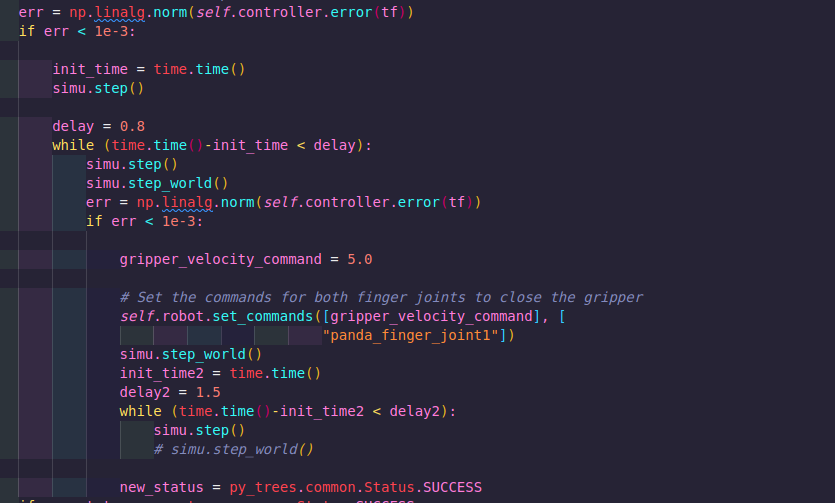
**Εργασία Τεχνολογίες Ευφυών Συστημάτων και Ρομποτική**

**Ονοματεπώνυμο:** Μαρία-Βασιλική Πετροπούλου **ΑΜ:** 1072540

**Ονοματεπώνυμο:** Ίων-Απόστολος Μπουρνάκας-Δρακόπουλος **ΑΜ:** 1075475

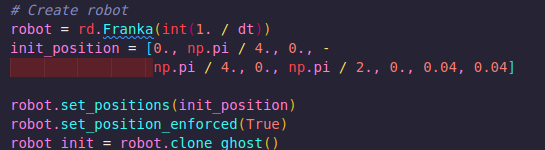
Αρχικά θα πρέπει να τονίσουμε ότι το πρόγραμμα δεν ολοκληρώνεται όπως θα έπρεπε. Αναλυτικότερα, υπήρχαν προσπάθειες για να μπορεί το ρομπότ να σηκώσει το κουτί, δηλαδή να κλείνει η αρπάγη, οι οποίες όμως δεν πέτυχαν. Δηλαδή. Όταν πάει στο πρώτο κουτί, το οποίο πρέπει να σηκώσει το ρομπότ κλείνει την αρπάγη τις περισσότερες φορές. Αλλά υπάρχουν και κάποιες ιδιαίτερες φορές , όπου είτε φταίει η θέση του κουτιού είτε φταίει το περίεργο rotation του end effector και τελικά το ρομπότ δεν καταφέρνει να κλείσει στο κουτί. Αφού λοιπόν καταφέρει να κλείσει στο κουτί παρατηρείται ότι όταν του δοθεί ο επόμενος στόχος που πρέπει να πάει, δηλαδή πάνω από το δεύτερο κουτί για να αφήσει το κουτί που έχει πιάσει , το ρομπότ ανοίγει πάλι την αρπάγη αφήνοντας έτσι το κουτί που είχε πιάσει και εκτελώντας έτσι την κίνηση μεταξύ των κουτιών μόνο του. Ο αντίστοιχος κώδικας που φτιάξαμε για την λειτουργία της αρπάγης είναι ο παρακάτω :



Όπως φαίνεται από τον κώδικα έχουμε βάλει 2 καθυστερήσεις. Η πρώτη είναι για να βεβαιωθούμε ότι το ρομπότ είναι στην θέση που πρέπει (κυρίως για όταν του λέμε να πιάσει εάν κουτί άμα το σπρώξει κατά λάθος αυτό ο κώδικα ως επι το πλείστων το αναγκάζει να πάει στην νέα θέση). Ενώ η δεύτερη είναι για να κλείσει η αρπάγη. Παρόλα αυτά το πρόγραμμα όπως είπαμε άμα καταφέρει να κλείσει την αρπάγη δεν την κρατάει κλειστή και την ανοίγει όταν πάει στον επόμενο στόχο.

Αυτά όσον αναφορά τα προβλήματα στην λειτουργικότητα του προγράμματος. Ας προχωρήσουμε τώρα στην ανάλυση του κώδικα που γράψαμε.

