ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Συστήματα Αυτοματισμού

Μελέτη τροχιάς ρομπότ σε περιβάλλον VR για συνεργασία ανθρώπου-ρομπότ σε συστήματα κατεργασιών

Μεταπτυχιακή Εργασία

Αγγελίδης Αντώνης

Επιβλέπων: Γ.-Χ. Βοσνιάκος, Καθηγητής

Σχολή Μηχανολόγων Μηχανικών

Τομέας Τεχνολογίας των Κατεργασιών

Αθήνα, Νοέμβριος 2017

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω ιδιαιτέρως τον επιβλέποντα καθηγητή κ. Γ.-Χ. Βοσνιάκο για την άψογη συνεργασία, τις συμβουλές και την υποστήριξη που μου παρείχε καθ' όλη τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Θεωρώ πως η εμπιστοσύνη που έδειξε στο πρόσωπό μου ήταν καταλυτική για την επιτυχή ολοκλήρωση των μεταπτυχιακών μου σπουδών.

Επίσης, θα ήθελα να ευχαριστήσω θερμά τον Δρα Ηλία Μάτσα για την βοήθεια και καθοδήγηση που μου παρείχε στα πρώτα και σημαντικά στάδια της έρευνας μου.

Τέλος, και από τα βάθη της καρδιάς μου, ευχαριστώ για ακόμα μία φορά την οικογένεια μου για την αμέριστη και αδιάκοπη συμπαράσταση που μου παρείχαν καθ' όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Σχημάτων

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι να ορισθεί δυναμικά η τροχιά ρομπότ με ‘έξυπνες’ στρατηγικές, με βάση το που βρίσκεται κάθε χρονική στιγμή ο συνεργαζόμενος άνθρωπος και προς τα πού κινείται, χωρίς την ανάγκη φυσικής υλοποίησης (δηλαδή σε εικονικό περιβάλλον) αλλά διατηρώντας το απρόβλεπτο των ανθρώπινων αντιδράσεων.

Η μετατροπή της λογικής της συνεργασίας ανθρώπου-ρομπότ από προ-ορισμένη σε δυναμική μπορεί να χαρακτηριστεί ως ένα αρκετά δύσκολο αντικείμενο της ρομποτικής και αποτελεί σήμερα μεγάλο πεδίο έρευνας και ανάπτυξης. Η ραγδαία εξέλιξη των υπολογιστικών συστημάτων μας δίνει τη δυνατότητα επεξεργασίας τεράστιου όγκου πληροφοριών για την καθοδήγηση ρομπότ σε πραγματικό χρόνο (real time) και με ταχύτατη απόκριση σε αλλαγές του περιβάλλοντος.

Η συνεργασία ανθρώπου-ρομπότ σε καθήκοντα που εκτυλίσσονται σε βιομηχανικά συστήματα κατεργασιών είναι ακόμη ταμπού, κυρίως για λόγους ασφάλειας του ανθρώπου. Η κατάσταση θα αλλάξει σύντομα, αλλά για να μελετηθούν οι ελεγκτές ρομπότ που θα υλοποιηθούν, ο καλύτερος τρόπος είναι να χρησιμοποιηθεί ένα εικονικό περιβάλλον –συμπεριλαμβανομένου του ρομπότ - μέσα στο οποίο θα λειτουργεί ο άνθρωπος μέσω ενός avatar που αναπαράγει τις κινήσεις του στο πραγματικό περιβάλλον. Το ενδιαφέρον, εν προκειμένω, είναι το πως αλλάζει την τροχιά του το ρομπότ σε πραγματικό χρόνο ώστε να μη συγκρούεται με τον άνθρωπο αλλά και να διατηρείται υψηλή παραγωγικότητα.

