



ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΕΛΛΑΔΟΣ

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ,
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ρομποτικός Βραχίονας
Ταξινόμηση αντικειμένων ανά χρώμα

Οδηγός με βήματα στον προσομοιωτή CoppeliaSim

Ομαδική Εργασία:
Αναστασιάδης Αλκίνοος (20003)
Ζήνα Ελένη (20046)

Επιβλέπων:
Κουρέας Αργύριος

ΣΕΠΡΕΣ, 2024

Περιεχόμενα

Κεφάλαιο 1	3
Εισαγωγή.....	3
Εγχειρίδιο χρήσης για το CoppeliaSim.....	4
Κεφάλαιο 2	5
Προσθήκη δομικών στοιχείων	5
Κεφάλαιο 3	19
Κώδικας αυτόματης δημιουργίας αντικειμένων με τυχαίο χρώμα	19
Κεφάλαιο 4	21
Κώδικας διαχείρισης αισθητήρα χρώματος στον ιμάντα μεταφοράς	21
Κεφάλαιο 5	23
Διαχείριση κίνησης του ρομποτικού βραχίονα.....	23
Βιβλιογραφία	34

Κεφάλαιο 1

Εισαγωγή

Ο ρομποτικός βραχίονας αποτελεί μια σημαντική εφεύρεση στον τομέα της ρομποτικής. Με τη δυνατότητα να μιμείται τις κινήσεις του ανθρώπινου χεριού, μπορεί να εκτελεί ποικίλες εργασίες με ακρίβεια και αποτελεσματικότητα. Ένα από τα παραδείγματα χρήσης του είναι η ταξινόμηση αντικειμένων ανά χρώμα. Με την χρήση του λογισμικού CoppeliaSim (v4.6.0 rev .18), μπορούμε να προσομοιώσουμε και να ελέγξουμε τον ρομποτικό βραχίονα για την αυτόματη ταξινόμηση αντικειμένων ανά χρώμα. Σε αυτήν την εργασία, θα μάθουμε πώς λειτουργεί αυτή η διαδικασία και πώς μπορεί να εφαρμοστεί με απλό και κατανοητό τρόπο.

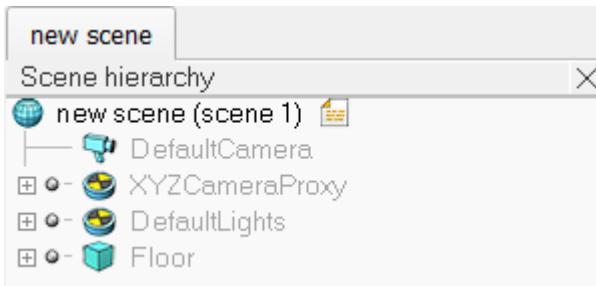
Εγχειρίδιο χρήσης για το CoppeliaSim

Μενού:



Περιγραφή επιλογών μενού:

- : μετακίνηση κάμερας στην επιφάνεια (camera pan)
- : περιστροφή κάμερας (camera rotate)
- : ζουμ στο επιλεγμένο αντικείμενο (fit to view)
- : μετακίνηση του επιλεγμένου αντικειμένου (object/item shift)
- : περιστροφή του επιλεγμένου αντικειμένου (object/item rotate)
- : εκκίνηση – παύση – διακοπή προσομοίωσης (start/resume – suspend – stop simulation)



- : ιεραρχικό δέντρο σκηνής (scene hierarchy) – εδώ θα εμφανίζονται όλα τα δομικά στοιχεία που προστέθηκαν



- : λίστα δομικών στοιχείων (model browser)

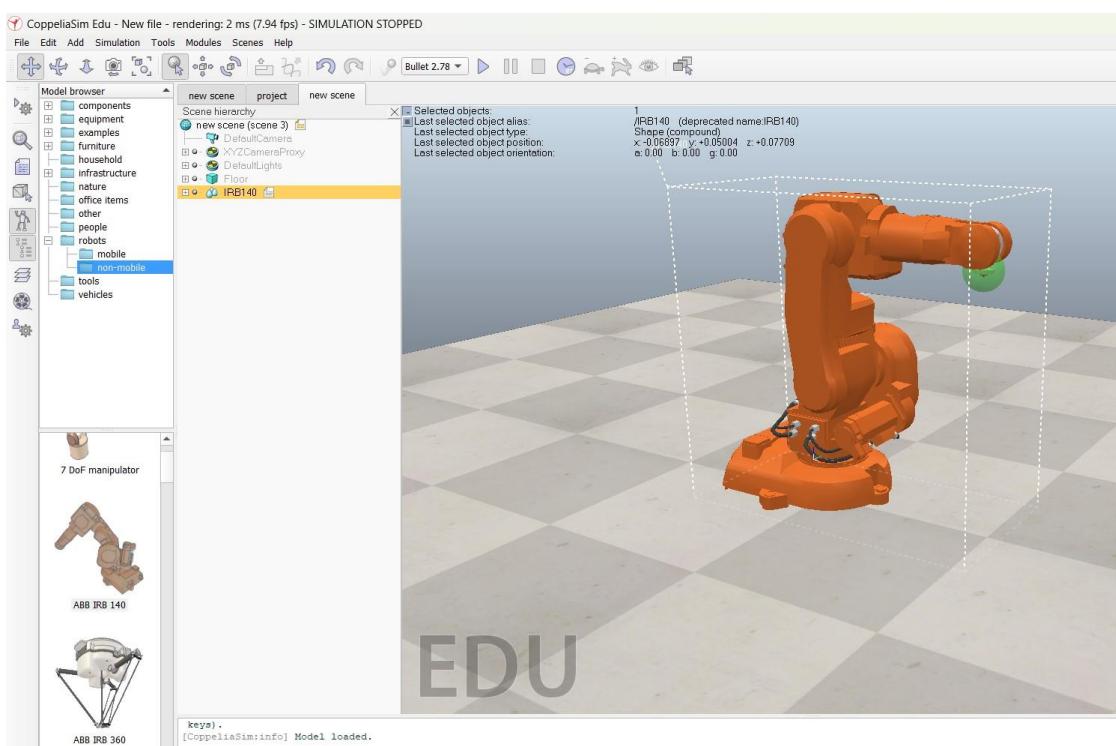
Κεφάλαιο 2

Προσθήκη δομικών στοιχείων

Για να προσθέσουμε τα δομικά στοιχεία που θα χρειαστούμε, τα παίρνουμε από το model browser και τα σέρνουμε μέσα στο χώρο της προσομοίωσης. Το model browser διαθέτει πολλές υποκατηγορίες από πράγματα που μπορούμε να προσθέσουμε, όπως ρομπότ (με/χωρίς κίνηση), τραπέζια, ελικόπτερα κτλ. Συγκεκριμένα, για την προσομοίωση θα χρειαστούν τα παρακάτω:

Ρομποτικός βραχίονας ABB IRB 140:

Ο συγκεκριμένος ρομποτικός βραχίονας είναι «φτιαγμένος» ειδικά για τον σκοπό της άσκησης, ώστε να μπορούμε να διαχειριστούμε εύκολα και γρήγορα τις κινήσεις του με λίγες εντολές. Προφανώς, ο ρομποτικός βραχίονας έχει τον κύριο ρόλο στην προσομοίωση για να ταξινομήσει τα κουτάκια στους κάδους που θα προστεθούν. Τον βρίσκουμε μέσα από το model browser, πηγαίνοντας στο **Model browser>robots>non-mobile** και πατώντας το, το σέρνουμε στο χώρο της προσομοίωσης.

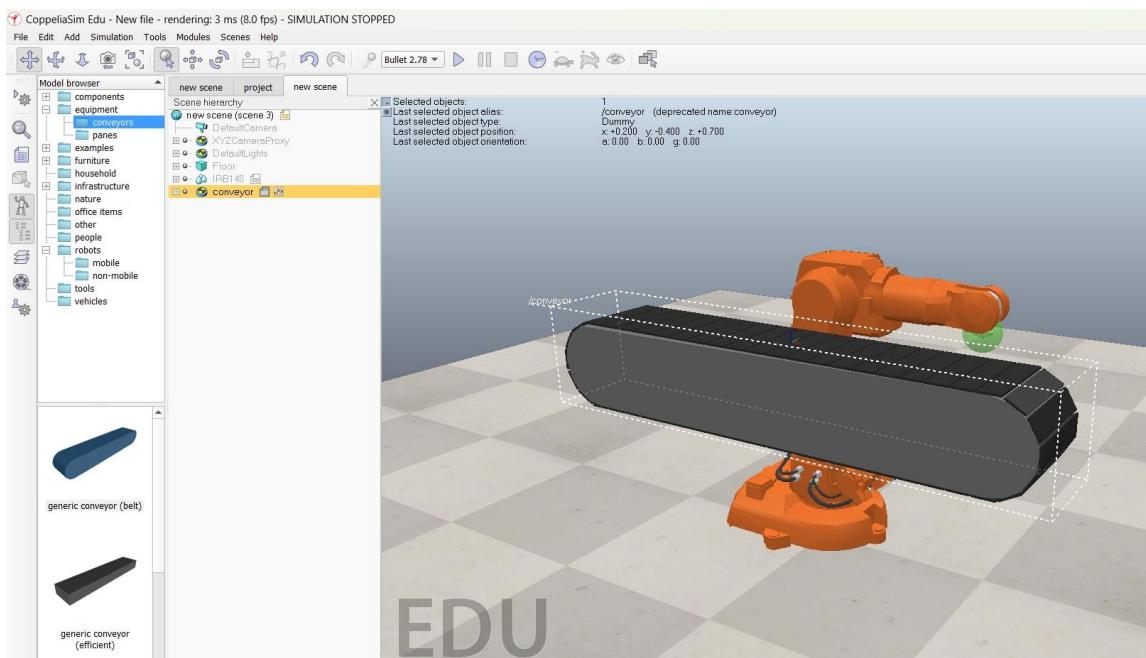


Εικόνα 1 Τοποθέτηση ρομποτικού βραχίονα

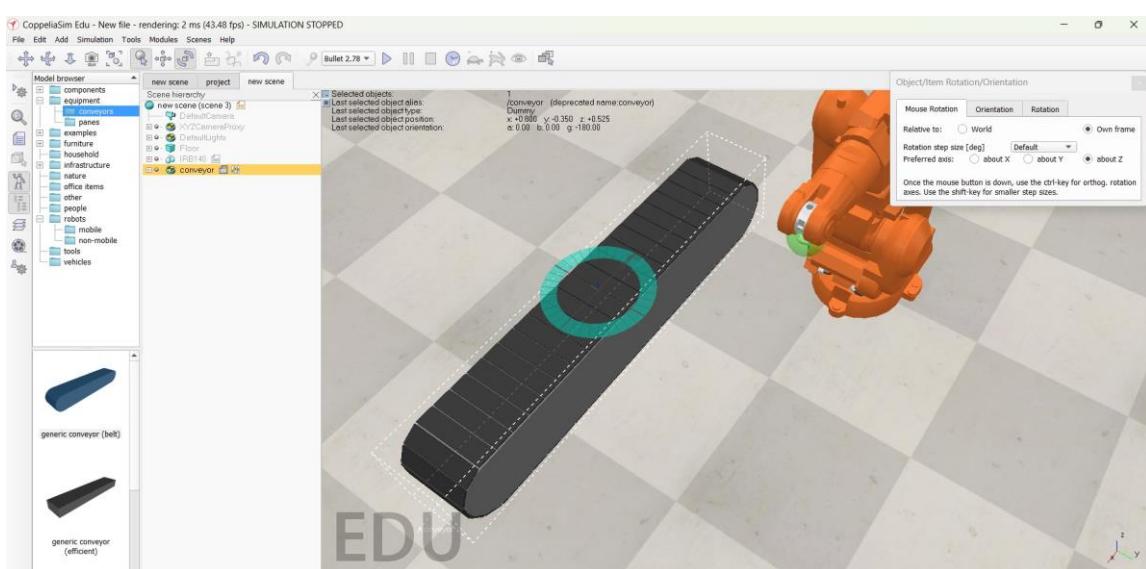
Οι εικόνες που απεικονίζονται είναι ενδεικτικές για την καθοδήγηση της άσκησης όσον αφορά τις συντεταγμένες τους (x, y, z), γιατί αυτό θα έχει σημασία αργότερα. Οι συντεταγμένες του ρομποτικού βραχίονα είναι (**x=-0.34397, y=+0.77504, z=+0.07709**). Μπορούν να αλλάξουν αυτές οι τιμές εύκολα με το κουμπί «object/item shift» στην καρτέλα «Position» με επιλεγμένο το «World».

Conveyor – ιμάντας μεταφοράς:

Ο ιμάντας μεταφοράς (conveyor) είναι αυτός που θα φέρει σε κατάλληλο σημείο τα κουτάκια στον ρομποτικό βραχίονα για να τα πάρει. Έχει σημασία ο τρόπος τοποθέτησης του ιμάντα, καθώς πρέπει ο ρομποτικός βραχίονας να έχει την ελευθερία κίνησης για να πιάσει καλά το κουτάκι. Τον βρίσκουμε μέσα από το model browser, πηγαίνοντας στο **Model browser>equipment>conveyors** και επιλέγουμε το «**generic conveyor (belt)**». Μετά την τοποθέτηση του ιμάντα μεταφοράς, τον κάνουμε περιστροφή 180 μοίρες, έχοντας τον επιλεγμένο και πατώντας το κουμπί «*fit to view*» που αναφέρθηκε στο Κεφάλαιο 1. Αυτή η περιστροφή είναι αναγκαία για να έρχονται τα κουτάκια από την αντίθετη κατεύθυνση προς τον ρομποτικό βραχίονα.