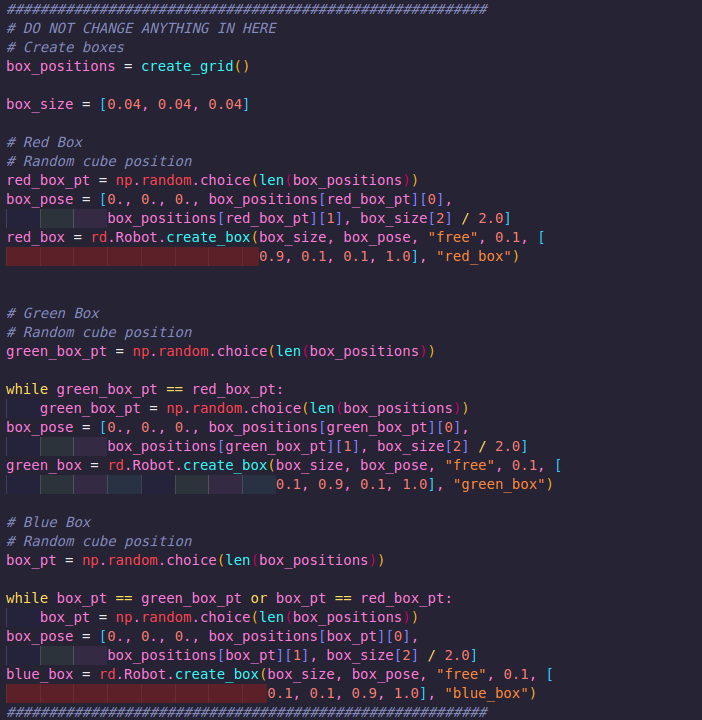
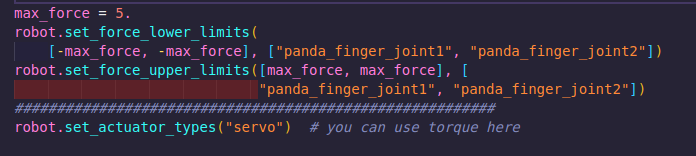
Αρχικά για την αρχικοποίηση του ρομπότ χρησιμοποιούνται αυτές οι εντολές:



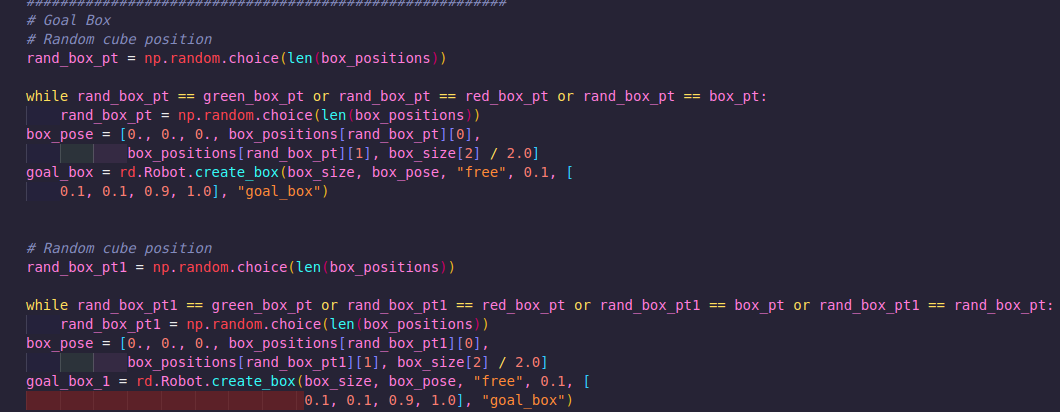
Η πρώτη εντολή λέει στο πρόγραμμα ποιο ρομπότ θα χρησιμοποιηθεί, η δεύτερη τις αρχικές θέσεις αυτού του ρομπότ (στο περιβάλλον προσομοιωτή). Όσον αναφορά την τρίτη εντολή σκεφτήκαμε λίγο τι διαφορά θα είχε άμα το αλλάζαμε από False σε True. Καταλήξαμε στα εξής συμπεράσματα :

robot.set\_position\_enforced(False): Εάν ορίσουμε το flag με False για τα DOF (Degrees of freedom), αυτά τα DOF δεν θα περιοριστούν στις επιθυμητές θέσεις τους. Η προσομοίωση θα επιτρέψει σε αυτά τα DOF να κινούνται φυσικά υπό την επίδραση δυνάμεων και ενδέχεται να μην ακολουθούν στενά τις επιθυμητές τροχιές.  
  
robot.set\_position\_enforced(True): Από την άλλη πλευρά, εάν ορίσουμε το flag με True για τα DOF, η προσομοίωση θα λειτουργήσει ενεργά για να διατηρήσει αυτά τα DOF στις επιθυμητές θέσεις τους. Ακόμα κι αν εξωτερικές δυνάμεις προσπαθήσουν να απομακρύνουν αυτά τα DOF από τις επιθυμητές γωνίες, η προσομοίωση θα αντισταθεί σε αυτές τις δυνάμεις και θα προσπαθήσει να κρατήσει τα DOF κοντά στις θέσεις στόχου.  
  