Η παρούσα εργασία μπορεί να θεωρηθεί ως εξέλιξη προηγούμενης δουλειάς στο πλαίσιο διδακτορικής διατριβής με βάση την πλατφόρμα ανάπτυξης παιχνιδιών εικονικής πραγματικότητας Unity. Πιο συγκεκριμένα, η αρχική έκδοση του συστήματος που αποτελεί την βάση της παρούσας εργασίας, περιλαμβάνει τα μοντέλα CAD του εικονικού χώρου, του ανθρώπου – avatar και του ρομπότ staubli RX90 για το στρώσιμο ανθρακοϋφασμάτων σε καλούπι. Ο κώδικας που αποτελεί τη λογική των σεναρίου που υλοποιήθηκαν, βασίζονταν σε απλές στρατηγικές ελέγχου του ρομπότ μέσω υπαρχουσών συναρτήσεων (functions) του

Unity ενώ η αλληλουχία κινήσεων ανθρώπου – ρομπότ ήταν προκαθορισμένο από τον χρήστη. Επιπλέον, σε άλλη διπλωματική εργασία, έχει υπολογιστεί το μοντέλο αντίστροφης κινηματικής του συγκεκριμένου ρομπότ, το οποίο χρησιμοποιείται στην παρούσα εργασία. Η νέα λογική ελέγχου του ρομπότ δημιουργείται και υλοποιείται με επέκταση του υπάρχοντος κώδικα σε C#.

Η εργασία χωρίζεται σε έξι θεματικές ενότητες. Πιο συγκεκριμένα:

Στο *κεφάλαιο 1* γίνεται η εισαγωγή στα συνεργατικά συστήματα και κατηγοριοποίηση των τύπων συνεργασίας.

Στο *κεφάλαιο 2* παρουσιάζεται η πλατφόρμα προγραμματισμού και οι υπάρχουσες τεχνικές αποφυγής συγκρούσεων.

Στο *κεφάλαιο 3* γίνεται ο διαχωρισμός του σχεδιασμού διαδρομής και του σχεδιασμού τροχιάς και λεπτομερής περιγραφή του αλγορίθμου RRT που υλοποιήθηκε.

Το *κεφάλαιο 4* αποτελεί την περιγραφή των λειτουργιών του κώδικα που αναπτύχθηκε και των επιλογών που παρέχει στο χρήστη για την παραμετροποίηση του συστήματος.

Στο *κεφάλαιο 5* παρουσιάζονται τα αποτελέσματα του σχεδιασμού κίνησης για αποφυγή εμποδίων από την προσομοίωση διαφορετικών σεναρίων.

Τέλος, στο *κεφάλαιο 6* σχολιάζονται τα αποτελέσματα των προσομοιώσεων, εξάγονται συμπεράσματα και παρατίθενται προτάσεις για μελλοντική έρευνα.

Abstract

In order for robots to operate safely in the presence of humans in their workspace, they need to be able to react to a dynamically changing environment in real-time. This involves both fast processing of sensor data and fast motion planning based on the sensor data.

This work describes the virtual implementation of a motion planner for a 6-DOF Stäubli RX90L robot arm, which is able to react to a dynamically changing environment very quickly. The framework used was built in the well known Unity3D platform in a previous PhD thesis and was extended to behave in an autonomous way with only the start and goal positions of the robot TCP being given. All coding was done in the C# language which is natively supported by the Unity 3D platform. The path palnning algorithm deployed is the Rapidly exploring Random Tree (RRT) which generates a tree and extends it in a random way until one of its branches reaches the goal. The final solution will most propably be far from optimal and for this reason a path-smoothing process using linear interpolation is also deployed after the successful execution of the RRT algorithm. During runtime, continuous obstacle detection algorithms are executed at specified time intervals and define whether the motion planning controller needs to be activated or not. Once a new collision-free path for the TCP of the robot has been generated, all intermediate points of the path are checked in the configuration space of the robot as a part of the trajectory planning controller. At this point, the forward and inverse kinematic model of the robot arm are utilized and a discrete obstacle check (intermediate point interpolation of specific step/resolution) takes place to verify that also the trajectory of all the links of the robot are collision free. The trajectory planner does not generate an optimal trajectory at this program version but calculates all possible joint solutions and detects any collisions on the entire path trajectory.