Εικόνα 2 Τοποθέτηση ιμάντα μεταφοράς

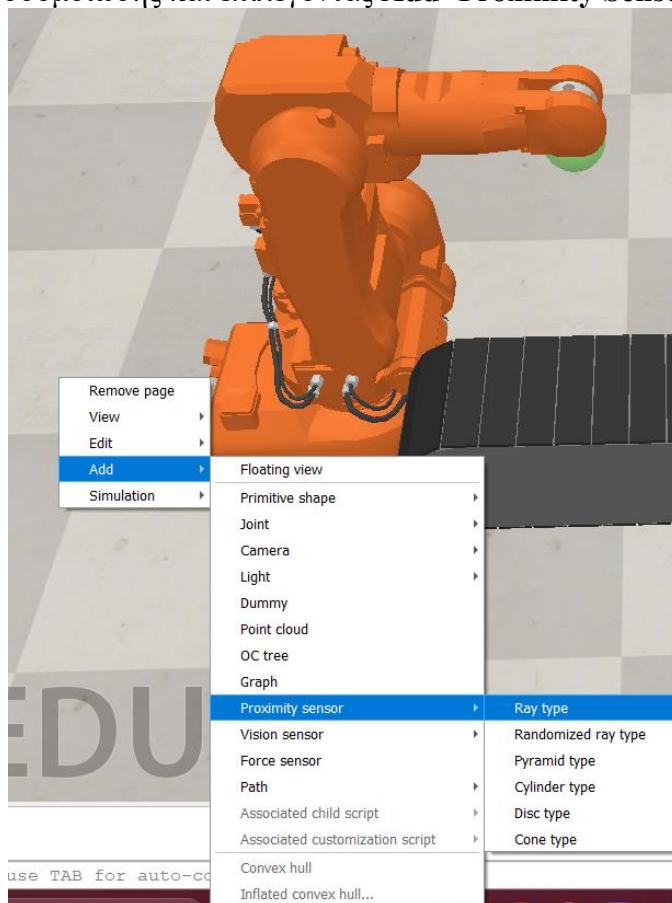


Εικόνα 3 Περιστροφή ιμάντα μεταφοράς

Οι συντεταγμένες του υμάντα μεταφοράς είναι (**x=+0.600, y=+0.425, z=+0.525**). Ένας άλλος τρόπος περιστροφής του είναι με το κουμπί «object/item rotate» στην καρτέλα «Orientation» με επιλεγμένο το «World» στο «Gamma [deg]» ίσο με +180.00 και τα υπόλοιπα 0.00.

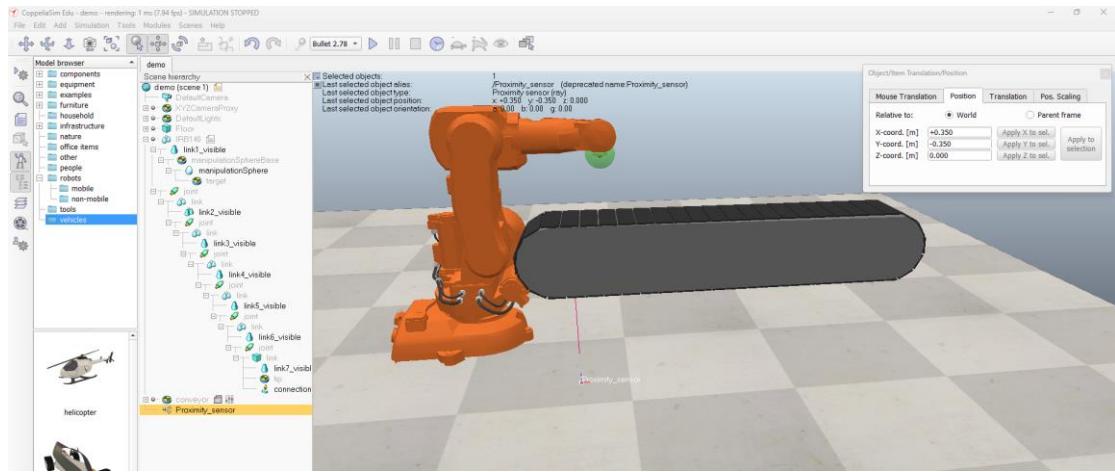
Proximity sensor – αισθητήρας ανίχνευσης χρώματος:

Ο αισθητήρας ανίχνευσης χρώματος (proximity sensor) είναι αυτός που θα ανιχνεύει το χρώμα από κάθε κουτάκι και θα ενημερώνει τον ρομποτικό βραχίονα σε ποιον κάδο θα το ρίξει. Έχει σημασία ο τρόπος τοποθέτησης του αισθητήρα ανίχνευσης χρώματος, καθώς πρέπει να είναι ακριβώς πάνω από τον υμάντα μεταφοράς και να καλύπτει όλο το μήκος του για να έχει πλήρης ακρίβεια ανίχνευσης. Ο αισθητήρας δε μπορεί να προστεθεί από το model browser και το προσθέτουμε πατώντας δεξά κλικ στο χώρο της προσομοίωσης και επιλέγοντας **Add>Proximity Sensor>Ray type**.



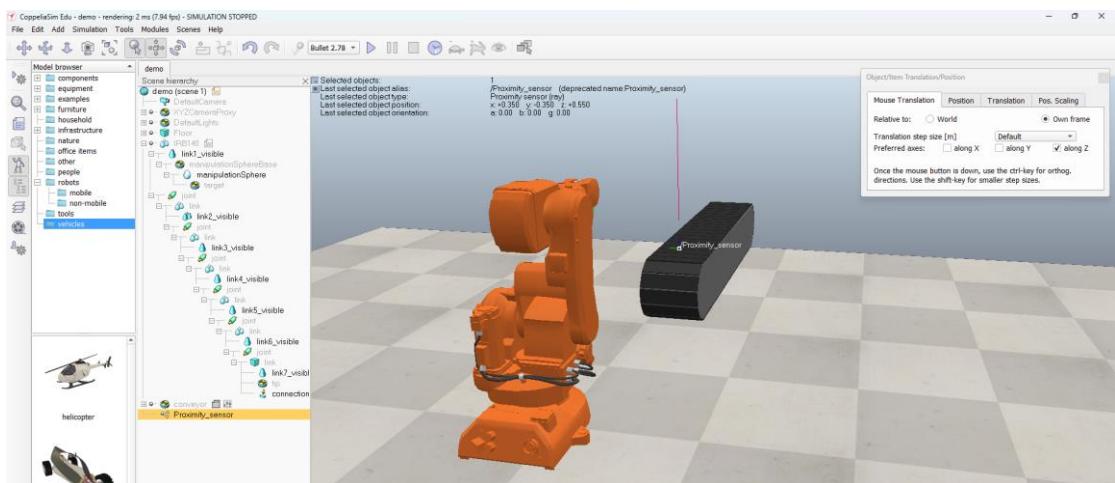
Εικόνα 4 Προσθήκη αισθητήρα ανίχνευσης χρώματος

Μόλις δημιουργηθεί ο αισθητήρας, θα τον επιλέξουμε από το «Scene hierarchy» και θα τον μετακινήσουμε με το κουμπί «object/item shift» πρώτα κάτω από τον ιμάντα σε σημείο που θα μπορεί να σταματήσει το κουτάκι έγκαιρα και για να έχει πρόσβαση ο ρομποτικός βραχίονας.



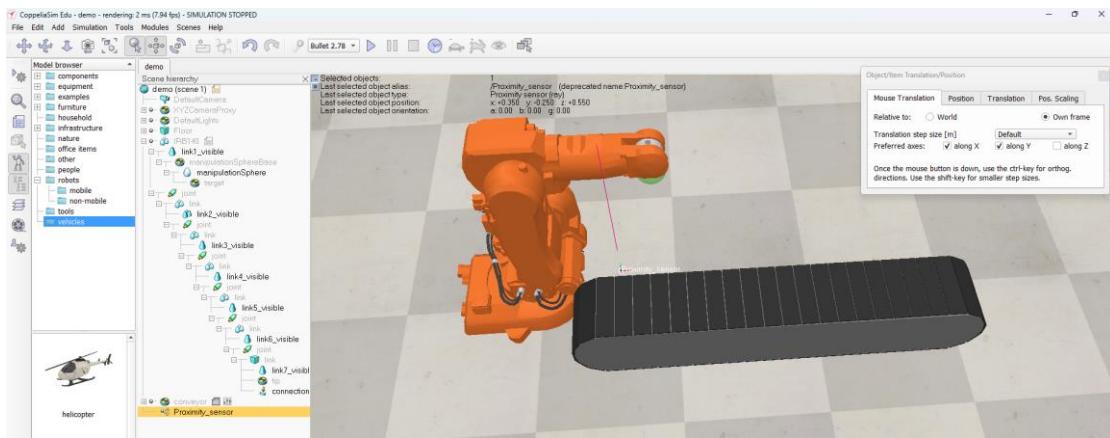
Εικόνα 5 Τοποθέτηση αισθητήρα ανίχνευσης χρώματος κάτω από τον ιμάντα μεταφοράς

Στη συνέχεια, τον μετακινούμε πάνω από τον ιμάντα με το ίδιο κουμπί, πατώντας την καρτέλα «Mouse Translation» και έχοντας επιλεγμένα μόνο τα «Own frame» και «along Z».



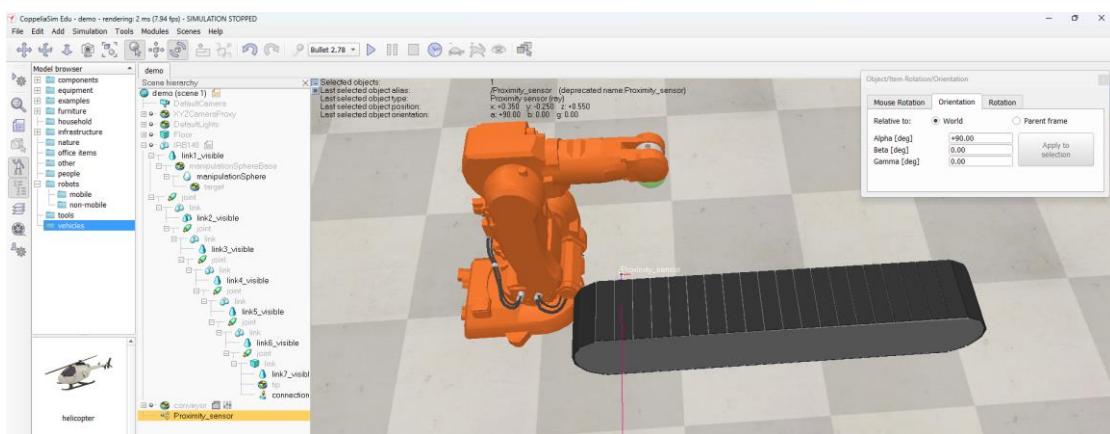
Εικόνα 6 Τοποθέτηση αισθητήρα ανίχνευσης χρώματος πάνω στον ιμάντα μεταφοράς

Παρατηρείται ότι ο αισθητήρας αποτελείται από ένα κυκλικό αντικείμενο, από το οποίο πιάνουμε τον αισθητήρα και μια ευθεία, η οποία ουσιαστικά είναι ο ίδιος ο αισθητήρας, δηλαδή από αυτήν γίνεται η ανίχνευση χρώματος. Γι' αυτό το λόγο, προσαρμόζουμε τη θέση του αισθητήρα (μόνο επιλεγμένα «along X», «along Y»), ώστε να είναι έξω από τον υμάντα το κυκλικό αντικείμενο και μέσα η ευθεία να καλύπτει την επιφάνεια του υμάντα.



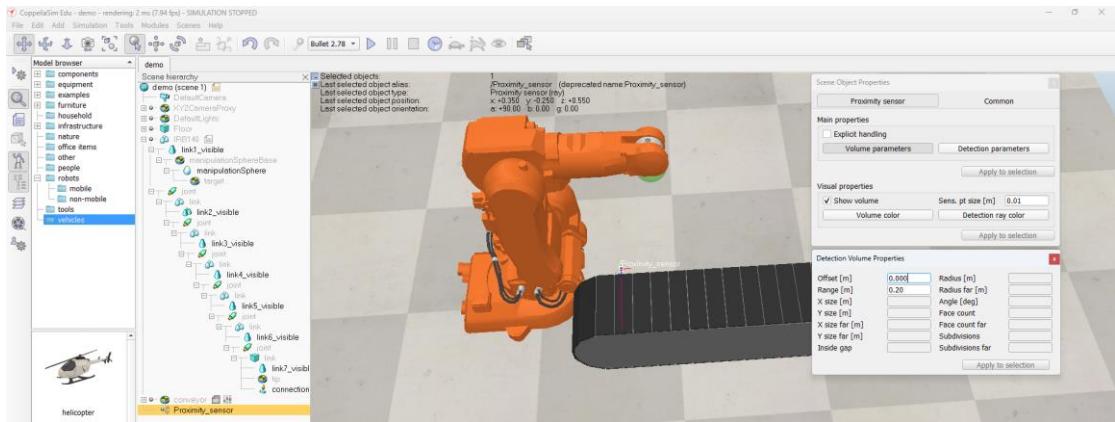
Εικόνα 7 Προσαρμογή θέσης αισθητήρα ανίχνευσης χρώματος

Με τον τρόπο που είναι τοποθετημένος ο αισθητήρας δε μπορεί να ανιχνεύσει τίποτα, γιατί «κοιτάει» στον αέρα και όχι στον υμάντα. Για να το πετύχουμε αυτό, πρέπει να τον περιστρέψουμε με το κουμπί «object/item rotate», πατώντας την καρτέλα «Orientation» με επιλεγμένο το «World» στο «Alpha [deg]» ίσο με +90.00 και τα υπόλοιπα 0.00.



Εικόνα 8 Προσαρμογή κατεύθυνσης αισθητήρα ανίχνευσης χρώματος

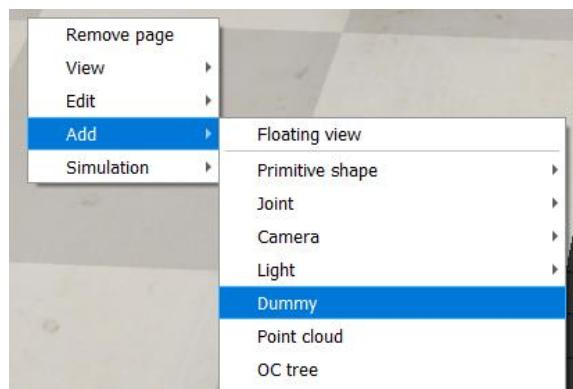
Τέλος, πρέπει η ευθεία του αισθητήρα να καλύπτει όλο το μήκος του ιμάντα και μόνο και γι' αυτό πατάμε διπλό κλικ στο εικονίδιο του «Proximity_sensor» στο «Scene hierarchy» για να δούμε τις ιδιότητες του. Πατάμε «Volume parameters» και ορίζουμε «Offset» ίσο με 0 και «Range» ίσο με 0.20 που είναι το ιδανικό.