Συνοπτικά, η διαφορά μεταξύ της ρύθμισης robot.set\_position\_enforced(False) και robot.set\_position\_enforced(True) έγκειται στον τρόπο με τον οποίο η προσομοίωση χειρίζεται την επιβολή των επιθυμητών θέσεων των αρθρώσεων. Η ρύθμιση του σε False επιτρέπει μεγαλύτερη ευελιξία στις κινήσεις των αρθρώσεων, ενώ η ρύθμιση του σε True επιβάλλει τις θέσεις των αρθρώσεων πιο άκαμπτα. Η επιλογή εξαρτάται από τις συγκεκριμένες απαιτήσεις της προσομοίωσής σας και τη συμπεριφορά που θέλετε να επιτύχετε για τις αρθρώσεις του ρομπότ. Εμείς επιλέξαμε True μιας και έχουμε αλλά σώματα τα οποία θέλουμε να αλληλοεπιδρούν με το ρομπότ μας.

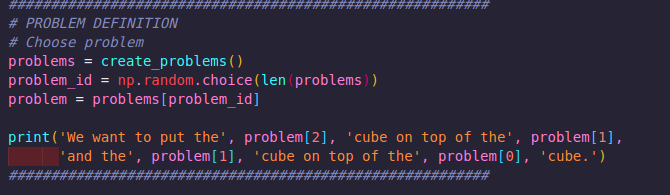
Η τελευταία εντολή της εικόνας είναι για να φτιάξουμε ένα ρομπότ - κλώνο του ρομπότ μας στην αρχική του θέση. Αυτό θα χρησιμοποιηθεί αργότερα για να μπορούμε να πάρουμε την θέση του ee που είχε το ρομπότ όταν άρχισε το simulation.



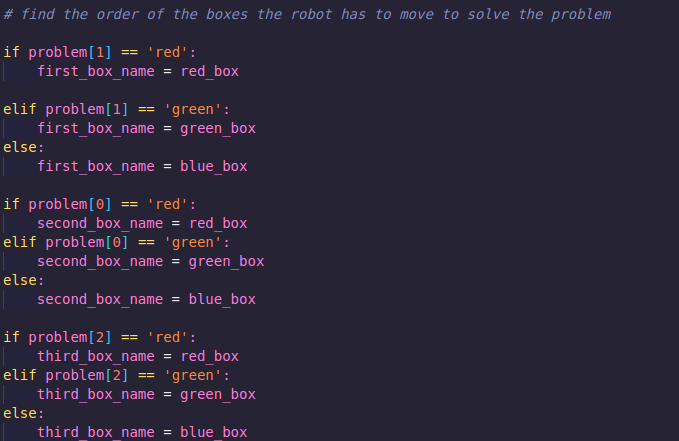
Ο παραπάνω κώδικας δόθηκε έτοιμος και έτσι δεν αλλάξαμε κάτι σε αυτόν. Με λίγα λόγια στην αρχή βάζουμε κάποια όρια στις δυνάμεις που μπορούν να λάβουν οι αρθρώσεις και λέμε στο ρομπότ να χρησιμοποιήσει servo actuators (πράγμα που κάνει τις παραπάνω εντολές για τις δυνάμεις λίγο περιττές ). Στην συνέχεια φτιάχνονται τα τρία κουτιά χρησιμοποιώντας την συνάρτηση create\_grid(), για να καθοριστεί η θέση τους, η οποία δημιουργεί μια λίστα με θέσεις πλέγματος εντός καθορισμένων ορίων και μεγεθών βημάτων. Τα κουτιά φτιάχνονται αφού εξασφαλίζεται ότι κανένα κουτί δεν έχει πάρει τις ίδιες αρχικές θέσεις με κάποιο άλλο.



