDU Drones δεμένα με ένα σκοινί μαυρισμένο και ένα πρώιμο swarming με παραλληλία και ανεξαρτησία. Συγκεκριμένα, χρησιμοποιήθηκε η λογική ανεξάρτητης αρχής για το κάθε drone ξεχωριστά μέσω βασικών συναρτήσεων Lambda λογικής για τον διαχωρισμό μεταξύ των threads, για κάθε drone καθώς και στοιχεία παραλληλίας για την οργάνωση των χρονικών μονάδων των drone. Για βάρος χρησιμοποιήθηκε ένα μαγνητικό άγκιστρο ενώ ο σκοπός της διαδικασίας ήταν να μπορέσουν να σηκώσουν το βάρος από το έδαφος, αφού πρώτα το βρουν, να το ανεβάσουν σε συγκεκριμένο ύψος και εν τέλη να το τοποθετήσουν σε συγκεκριμένο σημείο. Στόχος, να τοποθετηθεί όσο πιο κοντά στο σημείο είναι εφικτό.



Εικόνα 8: Σχεδιάγραμμα του προβλήματος Swarm Payload.

Για την σωστή αντιμετώπιση της συνεχούς δύναμης που ασκείται από το βάρος κατά μήκος του σκοινιού για κάθε ένα από τα drone, χρησιμοποιήθηκαν δυο controllers PD που να σταθεροποιούν το αντίστοιχο drone και να το απομακρύνουν από το άλλο για να μην υπάρχει σύγκρουση. Παράλληλα έγινε και προσπάθεια να μην χρησιμοποιηθεί controller και να γίνει χρήση του υποσυστήματος σταθεροποίησης μέσα από τις εντολές του SDK, πράγμα που δεν είχε ιδιαίτερα καλά αποτελέσματα. Η συμπεριφορά του σκοινιού δεν μοντελοποιήθηκε καθώς αυτό ήταν εκτός των ορίων της εργασίας ενώ το localisation έγινε με την βοήθεια των Pads της έκδοσης EDU καθώς και της κάτω κάμερας. Παρακάτω τίθεται ένα σχεδιάγραμμα του προβλήματος.

Για την σωστή παραμετροποίηση των διαδικασιών του, ώστε να μπορεί το κάθε drone να συμπεριφερθεί ανεξάρτητα, έγινε χρήση της βιβλιοθήκης Python SDK 2.0 Tello η οποία επιτρέπει σειριακή και παράλληλη κίνηση, σε συνδυασμό με lambda function προγραμματισμό για τον διαχωρισμό των TELLO μεταξύ τους,

για την αλληλεπίδραση τους καθώς και για τις ανεξάρτητες κινήσεις τους. Η χρήση συναρτήσεων με γνωρίσματα τα ξεχωριστά TELLO για την σύνθεση πολύπλοκων διαδικασιών καθώς και η χρήση των Pads με δυναμικό τρόπο επιτέλεσε τον καθοριστικό παράγοντα για την επιτυχία της προσπάθειας. Παράλληλα έγινε και προσπάθεια ελέγχου των threads για μεγαλύτερη ακρίβεια κινήσεων, αναγνώριση των δυναμικών συνθηκών μεταξύ των drones καθώς και για λόγους debugging. Για τον σωστό χρονισμό της συνολικής προσπάθειας χρησιμοποιήθηκαν τεχνικές παραλληλοποίησης με await με τον περιορισμό ότι το drone έπρεπε να συνεχίσει να αιωρείται χωρίς απώλειες πτήσης, skew-off και λοιπές αποκλίσεις.
Η συνάρτηση αναζήτησης πιθανών σημείων προσγείωσης φαίνεται στο παρακάτω Snippet. Η διαδικασία κοιτάει συνεχώς για πιθανά pads ενώ παράλληλα κινείται σταθερά προς τα εμπρός με αν απλό Controller ταχύτητας.

```
# Search Function for parallel use of spatial search
```

```
# Contains Controller for Pad Searching, no Reference
def Search(i, tello, boom):
    while boom[i]:
        tello.send_rc_control(0, 10, 0, 0)
        print(i, tello.get_mission_pad_id())
        if tello.get_mission_pad_id() != -1:
            tello.send_rc_control(0, 0, 0, 0)
            boom[i] = False
        time.sleep(1)
```

Ο κώδικας για την βασική λειτουργία της πτήσης είναι ο παρακάτω στον οποίο είναι αισθητά τα σημεία όπου γίνεται έλεγχος της διαδικασίας μέσω Controllers ταχύτητας όπως αναφέρθηκε και στο κεφάλαιο 2 του παρόντος εγγράφου.

```
# Loop Function, Space for Inheriting Controllers
```

```
while True:

    # Get time
    tim = time.perf_counter()

    # For 5 seconds remain above the pad that exists underneath
    while time.perf_counter() - tim < 5:
        swarm.parallel(lambda i, tello: tello.go_xyz_speed_mid(
            0, 0, 30, 30, tello.get_mission_pad_id()))
        swarm.sequential(lambda i, tello: print(tello.get_mission_pad_id()))

    #Move forward and try to catch wait
    swarm.parallel(lambda i, tello: tello.move_forward(35))
    swarm.parallel(lambda i, tello: tello.send_rc_control(0, 10, 0, 0))
```

```
    #Pick up and remain stationery
    swarm.parallel(lambda i, tello: tello.send_rc_control(0, 0, 0, 0))

    #Set the payload at certain hight
    swarm.parallel(lambda i, tello: tello.move_up(60))

    tim = time.perf_counter()
    #For 5 seconds remain in the previous Hight
    while time.perf_counter() - tim < 5:
        swarm.parallel(lambda i, tello: tello.send_rc_control(0, 0, 0, 0))
        # swarm.sequential(lambda i, tello: print(tello.get_mission_pad_id()))
```

```
#Call the Search function to find the next pads somewhere in front
    boom = [True, True]
    while True:
        swarm.parallel(lambda i, tello: Search(i, tello, boom))
        swarm.sync
        break


    tim = time.perf_counter()
    #For 5 seconds remain above the new pads that were found by Search
    while time.perf_counter() - tim < 5:
        swarm.parallel(lambda i, tello: tello.go_xyz_speed_mid(
            0, 0, 30, 10, tello.get_mission_pad_id()))

        swarm.sequential(lambda i, tello: print(tello.get_mission_pad_id()))
```

```
    tim = time.perf_counter()

    #For 5 seconds lower the height to set the payload down
    while time.perf_counter() - tim < 0.5:
        swarm.parallel(lambda i, tello: tello.send_rc_control(0, 0, -5, 0))

    break
```