Εικόνα 9 Προσαρμογή εμβέλειας αισθητήρα ανίχνευσης χρώματος

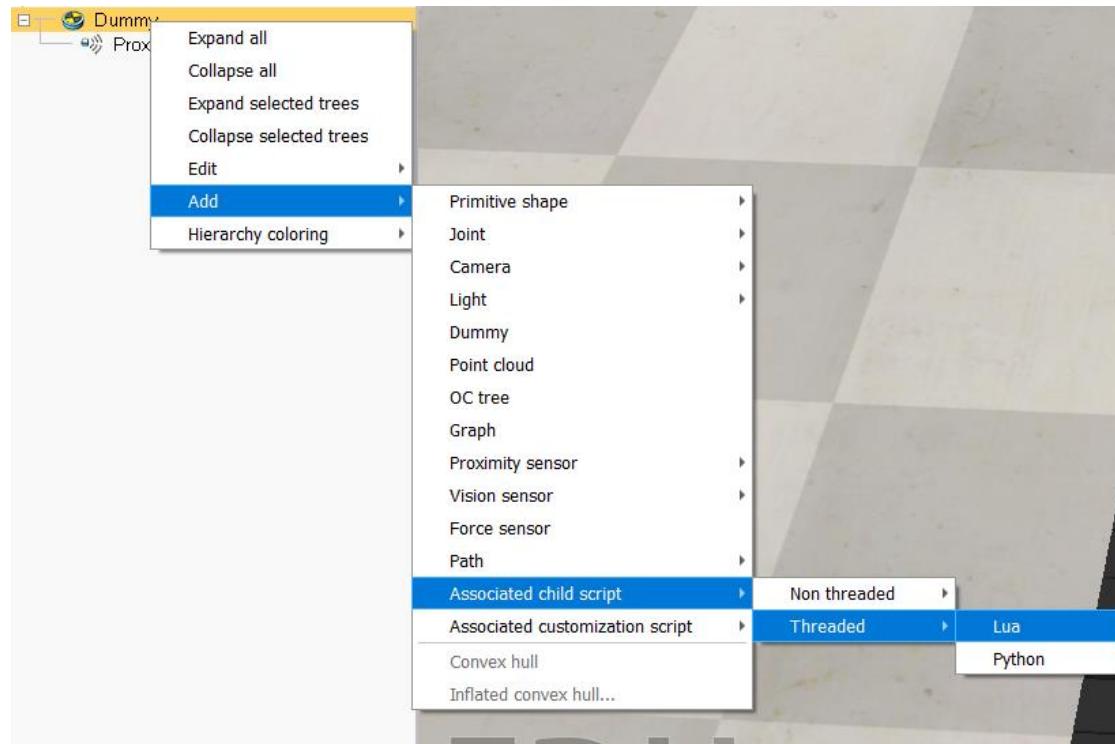
Οι συντεταγμένες του αισθητήρα ανίχνευσης χρώματος είναι (**x=+0.175, y=+0.525, z=+0.550**).

Ένα ακόμα βήμα που χρειάζεται να γίνει για ένα σκοπό που θα αναλυθεί σε επόμενο κεφάλαιο, είναι να προσθέσουμε ένα εικονικό αντικείμενο «Dummy» για να βάλουμε μέσα τον αισθητήρα. Αυτό θα μας βοηθήσει να γράψουμε κώδικα σε γλώσσα προγραμματισμού Lua, για να ενημερώνουμε το ρομποτικό βραχίονα για το χρώμα του κουτιού. Προσθέτουμε το «Dummy» με παρόμοιο τρόπο, όπως κάναμε με τον αισθητήρα, δηλαδή δεξί κλικ και **Add>Dummy**.



Εικόνα 10 Προσθήκη εικονικού αντικειμένου

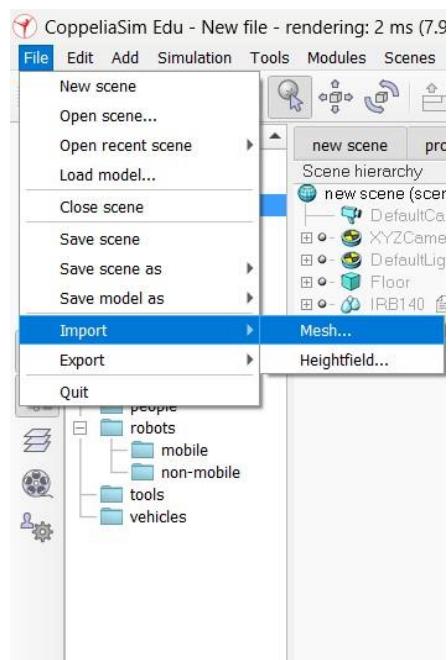
Στη συνέχεια, από το «Scene hierarchy», σέρνουμε το «Proximity_sensor» μέσα στο «Dummy» και κάνουμε δεξί κλικ στο «Dummy» και πατάμε **Add>Associated child script>Threaded>Lua** για να γράψουμε μετά τον κώδικα. Πατώντας διπλό κλικ στο «Dummy» ως λέξη, αλλάζουμε το όνομα του σε «conveyor_sensor».



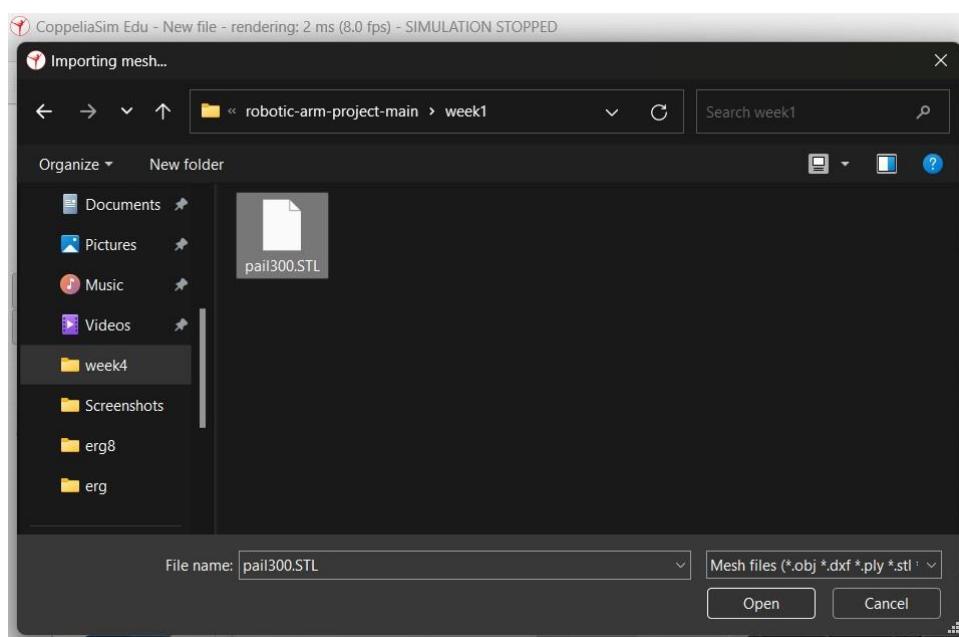
Εικόνα 11 Προσαρμογή εικονικού αντικειμένου

Χρωματισμένοι κάδοι:

Οι χρωματισμένοι κάδοι είναι αυτοί στους οποίους θα αποθηκεύονται τα κουτάκια ανάλογα το χρώμα τους. Οι κάδοι που χρησιμοποιήθηκαν δεν περιλαμβάνονται στο model browser, καθώς έχουν δημιουργηθεί χειροκίνητα και έχουν μορφή αρχείου .stl που πρέπει να κατεβαστεί από αυτόν τον σύνδεσμο: [2]. Για να εισάγουμε αυτό το αρχείο, πατάμε στο μενού **File>Import>Mesh...**, το επιλέγουμε και μετά «Import».



Εικόνα 12 Προσθήκη αρχείου κάδον (1)



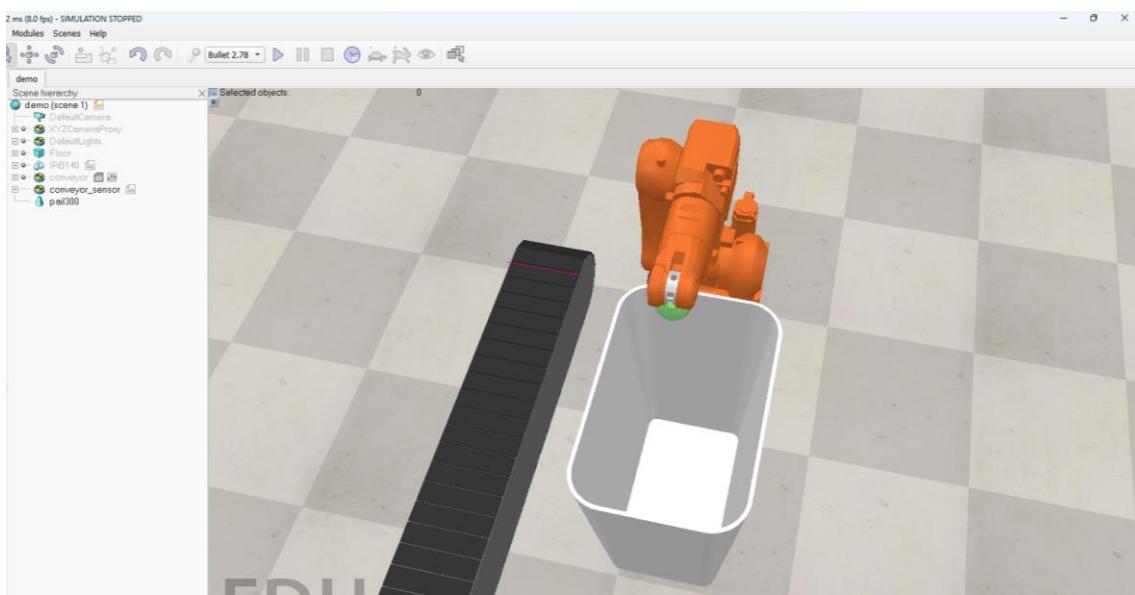
Εικόνα 13 Προσθήκη αρχείου κάδον (2)

Μόλις εισαχθεί στο πρόγραμμα, θα πρέπει να εμφανιστεί, όπως στην *Εικόνα 14*. Όπως αναφέρθηκε, οι κάδοι πρέπει να είναι μπροστά από τον ρομποτικό βραχίονα, οπότε μετακινούμε αντίστοιχα με το κουμπί «object/item shift» και περιστρέφουμε με το κουμπί «object/item rotate», ώστε να είναι κάθετοι.



Εικόνα 14 Εμφάνιση κάδου

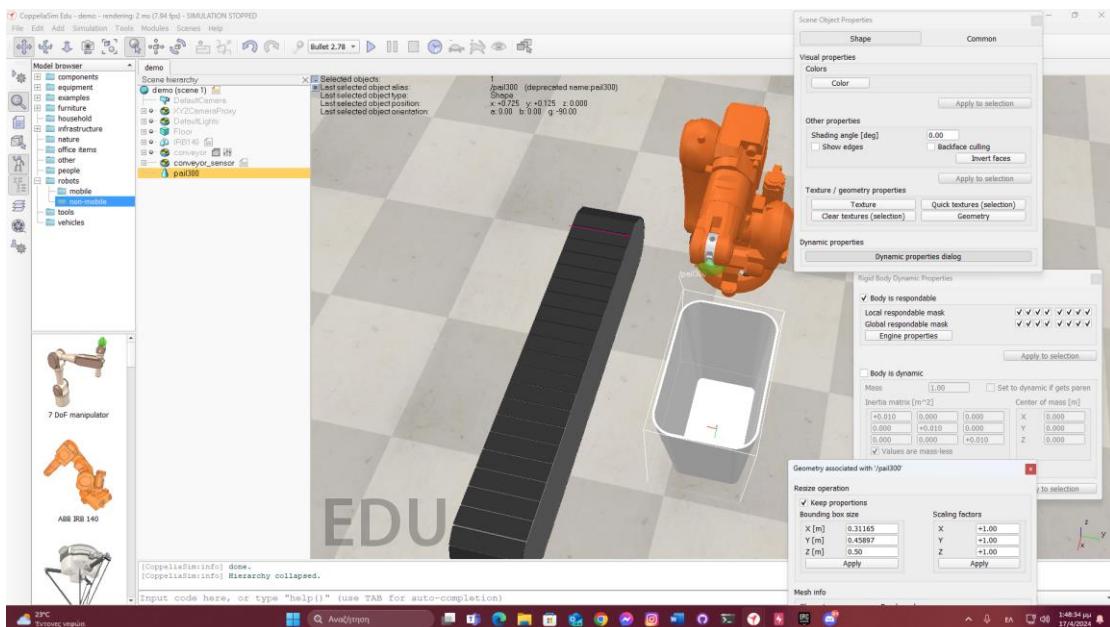
Οι συντεταγμένες του κάδου είναι (**x=+0.35, y=+0.725, z=0.00**). Ένας άλλος τρόπος περιστροφής του είναι με το κουμπί «object/item rotate» στην καρτέλα «Orientation» με επιλεγμένο το «World» στο «Gamma [deg]» ίσο με -90.00 και τα υπόλοιπα 0.00.



Εικόνα 15 Μετακίνηση κάδου

Συνολικά θα τοποθετηθούν 3 κάδοι, κόκκινος, πράσινος και μπλε. Σε αυτό το σημείο, πρέπει να ορίσουμε ειδικές ρυθμίσεις πριν βάλουμε χρώμα, όπως είναι το μέγεθος. Κάνοντας διπλό κλικ στο εικονίδιο του κάδου από το «Scene hierarchy» (pail300), μας εμφανίζονται οι ιδιότητες του. Έτσι, ορίζουμε δύο ρυθμίσεις:

1. **Dynamic properties dialog>Body is respondable (μόνο αυτό τσεκαρισμένο)**
2. **Geometry>X [m]=0.31165, Y [m]=0.45897, Z [m]=0.50>Apply (οι τιμές καθορίζονται με βάση τις δικές σας ανάγκες)**



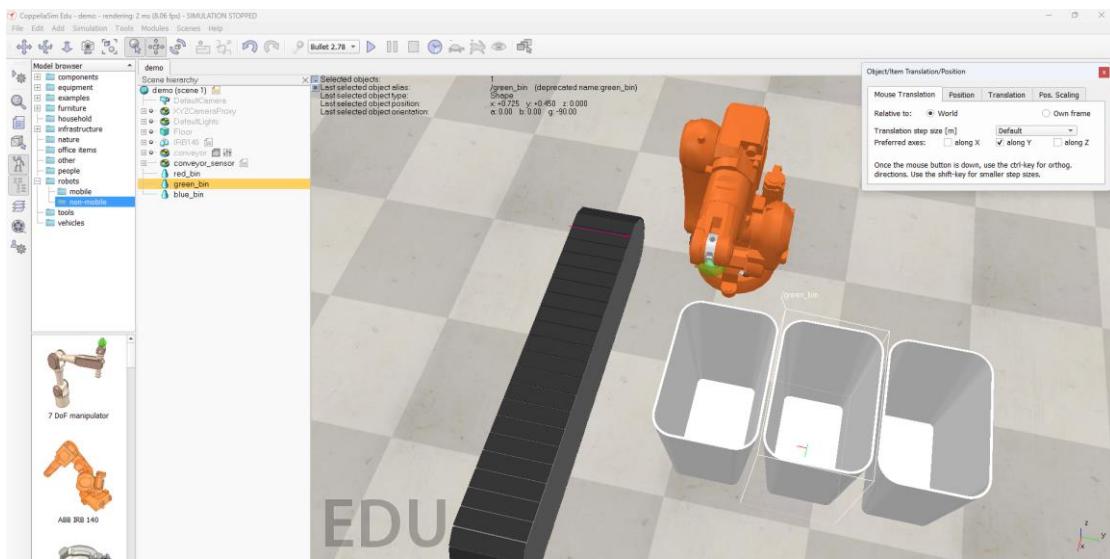
Εικόνα 16 Ειδικές ρυθμίσεις κάδου

Εφόσον έχουν εφαρμοστεί οι παραπάνω ρυθμίσεις, μπορούμε να αντιγράψουμε τον ίδιο κάδο με αυτές τις ρυθμίσεις για να δημιουργήσουμε τους άλλους δύο. Κάνουμε Ctrl+C και Ctrl+V το pail300 μέσα στο «Scene hierarchy» και ταυτόχρονα βάζουμε και ονόματα στο κάθε κάδο με τον πρώτο κάδο που προσθέσαμε να είναι ο κόκκινος.