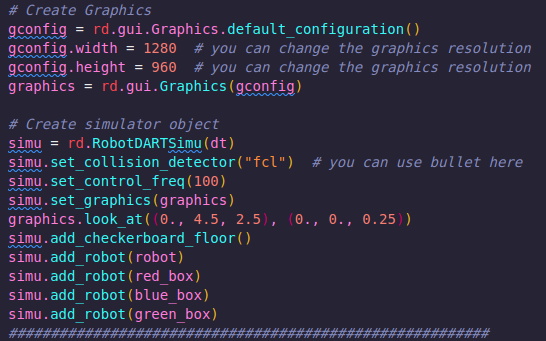
Εδώ δημιουργούμε εμείς δυο έξτρα κουτιά , τα οποία δεν εμφανίζουμε στο simulation προφανώς , για να μπορούμε να στείλουμε αυτά στις συναρτήσεις που έχουμε φτιάξει για να κουνάμε το ρομπότ στις θέσεις που θέλουμε. Αυτό γίνεται γιατί έχουμε φτιάξει τις συναρτήσεις να δέχονται ολόκληρα αντικείμενα (κουτιά) για στόχο του ρομπότ. Οπότε αφού θέλουμε να στείλουμε σαν στόχο , αφού πιάσουμε το δεύτερο κουτί , την θέση του πρώτου κουτιού αλλά με λίγο παραπάνω ύψος , και το ίδιο και όταν θέλουμε να βάλουμε στην στοίβα το τρίτο, θα πρέπει να έχουμε δυο κουτιά αόρατα που στέλνουμε σαν στόχο στην συνάρτηση.



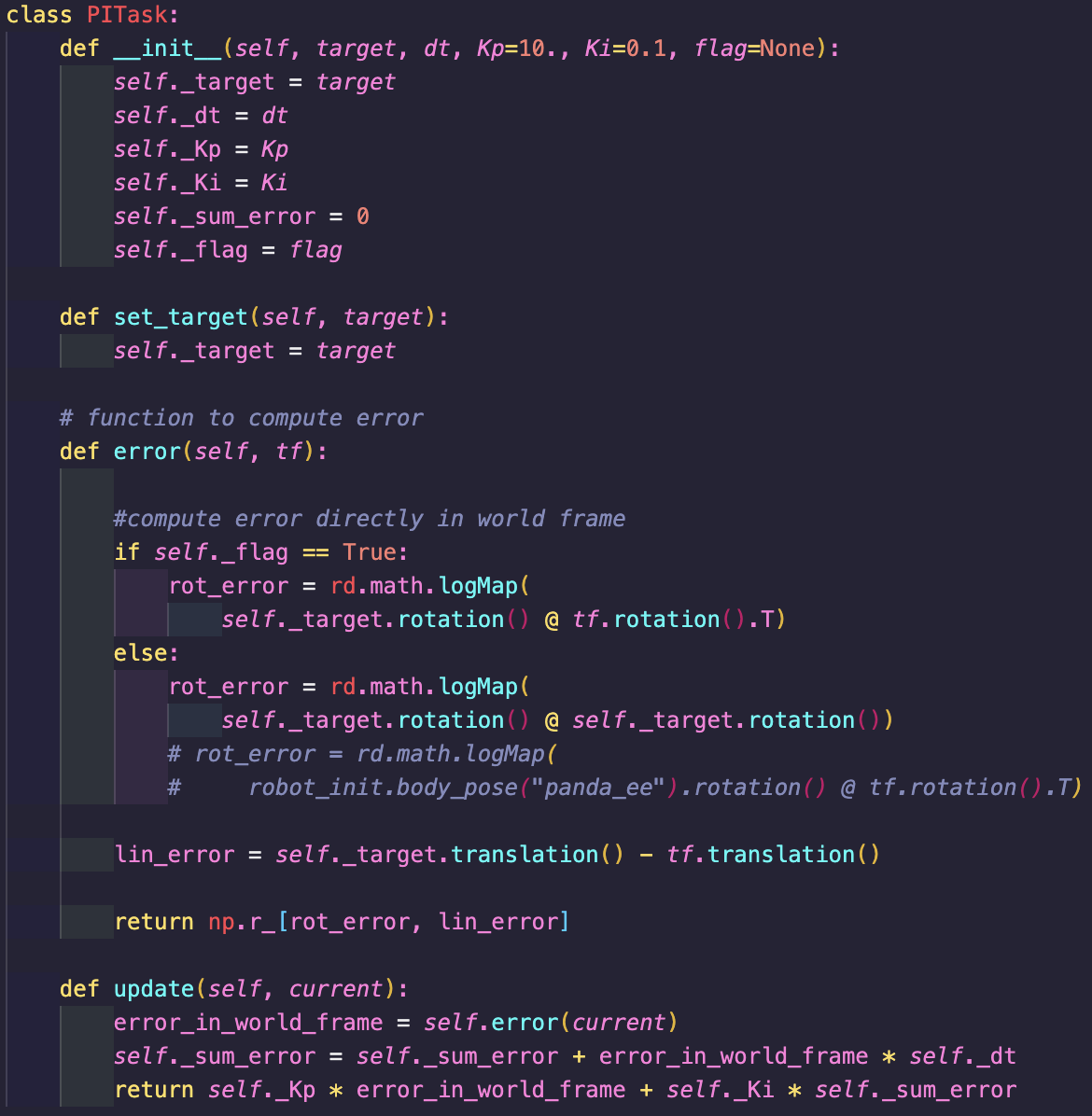
Εδώ πάλι ο κώδικας ήταν έτοιμος αλλά με λίγα λόγια διαλέγει τυχαία την σειρά με την οποία θα πρέπει να φτιάξουμε τα κουτιά στην στοίβα.