It is observed with the current implementation, that path calculation times of less than one second (< 1sec) can be reached with the proper parameters set, which is subject to the resolution required in obstacle avoidance. On the other hand, techniques for path planning (e.g. local planning), obstacle avoidance (e.g. lazy collision check) or programming (e.g. multithreading) can decrease the computational load and increase system responsiveness to acceptable levels for a real-time obstacle avoidance collaborative environment (e.g. <200 ms).

Κεφάλαιο 1:Εισαγωγή

1.1 Η εξέλιξη των ρομποτικών συστημάτων παραγωγής

Παρότι οι ρομποτικοί βραχίονες έχουν καθιερωθεί στα βιομηχανικά συστήματα παραγωγής, η λειτουργία τους έχει μείνει απαράλλαχτη και περιορισμένη στην εκτέλεση προκαθορισμένης εργασίας. Η εναλλαγές στον προγραμματισμό τους είναι συνήθως ελάχιστες ενώ η εργασία είναι υποχρεωτικά απομονωμένη από τον άνθρωπο.

Τα τελευταία χρόνια και με στόχο την περαιτέρω βελτίωση της ποιότητας και αύξηση της παραγωγικότητας, έχει εμφανιστεί η ανάγκη αλληλεπίδρασης του ρομπότ με το περιβάλλον του. Πιο συγκεκριμένα, η συνεργασία ανθρώπου –ρομπότ σε κοινό χώρο εργασίας βρίσκεται στο επίκεντρο της έρευνας σύγχρονων συνεργατικών συστημάτων παραγωγής καθώς συνδυάζει την ανθρώπινη γνώση και δεξιότητα με την μηχανική ακρίβεια και επαναληψιμότητα των ρομπότ. Το συνεργατικό μοντέλο εργασίας (collaborative work) ανθρώπου – ρομπότ έχει ήδη εισέλθει στη βιομηχανία σε μικρή, προς το παρόν, κλίμακα δημιουργώντας ακόμα και νέους όρους όπως «cobot» (**c**ollaborative r**obot**).

Η μετάβαση από ένα μοντέλο αντικατάστασης της ανθρώπινης εργασίας σε αυτό της συνεργατικής εργασίας αποτελεί βεβαίως μία πολύπλοκη διαδικασία η οποία κρύβει αρκετά προβλήματα και κινδύνους. Προτού καταφέρουμε να εκμεταλλευτούμε ένα συνεργατικό μοντέλο παραγωγής πρέπει πρώτα τα προβλήματα να προσδιορισθούν επαρκώς και ο κίνδυνος για τον άνθρωπο να εξαλειφθεί. Εμφανίζεται, δηλαδή, η ανάγκη δημιουργίας νέων τεχνικών και κανονισμών, που θα επιτρέπουν την ασφαλή αλληλεπίδραση του ανθρώπου με τα ρομπότ.

1.2 Ορισμός συνεργατικής εργασίας

Με βάση το ISO 10218-1:2011 ορίζεται η *συνεργατική λειτουργία* ως: Η κατάσταση κατά την οποία ειδικά σχεδιασμένα ρομπότ εργάζονται σε άμεση συνεργασία με τον άνθρωπο, μέσα σε έναν ορισμένο χώρο εργασίας. Ο χώρος εργασίας του ρομπότ ορίζεται ως το σύνολο των σημείων στο χώρο τα οποία μπορεί να προσεγγίσει το τελικό σημείο δράσης (end-effector) του ρομπότ.

Ο όρος *συνεργασία* περιγράφει μια κατάσταση η οποία εμπλέκει πολλά συνεργαζόμενα μέρη, με σκοπό την επίτευξη ενός καθήκοντος, ή ενός κοινού στόχου και στην οποία το κάθε μέρος αναλαμβάνει να διεκπεραιώσει διαφορετικά (υπό-)καθήκοντα, με κάποιου είδους αλληλεξάρτηση.