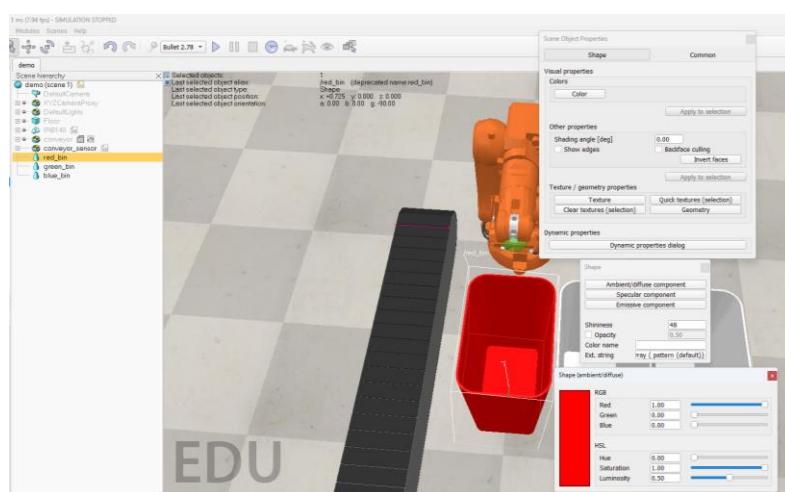
Εικόνα 17 Ονόματα κάδων

Μετακινούμε κάθε κάδο εκτός του κόκκινου (red_bin) κατά βήμα 0.325 μόνο ως προς το Y. Για παράδειγμα, αν οι συντεταγμένες του κόκκινου κάδου είναι (**x=+0.35, y=+0.725, z=0.00**), τότε του πράσινου θα είναι (**x=+0.35, y=+1.050, z=0.00**) κοκ.



Εικόνα 18 Μετακίνηση κάδων

Τέλος, πρέπει κάθε κάδος να έχει το χρώμα που του αρμόζει ανάλογα με το όνομα του. Για να ορίσουμε στον πρώτο κάδο το χρώμα κόκκινο, πατάμε διπλό κλικ στο εικονίδιο του «red_bin» και **Color>Ambient diffuse component>Red=1.00, Green=0.00, Blue=0.00** και αντίστοιχα για τους άλλους κάδους.



Εικόνα 19 Χρωματισμός κάδου

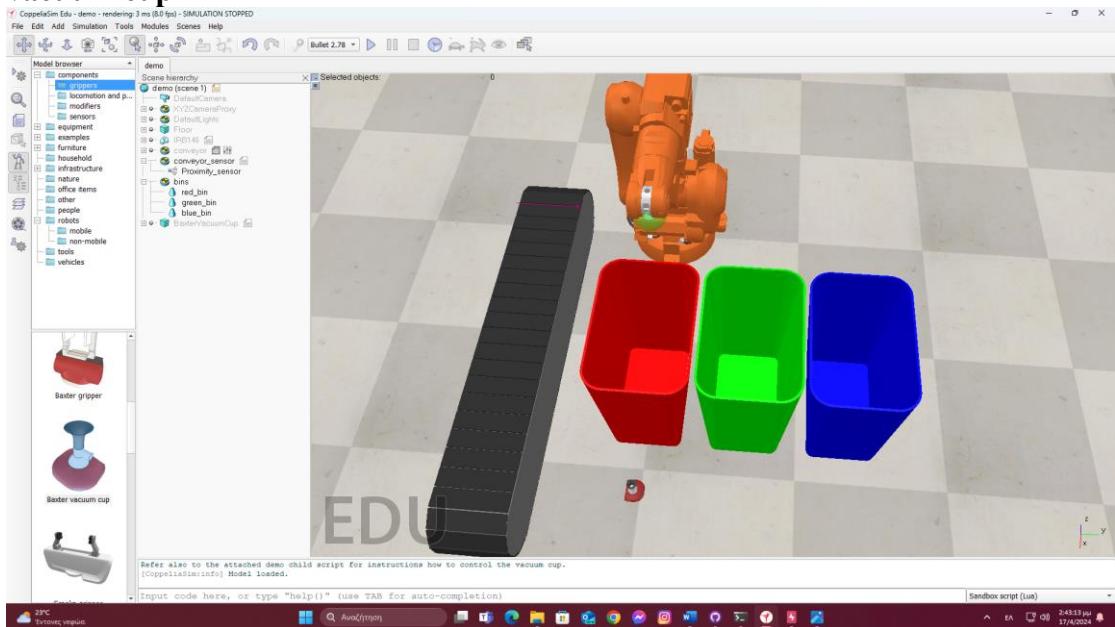
Κάτι ακόμα που πρέπει να κάνουμε για να έχουμε συγκεντρωμένους τους κάδους στο «Scene hierarchy» είναι να προσθέσουμε ένα απλό «Dummy», πατώντας **Add>Dummy**, να το ονομάσουμε «bins» και να σύρουμε μέσα σε αυτό τους κάδους.



Εικόνα 20 Συγκέντρωση κάδων

Baxter Vacuum Cup - βεντούζα:

Η βεντούζα που θα χρησιμοποιήσουμε είναι για να «κολλάνε» υποθετικά τα κουτάκια, όταν τα παίρνει ο ρομποτικός βραχίονας. Σε περιβάλλον προσομοίωσης, δε μας ενδιαφέρει, όπως στην πραγματικότητα, αν πιάνει σωστά τα κουτάκια από πάνω ή από το πλάι κτλ. Προσθέτουμε την βεντούζα μέσα από το model browser, πηγαίνοντας στο **Model browser>components>grippers** και επιλέγοντας το «**Baxter vacuum cup**».



Εικόνα 21 Προσθήκη βεντούζας

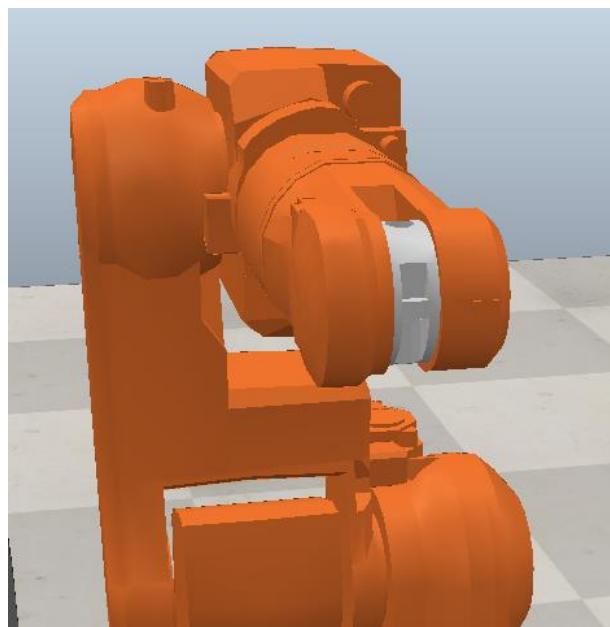
Αρχικά, πρέπει να ενσωματωθεί η βεντούζα στην άκρη από την οποία θα «πιάνει» ο ρομποτικός βραχίονας. Αυτό είναι ένα πολύ κρίσιμο σημείο και συνίσταται να γίνει πρώτα **Ctrl+S** για να αποθηκεύσετε τις αλλαγές σας. Βήματα:

1. Μετακίνηση του target στο manipulationSphereBase και διαγραφή του manipulationSphere.
2. Διαγραφή του link7_visible και του connection που περιέχει το link.
3. Μετακίνηση του BaxterVacuumCup στο link6_visible και μετακίνηση του link6_visible στο τελευταίο link.



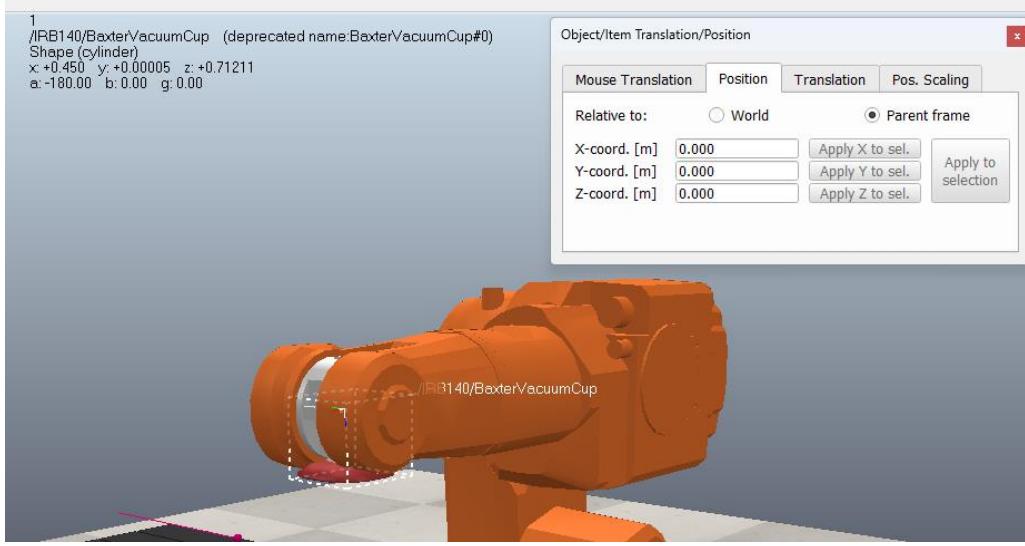
Εικόνα 22 Πριν τις ρυθμίσεις

Εικόνα 23 Μετά τις ρυθμίσεις



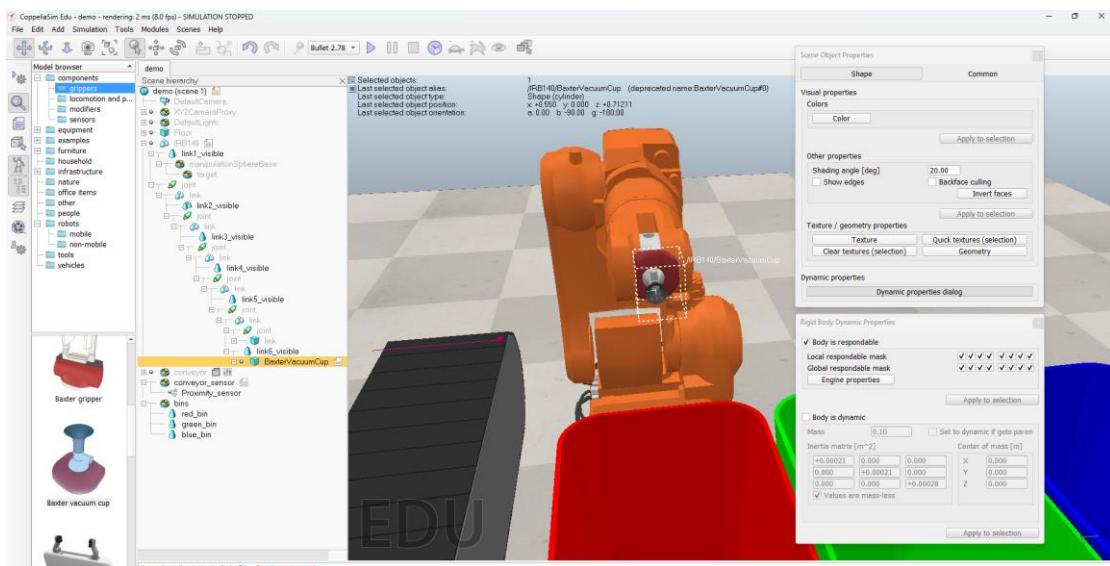
Εικόνα 24 Αποτέλεσμα ρυθμίσεων

Τέλος, πρέπει να τοποθετήσουμε την βεντούζα στην άκρη του ρομποτικού βραχίονα. Πρέπει να μετακινηθεί η βεντούζα πάνω στο μεσαίο σημείο που υπάρχει κενό στην προηγούμενη εικόνα με το μαύρο μέρος της βεντούζας να «κοιτάει» προς τα έξω. Για να το κατευθύνουμε να πάει στο «link6_visible» που το βάλαμε, πρέπει με το κουμπί «object/item shift» στην καρτέλα Position με επιλεγμένο «Parent frame» να είναι όλα 0.00.



Εικόνα 25 Μετακίνηση βεντούζας

Οι συντεταγμένες της βεντούζας είναι ($x=+0.275$, $y=+0.77505$, $z=+0.71211$) με επιλεγμένο το «World» και με περιστροφή (κουμπί «object/item rotate») στην καρτέλα «Orientation» με επιλεγμένο το «World», το «Alpha [deg]» ίσο με 0.00, το «Beta [deg]» ίσο με -90.00 και «Gamma [deg]» ίσο με +180.00. Μετακινήστε όσο χρειάζεται με τον άξονα X για να βγει η άκρη της βεντούζας προς τα έξω και να έχει την παρακάτω μορφή. Ένα τελευταίο βήμα είναι να ξετσεκάρουμε το «Body is dynamic» από τις ιδιότητες τις βεντούζας για να μην πέσει.

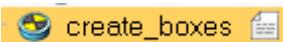


Εικόνα 26 Προσαρμογή θέσης βεντούζας

Κεφάλαιο 3

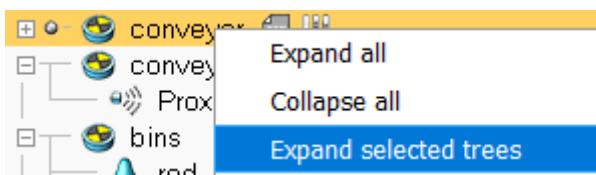
Κώδικας αυτόματης δημιουργίας αντικειμένων με τυχαίο χρώμα

Ο τρόπος με τον οποίο θα δημιουργούνται τα κουτάκια με χρώμα θα είναι αυτόματος μέσω κώδικα με την γλώσσα προγραμματισμού Lua. Όπως και τις προηγούμενες φορές, δημιουργούμε ένα απλό «Dummy» και κάνουμε δεξί κλικ στο «Dummy» και πατάμε **Add>Associated child script>Threaded>Lua**. Δίνουμε στο «Dummy» το όνομα «create_boxes» και πατάμε δεξιά στο άσπρο πλαίσιο που δημιουργήθηκε για να γράψουμε τον κώδικα.