Εμείς με αυτό καταφέραμε και γράψαμε τον παραπάνω κώδικα οπού αναθέτει κάθε ένα από τα κουτιά σε μια μεταβλητή με το κατάλληλο όνομα (first, second, third) για να ξέρουμε κάθε φορά σε ποιο κουτί πρέπει να πάμε.



Ο παραπάνω κώδικα αρχικά φτιάχνει και ορίζει τα γραφικά που θα χρησιμοποιήσει ο Simulator και στην συνέχεια φτιάχνει τα αντικείμενα σε αυτόν (κουτιά , πάτωμα , ρομπότ).



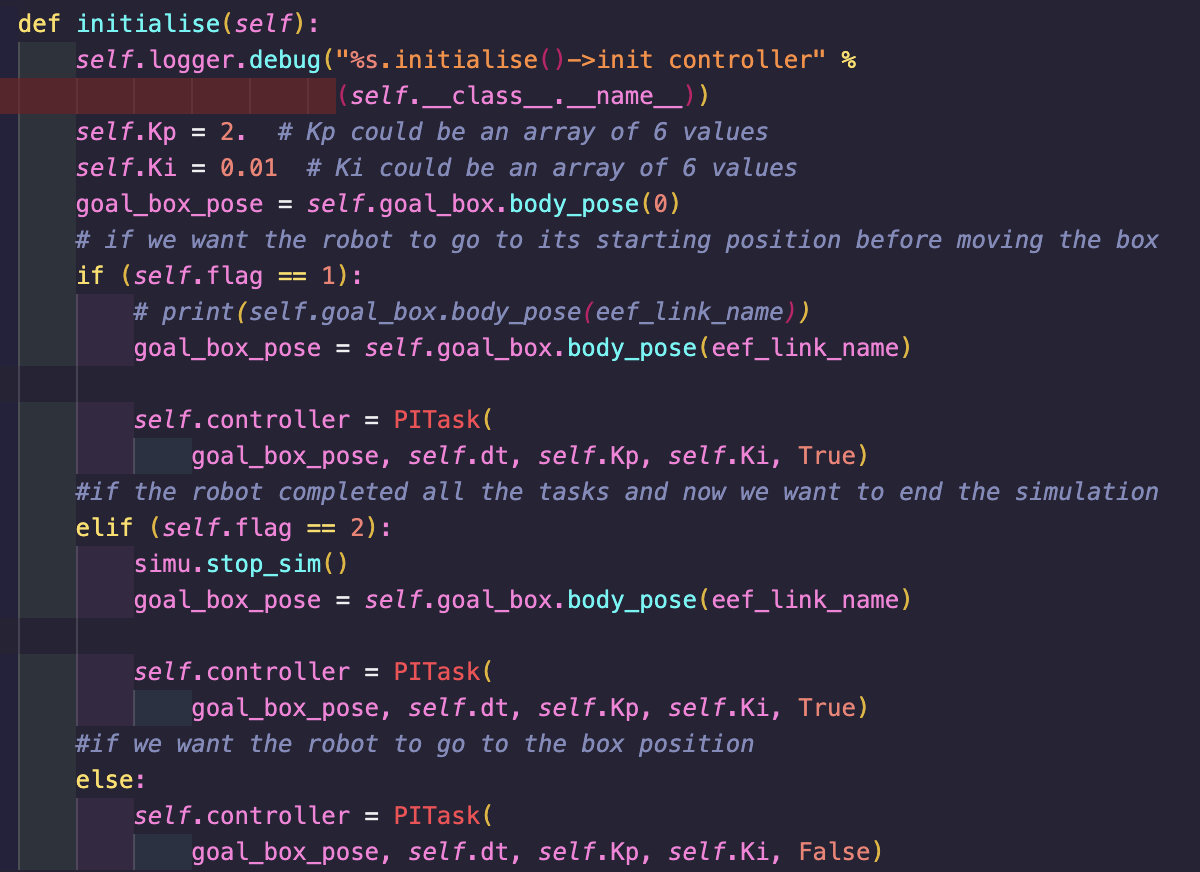
Ο παραπάνω κώδικας είναι αρκετά παρόμοιος με αυτόν που δίνεται στο φροντιστήριο. Έχουμε κάνει όμως κάποιες αλλαγές τις οποίες θα αναλύσουμε. Αρχικά ας περιγράψουμε τι κάνει αυτή η κλάση και γιατί υπάρχει.