1.2.1 Κατηγοριοποίηση συνεργασίας

Μία προσπάθεια κατηγοριοποίησης των συνεργατικών συστημάτων παραγωγής έχει γίνει με βάση τις εφαρμογές της αυτοκινητοβιομηχανίας. Υπάρχουν εφαρμογές που απαιτούν χειρωνακτική εργασία ενώ άλλες που ο πλήρης αυτοματισμός είναι πιο αποδοτικός. Η συνεργασία ανθρώπου – ρομπότ στην ουσία βρίσκεται κάπου ανάμεσα. Εισάγεται ο όρος «βαθμός συνεργασίας» και διακρίνεται σε χαμηλό, μέσο και υψηλό.

Το περιβάλλον που εξετάζεται στη παρούσα εργασία χαρακτηρίζεται σαν υψηλού βαθμού συνεργασίας. Στην περίπτωση αυτή, το ρομπότ κι ένας ή περισσότεροι χειριστές πραγματοποιούν καθήκοντα ταυτόχρονα ή συνεργατικά, εντός του χώρου εργασίας του ρομπότ, με τα εξής χαρακτηριστικά:

- Το ρομπότ βρίσκεται στον αυτόματο τρόπο λειτουργίας

- Τα συστήματα οδήγησης των σερβοκινητήρων είναι ενεργοποιημένα

- Μπορεί να πραγματοποιηθεί κίνηση του ρομπότ ή του τελικού σημείου δράσης (ΤΣΔ) ενόσω ο άνθρωπος βρίσκεται εντός οποιουδήποτε σημείου του χώρου εργασίας

- Το ρομπότ κινείται στον αυτόματο (προγραμματισμένο) τρόπο λειτουργίας και η ταχύτητα και οι κινήσεις του πρέπει να συγχρονίζονται με αυτές του ανθρώπου, με έναν συνεργατικό τρόπο λειτουργίας για την εκτέλεση του ίδιου καθήκοντος-στόχου.

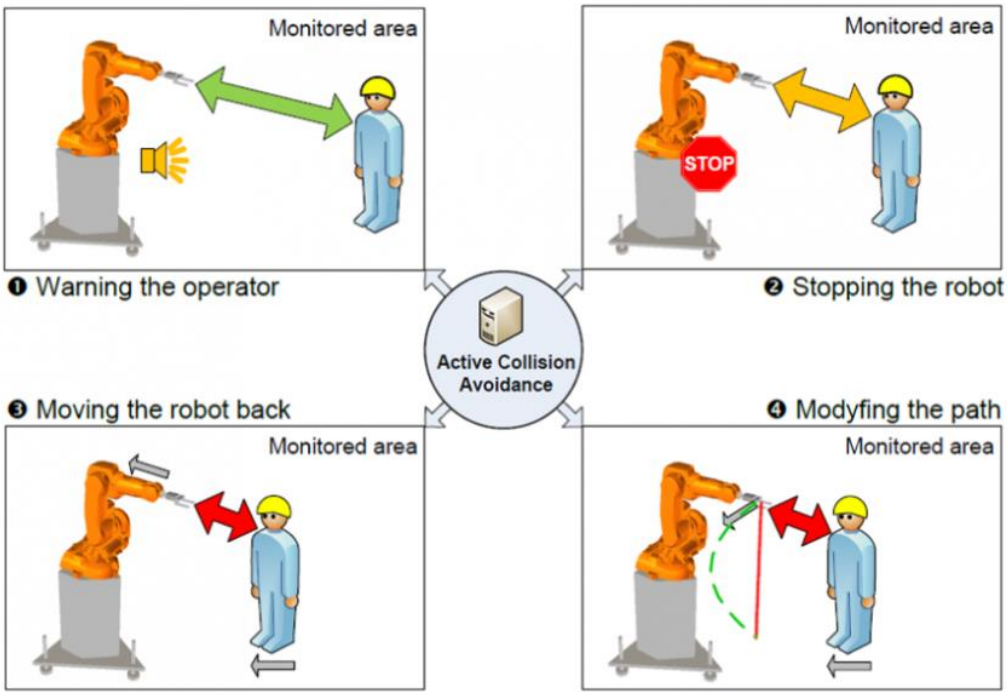
1.2.2 Κίνδυνοι και προβλήματα συνεργασίας