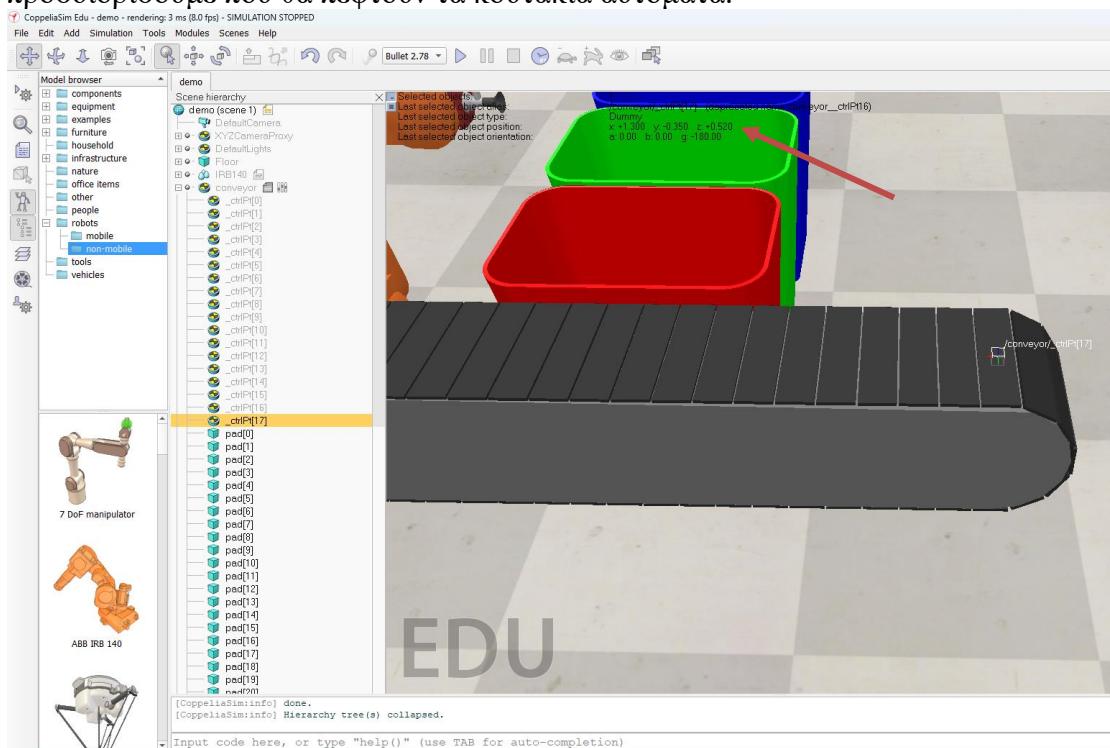
Εικόνα 27 Εμφάνιση εικονικού αντικειμένου «create_boxes»

Πριν γράψουμε τον κώδικα, πρέπει να πατήσουμε δεξί κλικ στο «conveyor» και να πατήσουμε «Expand selected trees», ώστε να δούμε όλη τη δομή του.



Εικόνα 28 Αποκάλυψη δομής ιμάντα μεταφοράς

Αυτό που πρέπει να κάνουμε τώρα είναι να επιλέξουμε το τελευταίο «_ctrlPt» για να βρούμε τις συντεταγμένες του (τις σημειώνουμε για τον κώδικα – το κόκκινο βέλος) και να το αλλάξουμε όνομα σε «ctrlPt». Στην εικόνα, αλλάζουμε το όνομα του «_ctrlPt[17]». Ο λόγος που θέλουμε να πάρουμε αυτές τις συντεταγμένες είναι για να προσδιορίσουμε που θα πέφτουν τα κουτάκια αυτόματα.



Εικόνα 29 Καταγραφή σημείου δημιουργίας κουτιών

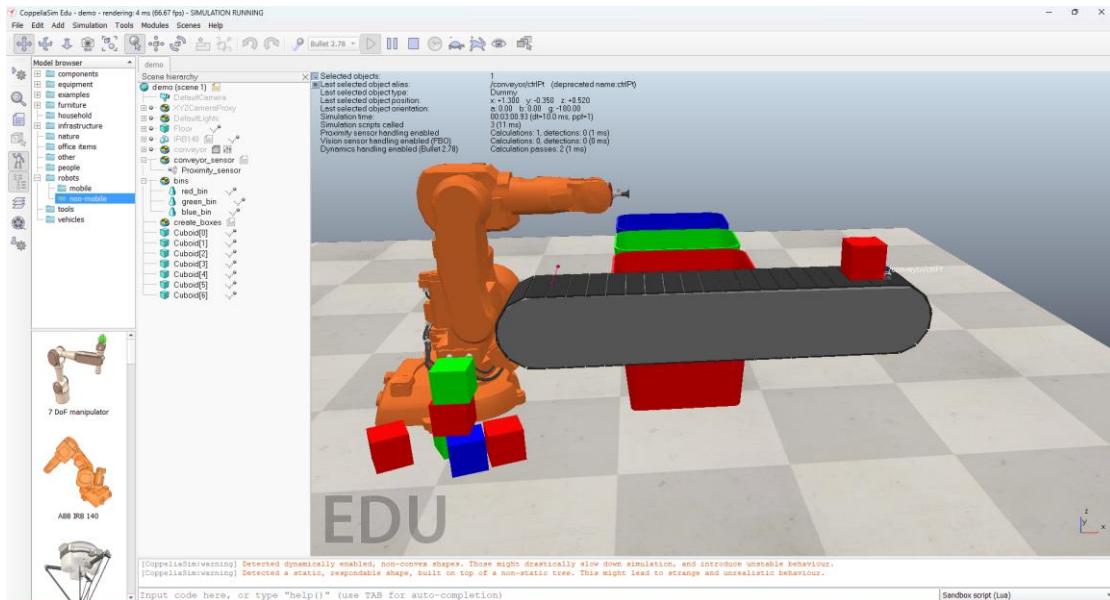
```

Child script "/create_boxes"
-- lua
1
2
3 function sysCall_init()
4     sim = require('sim')
5
6     -- Put some initialization code here
7     ctrlPt = sim.getObjectHandle('ctrlPt')
8     -- sim.setStepping(true) -- enabling stepping mode
9 end
10
11 function sysCall_thread()
12     -- Put your main code here, e.g.:
13
14
15
16     -- orismos ton xromaton
17     local colors = {
18         {1.0, 0.0, 0.0, 1.0}, -- kokkino
19         {0.0, 1.0, 0.0, 1.0}, -- prasino
20         {0.0, 0.0, 1.0, 1.0} -- mple
21     }
22
23
24
25     while sim.getSimulationState() ~= sim.simulation_advancing_aboutostop do
26
27         local colorIndex = math.random(1, #colors) -- Generate random index
28         local color = colors[colorIndex] -- Access random color from the array
29         local box = sim.createPseudoShape(0, 16, {0.1, 0.1, 0.1}, 0.01, color)
30
31         sim.setObjectPosition(box, -1, {1.1, 0.425, 0.59})
32         sim.setShapeColor(box,nil,sim.colorcomponent_ambient,color)
33         sim.setObjectInt32Parameter(box,sim.shapeintparam_static,0)
34         sim.setObjectInt32Parameter(box,3004,1)
35         sim.setObjectSpecialProperty(box,sim.objectspecialproperty_collidable+sim.objectspecialproperty_measurable+sim.objectspecialproperty_detectable_all)
36         sim.wait(30)
37
38     end
39 end

```

Εικόνα 30 Κώδικας *create_boxes*

Με λίγα λόγια, έχουμε ορίσει τον πίνακα των χρωμάτων και με έναν ατέρμων βρόγχο, δημιουργούνται τα κουτάκια με το χρώμα που επιλέχθηκε τυχαία (με διαστάσεις που έχουμε ορίσει – 0.1, 0.1, 0.1) στις συντεταγμένες (**x=1.1, y=0.425, z=0.59**) όπου ξεκινάει ο υμάντας ανά 30 δευτερόλεπτα. Η τιμή που έχει το z είναι αυτή που καταγράψαμε +0.07 για να πέφτει το κουτάκι από πάνω. Επίσης, στο τέλος ορίζουμε και ειδικές ιδιότητες για το κάθε αντικείμενο/κουτάκι, δηλαδή ότι το καθορισμένο αντικείμενο είναι επιλέξιμο (collidable), μετρήσιμο (measurable) και μπορεί να ανιχνευτεί (detectable) από οποιοδήποτε άλλο αντικείμενο στο περιβάλλον της προσομοίωσης [1].



Εικόνα 31 Παράδειγμα εκτέλεσης προσομοίωσης

Κεφάλαιο 4

Κώδικας διαχείρισης αισθητήρα χρώματος στον ιμάντα μεταφοράς

Ο τρόπος με τον οποίο θα ενημερώνεται ο ρομποτικός βραχίονας για το χρώμα του αντικειμένου γίνεται με βάση τον παρακάτω κώδικα. Πατώντας το εικονίδιο στο dummy «conveyor_sensor» από το «Scene hierarchy», μπορούμε να διαχειριστούμε τον κώδικα.

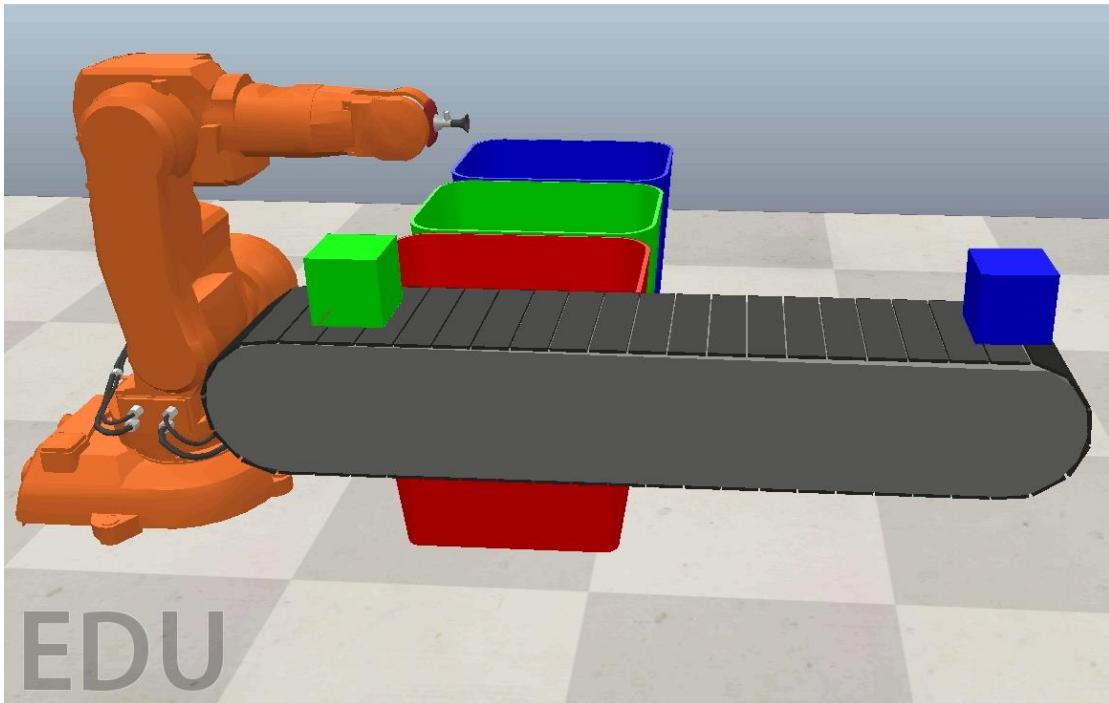
```

1 -- lua
2
3 function sysCall_init()
4     sim = require('sim')
5
6     -- Put some initialization code here
7     Proximity_sensor= sim.getObjectHandle('Proximity_sensor')
8     conveyor= sim.getObjectHandle('conveyor')
9
10    sim.setInt32Signal('boxColor',1)
11    sim.setInt32Signal('detectedBox',0)
12    -- sim.setStepping(true) -- enabling stepping mode
13 end
14
15 function sysCall_thread()
16     -- Put your main code here, e.g.:
17     while true do
18         result,sensor_dist,detectedPoint,detectedObjectHandle=sim.readProximitySensor(Proximity_sensor)
19         if result>0 then -- if sensor detect the box
20             -- stop the conveyor
21             --sim.writeCustomDataBlock(conveyor,'CONVMOV',sim.packTable({vel=0}))
22             sim.writeCustomTableData(conveyor,'_ctrl__',{vel=0.0})
23             sim.setInt32Signal('detectedBox',1)
24             sim.setInt32Signal('detectedObjectHandle',detectedObjectHandle)
25
26             -- get the detected box color and implement it to the 'boxcolor' signal
27             r,rgbData=sim.getShapeColor(detectedObjectHandle,nil,sim.colorcomponent_ambient_diffuse)
28             if rgbData[1] == 1 then sim.setInt32Signal('boxColor',1)
29             elseif rgbData[2] == 1 then sim.setInt32Signal('boxColor',2)
30             else sim.setInt32Signal('boxColor',3)
31             end
32             else
33                 --restart the conveyor
34                 --sim.writeCustomDataBlock(conveyor,'CONVMOV',sim.packTable({vel=0.1}))
35                 sim.writeCustomTableData(conveyor,'_ctrl__',{vel=0.05})
36                 sim.setInt32Signal('detectedBox',0)
37                 sim.setInt32Signal('detectedObjectHandle',-1)
38             end
39         end
40     end
41 end
42

```

Εικόνα 32 Κώδικας ιμάντα μεταφοράς

Με λίγα λόγια, ελέγχει αν ο αισθητήρας έχει ανιχνεύσει κάτι. Αν ναι, σταματά την ταινία (conveyor), καθορίζει το χρώμα του κουτιού (rgbData[1]=κόκκινο, ..[2]= μπλε, ..[3]=πράσινο) και ενημερώνει κάποια σήματα. Αν δεν ανιχνεύσει κάτι, επανεκκινεί τον ιμάντα μεταφοράς και επαναφέρει τις σηματοδοτήσεις. Αξίζει να σημειωθεί ότι η αρχική ταχύτητα που ορίστηκε στην ταινία ήταν 0.1 m/s όπου δεν άφηνε αρκετό χρονικό διάστημα για τον αισθητήρα να ανιχνεύσει το κουτάκι, με αποτέλεσμα να φεύγει. Γι' αυτό το λόγο, η ταχύτητα είναι πλέον 0.05m/s με χρόνο αναμονής 30 δευτερόλεπτα ανάμεσα στα κουτάκια.