* Η μέθοδος \_\_init\_\_ αρχικοποιεί το αντικείμενο με διάφορες παραμέτρους:
* target: Η τιμή ή η πόζα στόχος που προσπαθεί να επιτύχει η εργασία.
* dt: Το χρονικό βήμα ή το διάστημα μεταξύ των ενημερώσεων.
* Kp: Ο αναλογικός συντελεστής για την ελεγχόμενη ενέργεια (προεπιλογή είναι 10.0).
* Ki: Ο ολοκληρωτικός συντελεστής για την ελεγχόμενη ενέργεια (προεπιλογή είναι 0.1).
* Sum error: Αυτή η μεταβλητή θα αποθηκεύσει το αθροιστικό σφάλμα με την πάροδο του χρόνου.
* flag: Σημαία που υποδεικνύει εάν πρέπει να εκτελεστεί μια συγκεκριμένη λειτουργία (η προεπιλογή είναι None). Στην περίπτωση μας έχουμε βάλει το ρομπότ να επιστρέφει στην αρχική του θέση μετά την μεταφορά του πρώτου κουτιού (για να αποφύγουμε τυχόν συγκρούσεις άμα πάει κατευθείαν στο τρίτο). Οπότε το flag μας βοηθάει να ξέρουμε ποτέ γίνεται αυτό και έτσι αλλάζει τρόπος που υπολογίζουμε το error.
* Η μέθοδος set\_target μας επιτρέπει να ενημερώσουμε την τιμή της μεταβλητής target.
* Η μέθοδος error υπολογίζει το σφάλμα μεταξύ της τρέχουσας κατάστασης (δοθείσα από tf) και της επιθυμητής κατάστασης:
  + Εάν το \_flag είναι True, υπολογίζει το rotation σφάλμα μεταξύ της περιστροφής της επιθυμητής και της τρέχουσας περιστροφής.
  + Εάν το \_flag δεν είναι True, υπολογίζει το περιστροφικό σφάλμα μεταξύ της περιστροφής της επιθυμητής και της ίδιας της περιστροφής (που πρέπει να είναι μηδέν). Αυτό το κάνουμε γιατί άμα προσπαθήσουμε να κάνουμε match το rotation του κουτιού και του ee tου ρομπότ τότε παρουσιάζονται προβλήματα (αφού το rotation του κουτιού είναι 0 και του ρομπότ δεν είναι ποτέ).
  + Το γραμμικό σφάλμα (lin\_error) είναι η διαφορά στη μετάφραση μεταξύ της επιθυμητής και της τρέχουσας κατάστασης.
  + Η συνάρτηση np.r\_ συνενώνει τα περιστροφικά και γραμμικά σφάλματα σε έναν μονοδιάστατο πίνακα.
* Η μέθοδος update υπολογίζει την ενέργεια που πρέπει να κάνει ο controller με βάση την τρέχουσα κατάσταση:
  + Υπολογίζει το σφάλμα στο world frame χρησιμοποιώντας τη μέθοδο error.
  + Ενημερώνει την μεταβλητή sum\_error προσθέτοντας όλα τα error και το dt.
  + Επιστρέφει την ελεγχόμενη ενέργεια, η οποία είναι ένας συνδυασμός των αναλογικού και ολοκληρωτικων όρων, πολλαπλασιασμένων με τα αντίστοιχα error.

Τώρα για την κλάση ReachTarget. Θα την αναλύσουμε σε παραπάνω από ένα βήματα για να γίνει πιο κατανοητή η λειτουργία της.



* Η μέθοδος \_\_init\_\_ είναι ο κατασκευαστής της κλάσης ReachTarget, που κληρονομεί από την βάση py\_trees.behaviour.Behaviour. Ο κατασκευαστής αρχικοποιεί το αντικείμενο με τις παραμέτρους:
  + robot: Το ρομπότ που πρόκειται να ελεγχθεί.
  + tf\_desired: Η τρέχουσα κατάσταση (θέση και περιστροφή) του end effector.
  + dt: Το χρονικό βήμα για τις ενημερώσεις.
  + goal\_box: Το αντικείμενο που αντιπροσωπεύει το στόχο που πρέπει να επιτευχθεί (που πρέπει να πάει ο end effector).
  + name: Το όνομα του κόμβου του δέντρου.
  + flag: Μια σημαία που υποδηλώνει κάποια ειδική λειτουργία (Όπως και πριν για να ξέρουμε ποτέ θέλουμε το ρομπότ να μεταφέρει κουτί η να πάει στην αρχική θέση).
* Η μέθοδος setup εκτελείται κατά τη διάρκεια της αρχικοποίησης του στοιχείου ελέγχου. Σε αυτή την περίπτωση, δεν πραγματοποιείται καμία ενέργεια, αλλά καταγράφεται μόνο ένα μήνυμα στο log για να υποδείξει ότι η μέθοδος εκτελέστηκε.