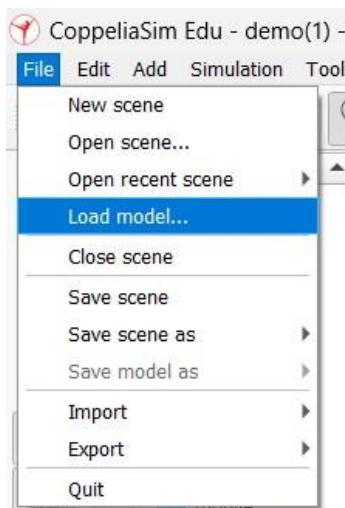
Εικόνα 33 Παράδειγμα εκτέλεσης προσομοίωσης

Παρατηρούμε ότι έχει ανιχνευτεί το πράσινο κουτάκι και περιμένει μέχρι να το πάρει ο ρομποτικός βραχίονας (σε επόμενο κεφάλαιο). Κάθε 30 δευτερόλεπτα, δημιουργείται νέο κουτάκι (όπως το μπλε που απεικονίζεται) και περιμένει και εκείνο μέχρι να μην ανιχνεύει τίποτα ο αισθητήρας χρώματος, ώστε να αρχίσει να λειτουργεί και πάλι ο ιμάντας και να κινείται.

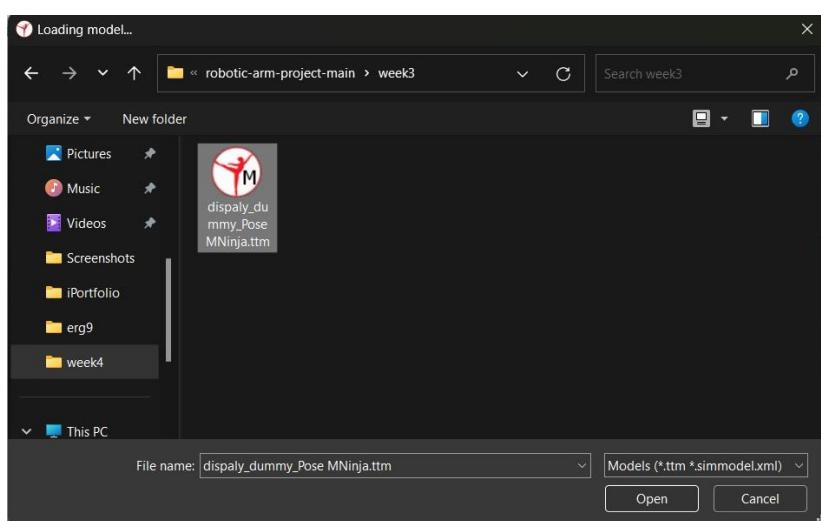
Κεφάλαιο 5

Διαχείριση κίνησης του ρομποτικού βραχίονα

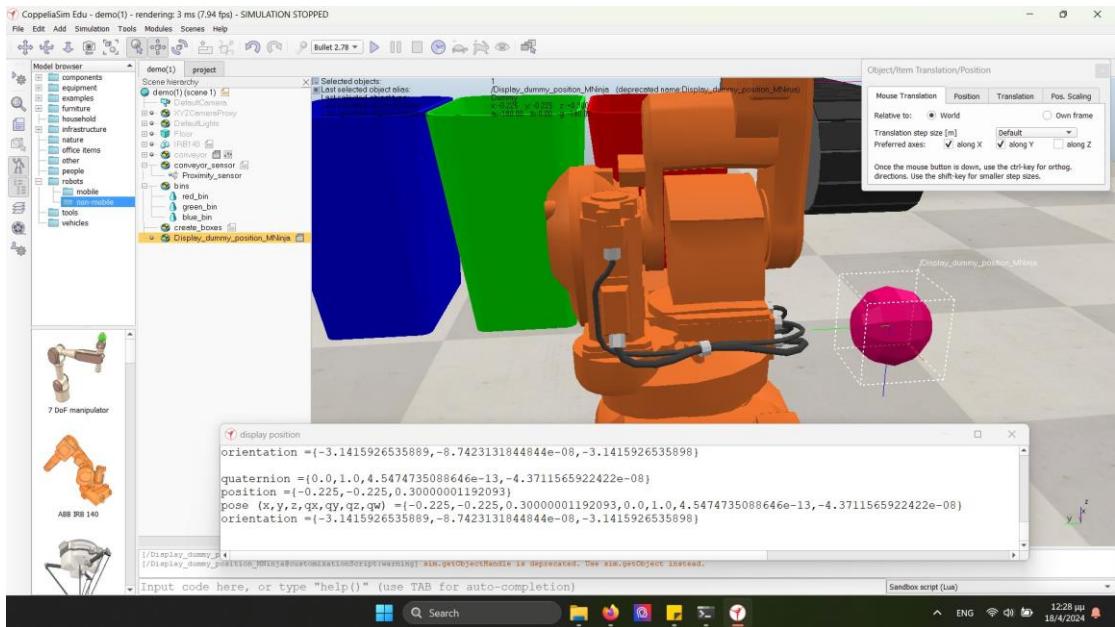
Η δυσκολία όλης της άσκησης βρίσκεται στον τρόπο κίνησης του ρομποτικού βραχίονα, ώστε να ταξινομήσει τα έγχρωμα αντικείμενα στον κατάλληλο κάδο. Με τη βοήθεια του εργαλείου «display_position» (σύνδεσμος για κατέβασμα [3]) προσδιορίζονται συντεταγμένες (x,y,z) με μεγάλη ακρίβεια, ώστε να καθορίσουμε τις φάσεις της ταξινόμησης. Ουσιαστικά, αυτό το εργαλείο δίνει τις συντεταγμένες της θέσης του (quaternion και position) για να ορίσουμε στον κώδικα σε ποια θέση θέλουμε το ρομπότ να μετακινηθεί. Για να το εισάγουμε, πατάμε από το μενού File>Load model... και το επιλέγουμε.



Εικόνα 34 Εισαγωγή display_position (1)



Εικόνα 35 Εισαγωγή display_position (2)



Εικόνα 36 Εμφάνιση display_position (Display_dummy_position_MNinja)

Φάσεις ταξινόμησης:

1. Το ρομπότ μετακινείται πάνω από τη θέση που πρέπει να παραλάβει ένα αντικείμενο.
2. Αναμένεται να ανιχνευθεί ένα αντικείμενο στον ιμάντα μεταφοράς.
3. Ο ρομποτικός βραχίονας παραλαμβάνει το ανιχνευμένο αντικείμενο.
4. Το ανιχνευμένο αντικείμενο παραλαμβάνεται από το ρομπότ.
5. Ο ρομποτικός βραχίονας επιστρέφει στην αρχική θέση.
6. Το ανιχνευμένο αντικείμενο μετακινείται στον σωστό κάδο.
7. Ο ρομποτικός βραχίονας απελευθερώνει το αντικείμενο.
8. Εάν ο κάδος είναι γεμάτος, αδειάζει.

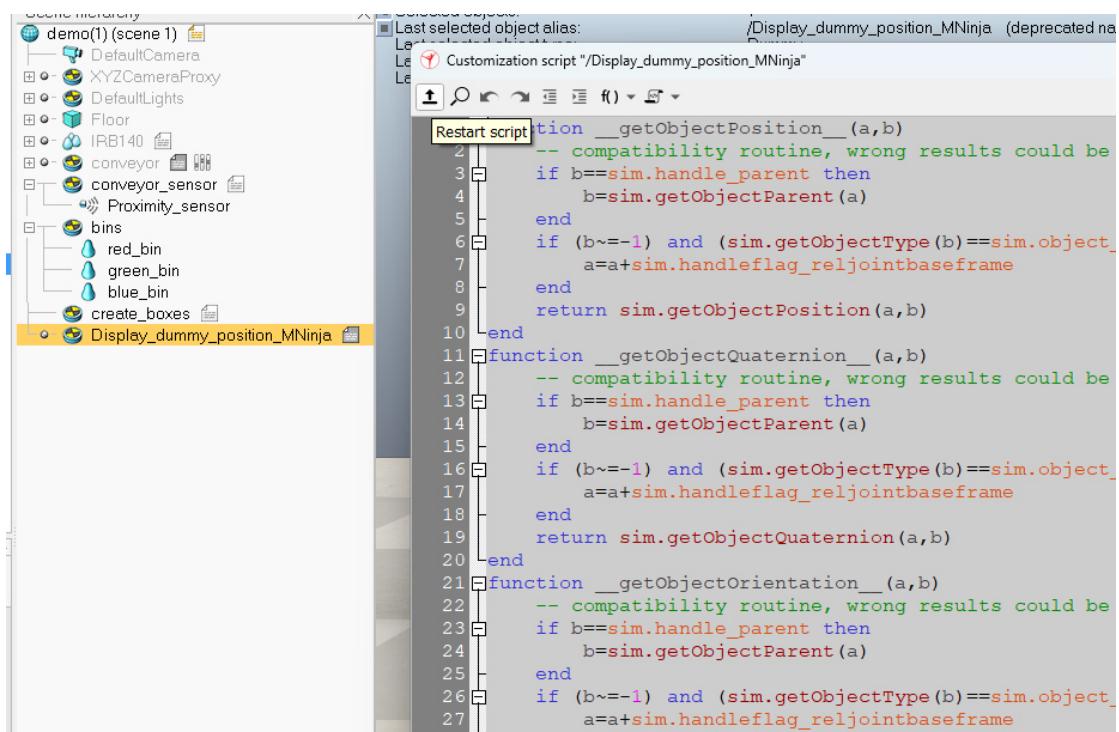
Στις φάσεις 1, 3 και 6 χρησιμοποιούμε το εργαλείο για κάθε περίπτωση:

- a. Πάνω από το κουτάκι (αρχική θέση).
- β. Στο κουτάκι – παραλαβή.
- γ. Στον κάδο – απελευθέρωση.

Αναλυτικά για κάθε βήμα της ταξινόμησης περιγράφονται παρακάτω και ο κώδικας που χρησιμοποιήθηκε, για να γίνουν όλα τα παραπάνω με την αίσθηση του αυτοματισμού με αποτελεσματικό τρόπο.

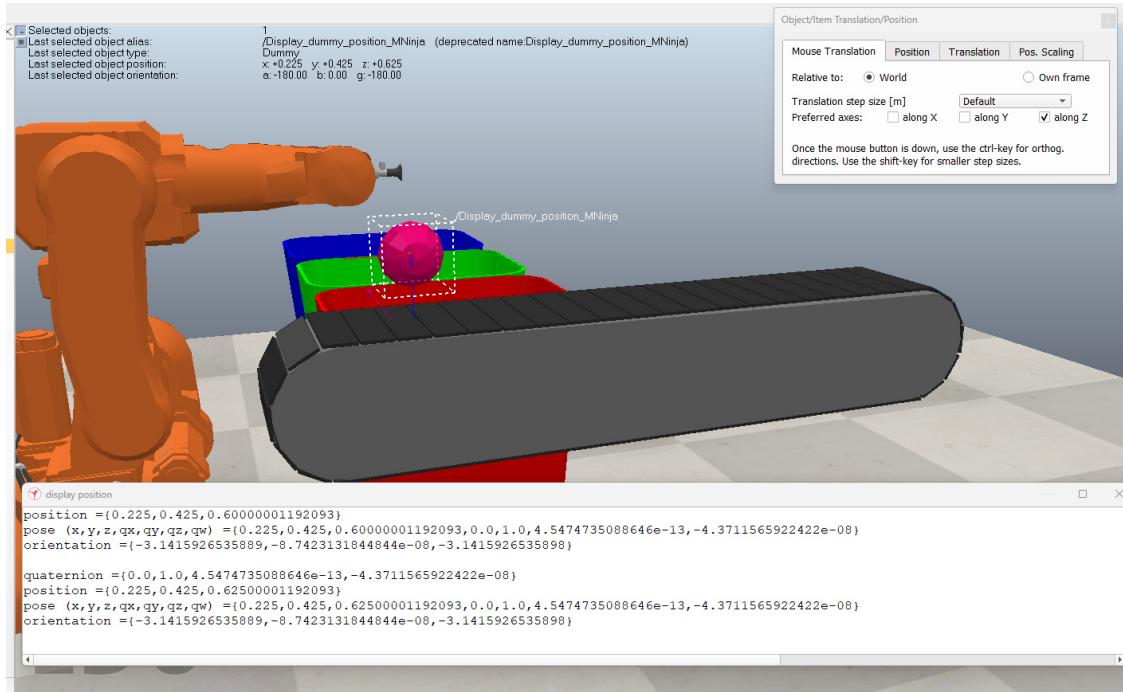
Καταγραφή θέσεων ρομποτικού βραχίονα:

Το «display_position» είναι ένα «Dummy» και μπορεί να μετακινηθεί, όπως μετακινούνται και όλα τα δομικά στοιχεία που προσθέσαμε στο Κεφάλαιο 1 με το κουμπί «object/item shift». Για να εμφανίζονται οι συντεταγμένες όταν το μετακινούμε, πρέπει να πατήσουμε το γκρι πλαίσιο δεξιά του «Display_dummy_position_MNinja», για να δούμε τον κώδικα του και να πατήσουμε το εικονίδιο ή restart script.



Εικόνα 37 Τρόπος εμφάνισης συντεταγμένων

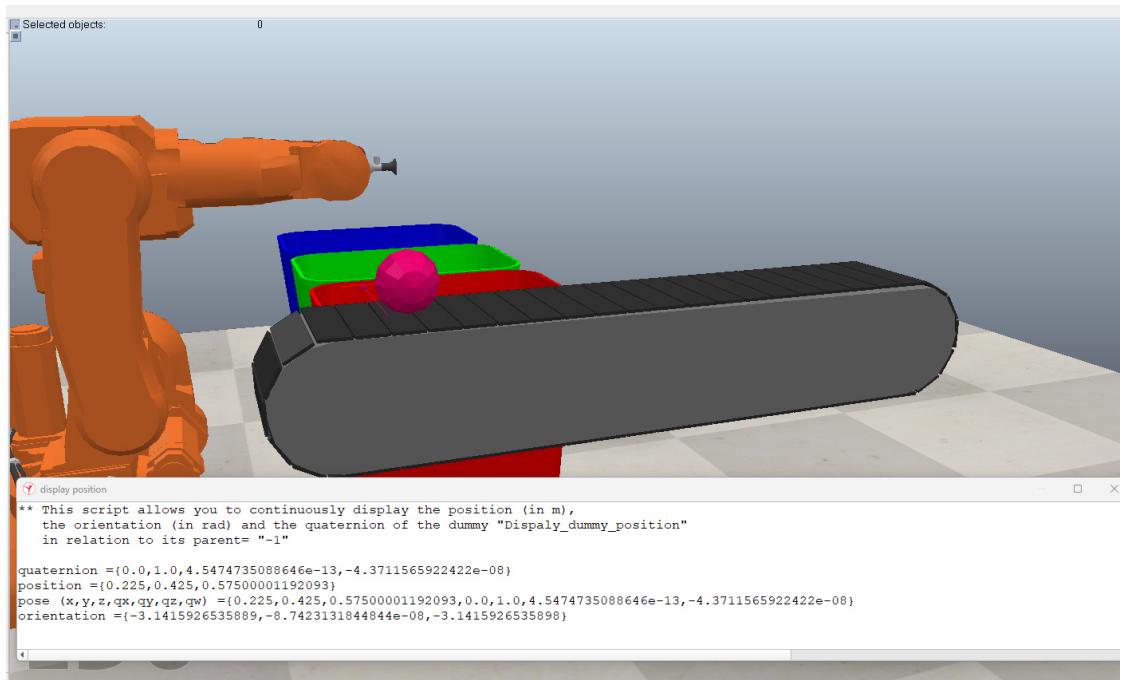
α. Πάνω από το κουτάκι (αρχική θέση).



Εικόνα 38 Αρχική θέση - Κατεύθυνση ρομποτικού βραχίονα προς το κουτάκι

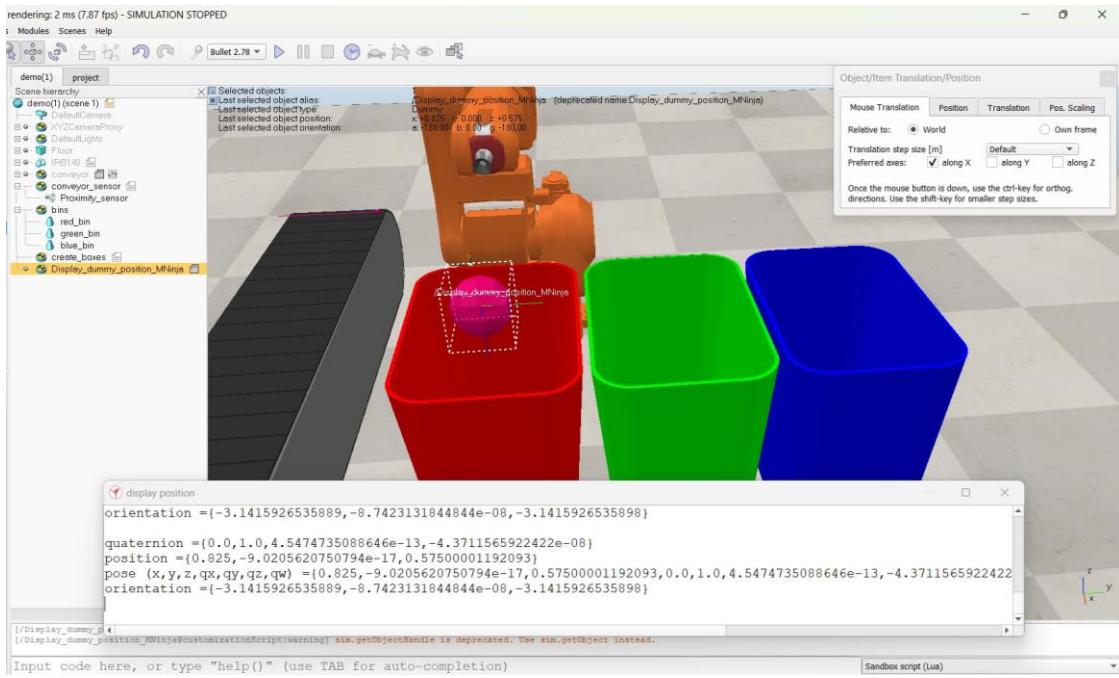
β. Στο κουτάκι – παραλαβή.

Με τη μόνη διαφορά ότι το Ζ είναι -0.05 από την αρχική θέση στο position.



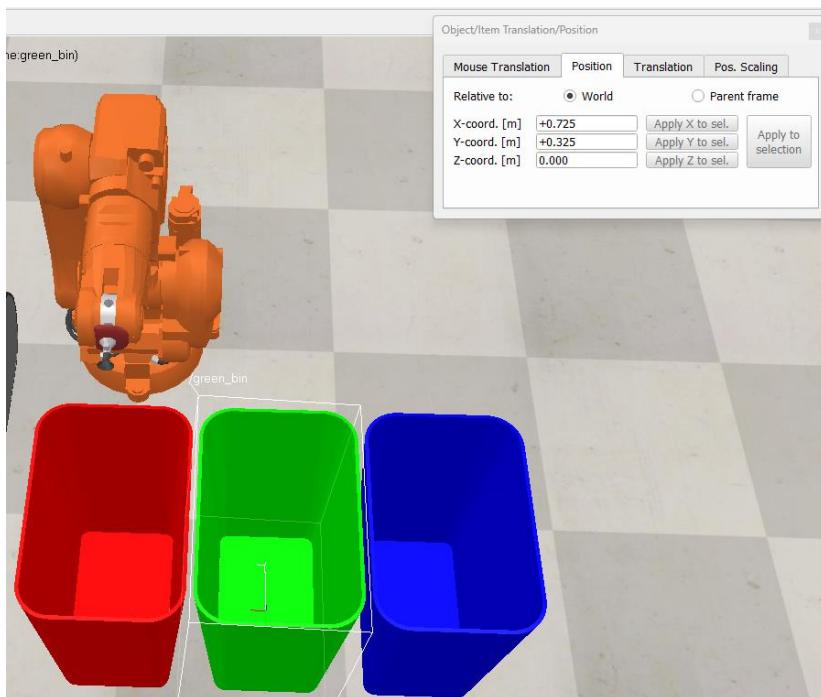
Εικόνα 39 Θέση παραλαβής αντικειμένου

γ. Στον κάδο – απελευθέρωση.



Εικόνα 40 Θέση απελευθέρωσης αντικειμένου στο κέντρο του κάδου

Δε χρειάζεται να καταγράψουμε αυτή τη θέση για κάθε κάδο, καθώς μπορούμε να χρησιμοποιήσουμε τη διαφορά απόστασης μεταξύ τους κατά του άξονα Y («βήμα»), καθώς οι υπόλοιπες συντεταγμένες είναι ίδιες. Αυτό θα χρειαστεί για τον κώδικα. Εφόσον τελειώσουμε με το «display_position», το μετακινούμε στην άκρη και κοιτάμε το «βήμα» μεταξύ του κόκκινου με τον πράσινο κάδο και βλέπουμε ότι είναι 0.325 (**object item/shift>Position**).



Εικόνα 41 Καθορισμός ενδιάμεσης απόστασης κάδων

Με τη βοήθεια του «βήματος», θα ενημερώνουμε τον ρομποτικό βραχίονα πόσο αριστερά του πρέπει να μετακινηθεί για να βρει τον κατάλληλο κάδο, έχοντας ως δεδομένο το χρώμα του αντικειμένου, δηλαδή στην πράξη θα γίνει red=0 (0*0.325+θέση της περίπτωσης γ), green=1 (1*0.325+θέση της περίπτωσης γ) και blue=2 (2*0.325+θέση της περίπτωσης γ). Αυτό μπορεί να φανεί καλύτερα στον παρακάτω κώδικα, εφόσον έχουμε δημιουργήσει ένα νέο «Dummy» με το όνομα «main» (Add>Associated child script>Threaded>Lua).

```

Child script "/main"
1  |-- lua
2
3  function sysCall_init()
4      sim = require('sim')
5
6      -- Put some initialization code here
7
8
9
10     --object handles
11
12     target = sim.getObjectHandle('/IRB140/target')
13     IRB140 = sim.getObjectHandle('/IRB140')
14     tip = sim.getObjectHandle('/IRB140/tip')
15
16
17     -- set up parameters :
18     vel = 0.1
19     accel = 70
20     jerk = 70
21     currentVel={0,0,0}
22     currentAccel={0,0,0}
23     maxVel={vel,vel,vel,vel}
24     maxAccel={accel,accel,accel,accel}
25     maxJerk={jerk,jerk,jerk,jerk}
26     targetVel={0,0,0}
27
28
29     -- sim.setStepping(true) -- enabling stepping mode
30 end
31
32 function sysCall_thread()
33     -- Put your main code here, e.g.:
34     local bins =
35     [1] = {handle = sim.getObjectHandle('/red_bin'), count = 0, name="red"},
36     [2] = {handle = sim.getObjectHandle('/green_bin'), count = 0, name="green"},
37     [3] = {handle = sim.getObjectHandle('/blue_bin'), count = 0, name="blue"}
38
39     local max=2
40
41     --main code
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

Εικόνα 42 Κώδικας διαχείρισης κίνησης ρομποτικού βραχίονα με καθάρισμα κάδων (1)

```

Child script "/main"
1  |-- lua
2
3  function sysCall_init()
4      -- sim.setStepping(true) -- enabling stepping mode
5 end
6
7 function sysCall_thread()
8     -- Put your main code here, e.g.:
9     local bins =
10     [1] = {handle = sim.getObjectHandle('/red_bin'), count = 0, name="red"},
11     [2] = {handle = sim.getObjectHandle('/green_bin'), count = 0, name="green"},
12     [3] = {handle = sim.getObjectHandle('/blue_bin'), count = 0, name="blue"}
13
14     local max=2
15
16     --main code
17     while true do
18
19         --1-- move the robot above the pickup position
20         quaternion1 ={0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
21         position1 ={0.41600000113249,-0.35,0.555000004142523}
22
23         targetPos1={0.391000000113249,-0.35,0.555000004142523,0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
24
25         --moveToPosition rmlMoveToPosition an den doubleuse to apo kato
26         sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position1,quaternion1,targetVel)
27
28
29         --2-- waiting for the sensor to detect the imported object on the conveyor
30         while sim.getInt32Signal('detectedBox')==1 do
31             detectedObjectHandle=sim.getInt32Signal('detectedObjectHandle')
32
33             --3-- pickup the detected object
34             quaternion2 ={0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
35             position2 ={0.41600000113249,-0.35,0.555000004142523}
36
37             targetPos2={0.39100000113249,-0.35,0.555000004142523,0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
38
39             sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position2,quaternion2,targetVel)
40
41             --4-- fake the gripping by making the detected object a child to the robot flange
42             sim.setObjectInt32Param(detectedObjectHandle,3003,1) --static parameter
43             sim.resetDynamicObject(detectedObjectHandle)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

Εικόνα 43 Κώδικας διαχείρισης κίνησης ρομποτικού βραχίονα με καθάρισμα κάδων (2)

```

52
53
54    --2-- waiting for the sensor to detect the imported object on the conveyor
55    while sim.getInt32Signal('detectedBox')~=1 do
56        detectedObjectHandle=sim.getInt32Signal('detectedObjectHandle')
57
58        --3-- pickup the detected object
59        quaternion2 = {0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
60        position2 = {0.41600000113249,-0.35,0.50500004142523}
61
62        targetPos2=(0.39100000113249,-0.35,0.50500004142523,0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08)
63
64        sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position2,quaternion2,targetVel)
65
66        --4-- fake the gripping by making the detected object a child to the robot flange
67        sim.setObjectInt32Param(detectedObjectHandle,3003,1) --static parameter
68        sim.resetDynamicObject(detectedObjectHandle)
69        sim.setObjectParent(detectedObjectHandle,tip,true)
70
71        --counting each bin's boxes
72        local boxColor = sim.getInt32Signal('boxColor')
73        if boxColor == nil and bins[boxColor] == nil then
74            bins[boxColor].count = bins[boxColor].count + 1
75            binHandle=bins[boxColor].handle
76        end
77
78        --5-- return to position1
79        sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position1,quaternion1,targetVel)
80
81        --6-- move this object to the correct bin
82        quaternion3 = {0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
83        position3 = {(0.59100000113249,0.02500000000001+(sim.getInt32Signal('boxColor')-1)*0.325,0.50500004142523)}
84
85        targetPos3=(0.59100000113249,9.9920072216264e-16+(sim.getInt32Signal('boxColor')-1)*0.325,0.50500004142523,0.0,1.0,4.5474735088646e-13,-4.371
86
87        sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position3,quaternion3,targetVel)
88
89        --7-- release the object
90        sim.setObjectInt32Param(detectedObjectHandle,3003,0) --static parameter
91
92        if binHandle == nil then
93            sim.setObjectParent(detectedObjectHandle, binHandle, true)
94        else
95            sim.setObjectParent(detectedObjectHandle, -1, true)
96        end
97
98        --8-- emptying bin if it's full
99        for color, bin in pairs(bins) do
100            if bin.count == max then
101                local binBoxes = sim.getObjectsInTree(bin.handle, sim.object_shape_type, 1)
102                for i = 1, #binBoxes do
103                    sim.removeObject(binBoxes[i])
104                end
105                print("Emptying " .. bin.name .. " bin because it's full.")
106                bin.count = 0
107            end
108        end
109    end
110
111    end
112
113
114    -- See the user manual or the available code snippets for additional callback functions and details
115

```

Εικόνα 44 Κώδικας διαχείρισης κίνησης ρομποτικού βραχίονα με καθάρισμα κάδων (3)

```

76
77
78        --5-- return to position1
79        sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position1,quaternion1,targetVel)
80
81        --6-- move this object to the correct bin
82        quaternion3 = {0.0,1.0,4.5474735088646e-13,-4.3711565922422e-08}
83        position3 = {(0.59100000113249,0.02500000000001+(sim.getInt32Signal('boxColor')-1)*0.325,0.50500004142523)}
84
85        targetPos3=(0.59100000113249,9.9920072216264e-16+(sim.getInt32Signal('boxColor')-1)*0.325,0.50500004142523,0.0,1.0,4.5474735088646e-13,-4.371
86
87        sim.rmlMoveToPosition(target,IRB140,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,position3,quaternion3,targetVel)
88
89        --7-- release the object
90        sim.setObjectInt32Param(detectedObjectHandle,3003,0) --static parameter
91
92        if binHandle == nil then
93            sim.setObjectParent(detectedObjectHandle, binHandle, true)
94        else
95            sim.setObjectParent(detectedObjectHandle, -1, true)
96        end
97
98        --8-- emptying bin if it's full
99        for color, bin in pairs(bins) do
100            if bin.count == max then
101                local binBoxes = sim.getObjectsInTree(bin.handle, sim.object_shape_type, 1)
102                for i = 1, #binBoxes do
103                    sim.removeObject(binBoxes[i])
104                end
105                print("Emptying " .. bin.name .. " bin because it's full.")
106                bin.count = 0
107            end
108        end
109    end
110
111    end
112
113
114    -- See the user manual or the available code snippets for additional callback functions and details
115