* Η μέθοδος initialise καλείται κατά την αρχικοποίηση του controller. Κατά την εκτέλεσή της, ορίζει τους πίνακες ελέγχου Kp και Ki για τον controller.
  + Στη συνέχεια, παίρνει την θέση για αυτό του κουτιού-στόχου χρησιμοποιώντας τη μέθοδο body\_pose του αντικειμένου goal\_box με δείκτη 0, που αναφέρεται στο βασικό του σώμα.
  + Στο πρώτο "if", εάν το flag έχει την τιμή 1, τότε το ρομπότ σχεδιάζεται να πάει στην επιθυμητή του θέση (θέση του end effector) πριν μετακινήσει το κουτί. Η θέση του εφέδρου λαμβάνεται από το goal\_box με τη μέθοδο body\_pose(eef\_link\_name). ακόμη δημιουργείται ένα αντικείμενο ελεγκτή PITask με την παράμετρο True, υποδεικνύοντας ότι ο ελεγκτής θα υπολογίσει το σφάλμα στην περιστροφή εκτός από μόνο την θέση.
  + Στο δεύτερο "elif", εάν το flag έχει την τιμή 2, τότε καλείται η μέθοδος stop\_sim για να τερματίσει την προσομοίωση.
  + Στο τρίτο "else", εάν το flag έχει οποιαδήποτε άλλη τιμή, το ρομπότ προσαρμόζεται για να εκτελέσει τον ελεγκτή PITask και να πάει στην επιθυμητή του θέση (θέση του κουτιού στόχου).

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

* Η μέθοδος update υλοποιεί τη λειτουργία του controller κατά τη διάρκεια κάθε ενημέρωσης του συστήματος.
  + Αρχικά, ορίζεται η αρχική κατάσταση του controller ως RUNNING.
  + Στον "έλεγχο του ρομπότ", παίρνεται η θέση και o προσανατολισμός του end effector του ρομπότ (eef\_link\_name) με τη χρήση της μεθόδου body\_pose.
  + Υπολογίζεται η επιθυμητή ταχύτητα με τη χρήση του ελεγκτή που έχει οριστεί (controller.update(tf)).
  + Υπολογίζεται ο jacobian του end effector του ρομπότ, και στη συνέχεια, υπολογίζεται η ψευδοαντίστροφη της με τη χρήση της μεθόδου damped\_pseudoinverse.
  + Υπολογίζονται οι εντολές ελέγχου των αρθρώσεων του ρομπότ με την εξίσωση cmd = jac\_pinv @ vel.
  + Υπολογίζεται το σφάλμα μεταξύ της επιθυμητής θέσης και της τρέχουσας θέσης του ρομπότ.
  + Εάν το σφάλμα είναι μικρότερο από ένα πολύ μικρό κατώφλι (1e-3), τότε ξεκινάει μια διαδικασία. Αυτή η διαδικασία έχει τρία πιθανά αποτελέσματα. Αρχικά πρέπει να είναι στην σωστή θέση ο end effector (το error είναι πολύ μικρό) για παραπάνω από μια στιγμή (για να βεβαιωθούμε άμα σπρωχτεί το κουτί ο end effector θα προσπαθήσει να το πιάσει στην καινούρια θέση). Αφού είναι στην σωστή θέση τότε αρχίζει να κλείνει το gripper. Για να προλάβει να κλείσει τον gripper έχουμε προσθέσει άλλο ένα delay το οποίο αφού περάσει και κλείσει ο gripper σημειώνουμε σαν επιτυχές το status.
  + Εάν το status είναι success , ο controller παίρνει την κατάσταση SUCCESS και ενημερώνει το log.
  + Διαφορετικά, εάν το σφάλμα δεν είναι τόσο μικρό, ο controller παραμένει σε κατάσταση RUNNING και ενημερώνει το μήνυμα σφάλματος.
* Η μέθοδος terminate καλείται κατά την λήξη της λειτουργίας του controller. Επίσης καταγράφεται η κατάσταση του controller πριν και μετά τον τερματισμό του.

A computer screen shot of text

Description automatically generated

Αυτός ο κώδικας τρέχει όταν αρχίσει το πρόγραμμα και Μετα καλεί τις κλάσεις που αναφέραμε πιο πριν.

Αρχικά, λαμβάνεται η θέση και ο προσανατολισμός του end effector του ρομπότ με το όνομα eef\_link\_name χρησιμοποιώντας τη μέθοδο body\_pose και body\_pose\_vec. Οι θέσεις και οι προσανατολισμοί των συνδέσμων του ρομπότ μπορούν να ληφθούν με αυτές τις μεθόδους.

Στη συνέχεια, ρυθμίζεται το επίπεδο του log για το Py Trees.

Δημιουργείται η ρίζα του δέντρου συμπεριφοράς (root) ως ένα σύνθετο κόμβο παράλληλης εκτέλεσης. Ορίζεται πολιτική εκτέλεσης όπου το δέντρο επιστρέφει επιτυχία αν ένας μόνο κόμβος επιτύχει.

Δημιουργείται ένας κόμβος ακολουθίας (sequence) για να εκτελούνται οι στόχοι σε ακολουθία.

Δημιουργείται ο στόχος Α (trA) με χρήση της κλάσης ReachTarget. Προστίθεται αυτός ο κόμβος στον κόμβο ακολουθίας. Στην κλάση στέλνουμε σαν tf\_desired την θέση του end effector και σαν στόχο το box που πρέπει να σηκώσει το ρομπότ. Στην συγκεκριμένη περίπτωση το κουτί που σηκώνει είναι αυτό που πάει δεύτερο στην στοίβα.

Για τον κόμβο Β δημιουργείται ένα αντικείμενο tf που αντιστοιχεί στη θέση και τον προσανατολισμό του end effector του ρομπότ. Στη συνέχεια, η θέση του tf αντικαθίσταται με την θέση του στόχου Β (second\_box\_name.body\_pose(0)). Ακόμα, αλλάζει η θέση της θετικής ζώνης του άξονα Z κατά 0.08, και αυτό χρησιμοποιείται για να ενημερωθεί η θέση του goal\_box. Στη συνέχεια, δημιουργείται ο στόχος Β (trB) και προστίθεται στον κόμβο ακολουθίας.

A computer screen shot of text

Description automatically generated

Δημιουργείται ένα αντικείμενο tf που αντιστοιχεί στη θέση και τον προσανατολισμό του end effector του ρομπότ.

Δημιουργείται ο στόχος Αρχικός (trIn) χρησιμοποιώντας την κλάση ReachTarget. Ο στόχος αυτός έχει σκοπό να μετακινήσει το ρομπότ στην αρχική του θέση για να αποφευχθούν συγκρούσεις με κάποιο άλλο κουτί κατά την κίνηση από όταν άφησε το δεύτερο κουτί προς το τρίτο. Η παράμετρος 1 που περνάται ως flag στον κατασκευαστή της ReachTarget υποδηλώνει την χρήση διαφορετικών εντολών μέσα στην ReachTarget από όταν μετακινεί κουτιά (όπως εξηγήσαμε πιο πριν).

Ο στόχος Γ (trC) δημιουργείται παρόμοια με τον προηγούμενο, εκτός από το γεγονός ότι έχει σκοπό να μετακινήσει το ρομπότ προς το τρίτο κιβώτιο της στοίβας για να το πιάσει. Η παράμετρος 0 που περνάται ως flag υποδηλώνει κανονική κίνηση στον κόμβο.

A screenshot of a computer program

Description automatically generated

Ο στόχος Δ (trD) δημιουργείται για να τοποθετηθεί το τρίτο κιβώτιο πάνω από το δεύτερο κιβώτιο. Για να γίνει αυτό, το ρομπότ πρέπει να μετακινηθεί με ασφάλεια για να ανασηκώσει το τρίτο κιβώτιο. Η παράμετρος 0 που περνάται ως flag υποδηλώνει κανονική κίνηση.

Ο στόχος Τέλος (trEnd) δημιουργείται ώστε να τερματίσει η προσομοίωση. Η παράμετρος 2 που περνάται ως flag υποδηλώνει ότι η εργασία που επιθυμούμε είναι να τερματιστεί η προσομοίωση.

Στη συνέχεια, προστίθεται η ακολουθία στον κόμβο ρίζας (root) του δέντρου συμπεριφοράς.

Ο κώδικας συνολικά δημιουργεί ένα δέντρο συμπεριφοράς (Behavior Tree) με έξι κόμβους στόχων (trA, trB, trIn, trC, trD, trEnd) που εκτελούνται σε ακολουθία. Οι στόχοι είναι αντικείμενα της κλάσης ReachTarget που χρησιμοποιούνται για να εκτελέσουν εργασίες ελέγχου πάνω στο ρομπότ.

Τέλος, γίνεται η εκτέλεση της ακολουθίας και ο έλεγχος της προσομοίωσης για έναν αριθμό συγκεκριμένων βημάτων, όπως ορίζεται από τη μεταβλητή total\_steps.