```

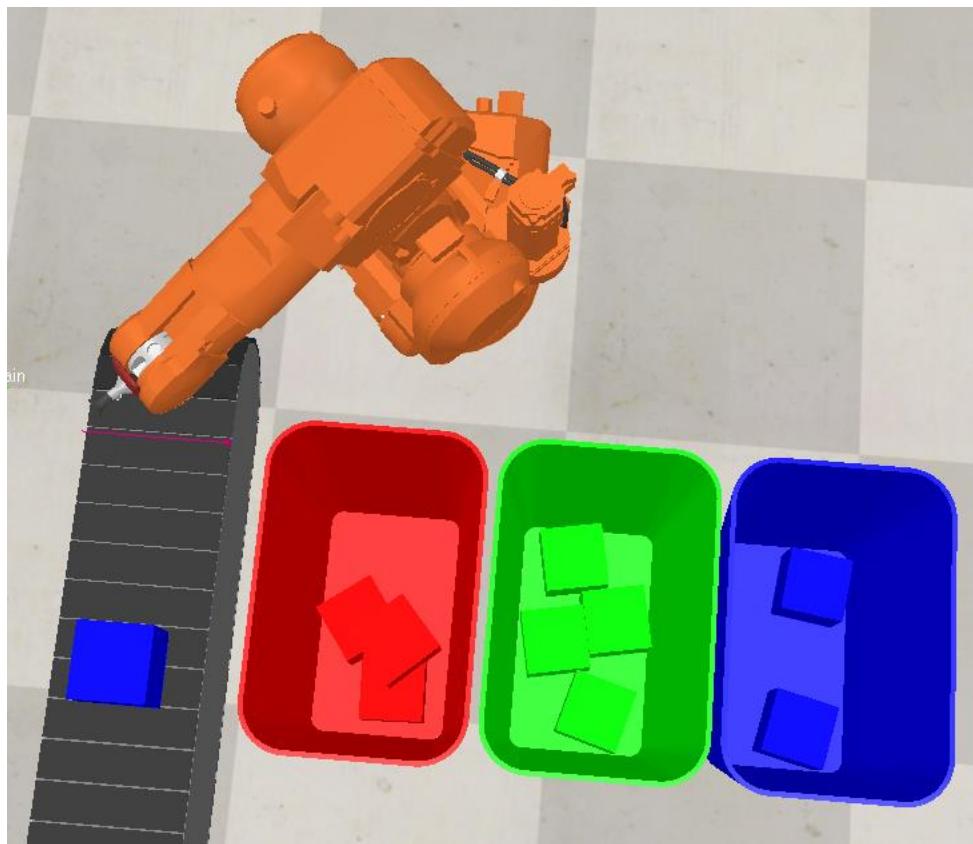
Εικόνα 45 Κώδικας διαχείρισης κίνησης ρομποτικού βραχίονα με καθάρισμα κάδων (4)

Με λίγα λόγια, ο ρομποτικός βραχίονας μετακινείται πρώτα στην αρχική θέση (**περίπτωση a - quaternion1 και position1**) που είναι ακριβώς πάνω από τον ιμάντα. Επίσης, έχει οριστεί ένα ενδεικτικό κατώφλι αδειάσματος ενός γεμάτου κάδου (local max=2) και ένας πίνακας «bins» που περιέχει αντικείμενα για κάθε χρώμα με στοιχεία, όπως η διαχείριση του κάδου (handle), το πλήθος των αντικειμένων κάθε κάδου (count) και το όνομα του (name) για την εμφάνιση κατάλληλων μηνυμάτων, όταν γεμίσει κάποιος από αυτούς.

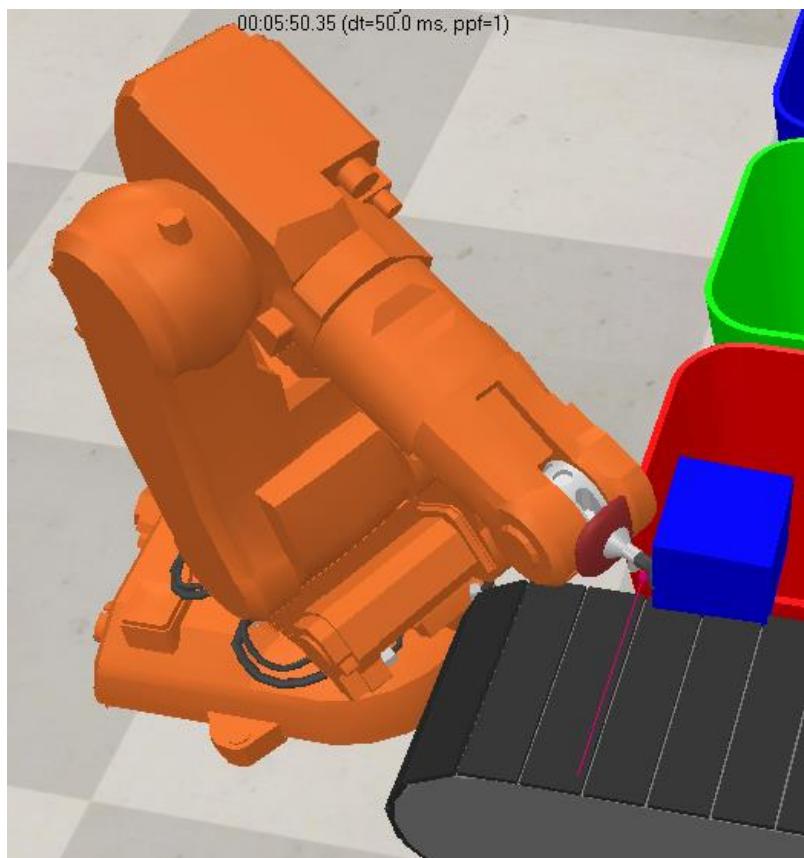
Στη συνέχεια, ελέγχεται η ανίχνευση κάποιου αντικειμένου και αν είναι αληθής, τότε ο ρομποτικός βραχίονας μπαίνει σε θέση παραλαβής (**περίπτωση β - quaternion2 και position2**) σκύβοντας να πάρει το αντικείμενο και το «παίρνει» με υποθετικό τρόπο με την βεντούζα, βάζοντας το child με parent το tip. Αυτό σημαίνει ότι θα είναι κολλημένο στο tip (η άκρη της βεντούζας), ακόμα κι αν είναι πιο μακριά από αυτό. Εφόσον έχει ανιχνευτεί το χρώμα του αντικειμένου, σε αυτό το σημείο πρέπει να αυξηθεί το αντίστοιχο πλήθος αντικειμένων και να γίνει η ταξινόμηση του στον κατάλληλο κάδο (**περίπτωση γ - quaternion3 και position3**). Μια σημαντική λεπτομέρεια που συνέβαλε στο άδειασμα των κάδων είναι ο ορισμός των αντικειμένων τους ως childs σε αυτούς (setObjectParent() [1]), καθώς με αυτόν τον τρόπο μπορέσαμε πολύ απλά να διαγράψουμε τα «κλαδιά» (removeObject()) του «δέντρου» (getObjectsInTree()).

Για να δουλέψει ο κώδικας με τις δικές μας συντεταγμένες και «βήμα», κάνουμε αντικατάσταση των τιμών quaternion1,2,3 και position1,2,3 και το «βήμα» (σημείο κώδικα: sim.getInt32Signal('boxColor')-1)***0.325**) αντίστοιχα. Επίσης, μπορούμε να ορίσουμε δικό μας κατώφλι για το άδειασμα των κάδων για να δούμε πόσα κουτάκια θα χρειαστούν για να γεμίσει μέχρι πάνω, π.χ. local max=50.

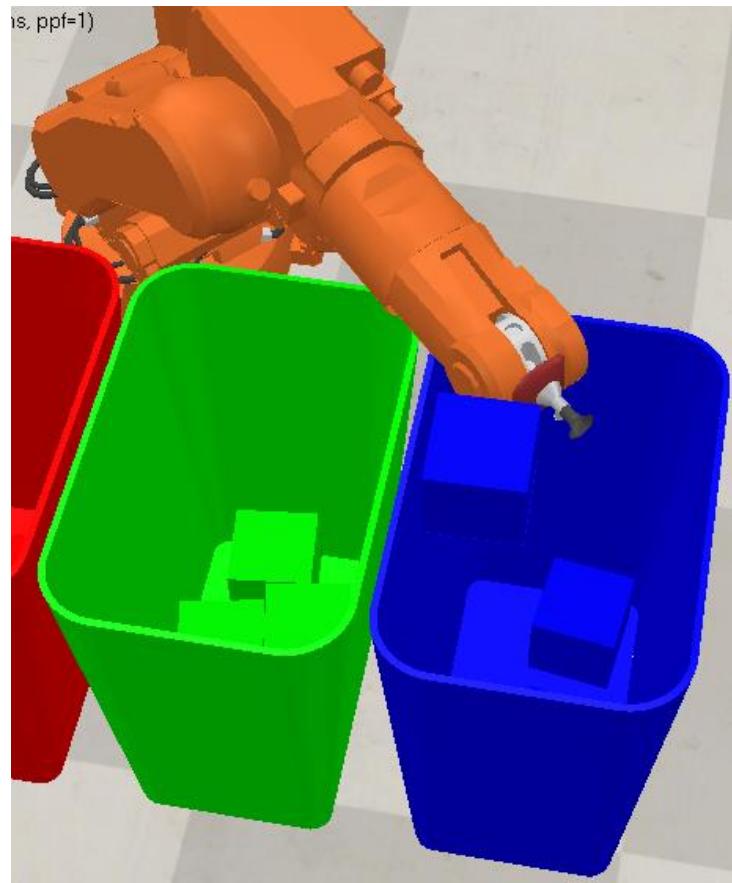
Παράδειγμα εκτέλεσης προσομοίωσης:



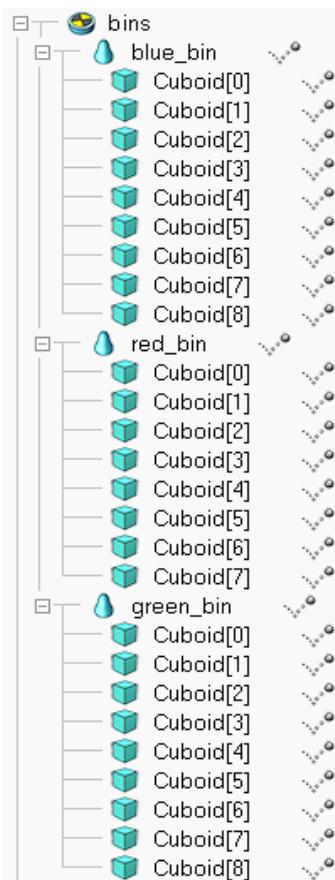
Εικόνα 45 Βραχίονας σε ετοιμότητα



Εικόνα 46 Παραλαβή κοντιού με τη βεντούζα του βραχίονα



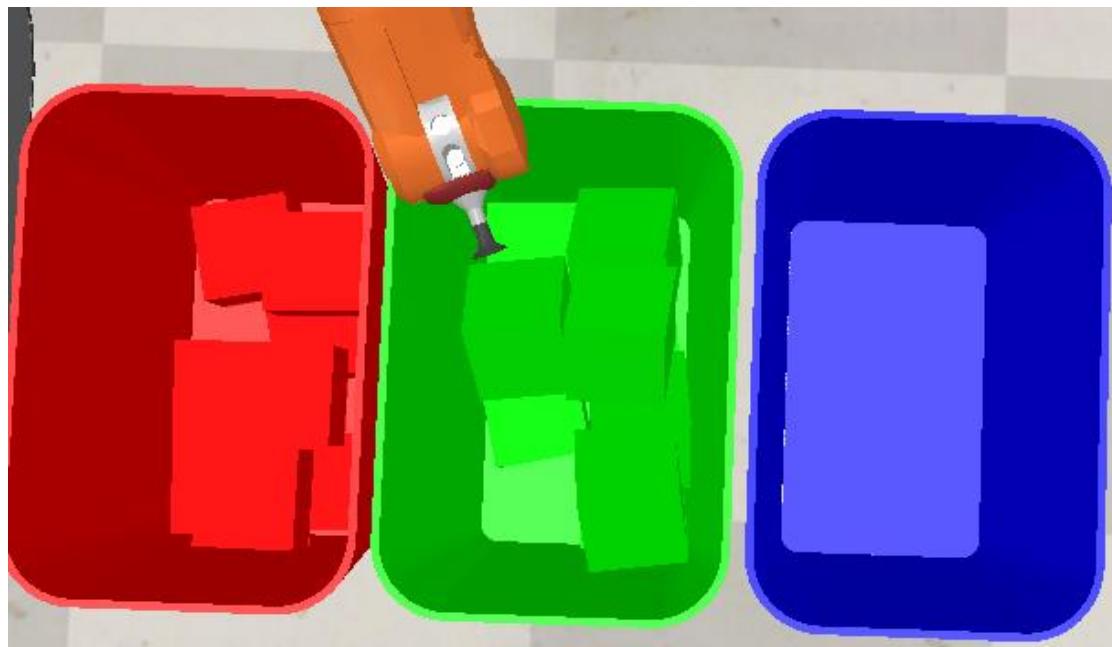
Εικόνα 47 Απομόνωση κουτιού από τη βεντούζα του βραχίονα – Τοποθέτηση κουτιού στον κάδο του χρώματος του



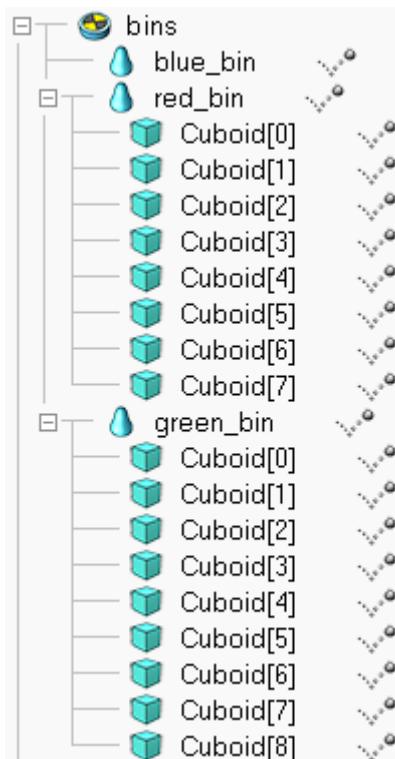
Εικόνα 48 Τρόπος εμφάνισης κατάταξης των κουτιών σε κάθε κάδο

Emptying blue bin because it's full.

Εικόνα 49 Ενημερωτικό μήνυμα όταν αδειάζει γεμάτος κάδος



Εικόνα 50 Άδειασμα του κάδου που γέμισε



Εικόνα 51 Τρόπος εμφάνισης κατάταξης των κοντιών σε κάθε κάδο μετά το άδειασμα

Βιβλιογραφία

- [1] CoppeliaSim - Regular API reference
<https://manual.coppeliarobotics.com/en/apiFunctions.htm>
- [2] Αρχείο 3D κάδου
<https://drive.google.com/file/d/1zD0vsH7t15jEGhuzms557tJR-9FaqMjb/view>
- [3] Display position tool
https://drive.google.com/file/d/1xjCQ6Hkgimtvn9xdH7vlqhXknn_ty3wN/view