Ο συμβατικός τρόπος χρήσης των βιομηχανικών ρομπότ μέχρι και σήμερα χαρακτηρίζεται από υψηλές ταχύτητες παραγωγής σε πλήρως απομονωμένο, από τον άνθρωπο, χώρο εργασίας. Ο ολοκληρωτικός διαχωρισμός του χώρου εργασίας στην ουσία εγγυάται την ασφάλεια του ανθρώπου αφού δεν υπάρχει φυσικός τρόπος επαφής με το ρομποτικό βραχίονα. Στα συνεργατικά συστήματα ωστόσο, οι χώροι εργασίας ανθρώπου και ρομπότ όχι μόνο δεν είναι πλήρως διαχωρισμένοι αλλά επικαλύπτονται, αυξάνοντας έτσι την πιθανότητα εμφάνισης ατυχημάτων.

Ο τύπος των ατυχημάτων και οι αιτίες που τα προκαλούν είναι ήδη γνωστά από τα υπάρχοντα συμβατικά ρομποτικά συστήματα. Οι τύποι των ατυχημάτων μπορούν να χωριστούν σε δύο μεγάλες κατηγορίες: α) Σύνθλιψη ανάμεσα σε μέλη του ρομπότ ή μεταξύ ρομπότ και τρίτου αντικειμένου, β) Σύγκρουση. Η πιο συχνή αιτία ατυχήματος είναι ο ανθρώπινος παράγοντας (ανθρώπινο λάθος), που περιλαμβάνει όλες της άμεσες και έμμεσες ενέργειες όπως ανεπαρκή εκπαίδευση, κόπωση, απώλεια αντίληψης και προσοχής, λανθασμένο χειρισμό, κλπ.

1.2.3 Τεχνικές ασφαλούς συνεργασίας

Τα τελευταία χρόνια, έχει γίνει μια αξιόλογη ερευνητική προσπάθεια επίλυσης προβλημάτων ασφάλειας της συνεργασίας Ανθρώπου-Ρομπότ και ειδικότερα τόσο στο σχεδιασμό του ίδιου του ρομπότ, όσο και σε τεχνικές ασφαλούς σχεδιασμού τροχιάς. Με την παράλληλη βελτίωση αυτών των δύο γνωστικών περιοχών επιδιώκεται η εξισορρόπηση μεταξύ της ασφάλειας της συνεργασίας και της διατήρησης της απόδοσης σε υψηλά επίπεδα. Στη παρούσα εργασία εστιάζουμε στις τεχνικές ασφαλούς σχεδιασμού κίνησης (προ-σύγκρουσης) μέσα σε ένα περιβάλλον όπου το ρομπότ έχει πλήρη ‘επίγνωση’ της κατάστασης. Οι αλγόριθμοι σχεδιασμού κίνησης αποτελούνται επιμέρους από αλγορίθμους σχεδιασμού διαδρομής και σχεδιασμού τροχιάς και χρησιμοποιούνται επιτυχώς εδώ και αρκετές δεκαετίες σε ρομποτικούς βραχίονες.



Εικόνα 1: Τεχνική εκ νέου σχεδιασμού τροχιάς προς αποφυγή συγκρούσεων **[1]**

1.3 Αντικείμενο της εργασίας

Στόχος της παρούσας εργασίας είναι η υλοποίηση ενός ελεγκτή σχεδιασμού τροχιάς για την αποφυγή σύγκρουσης του ρομπότ με τον άνθρωπο σε συνεργατικό περιβάλλον. Το συνεργατικό αυτό σενάριο έχει μοντελοποιηθεί και προσομοιωθεί σε εικονικό περιβάλλον (ΕΠε) μέσω της πλατφόρμας Unity και αποτελεί το αποτέλεσμα της διδακτορικής διατριβής του Δρα. Ηλία Μάτσα: «*Ανάπτυξη και Αξιολόγηση Διαδραστικών Μοντέλων Εικονικής Πραγματικότητας για το Σχεδιασμό Ρομποτικών Συστημάτων Παραγωγής*». Ακολούθως, στο Κεφάλαιο 2, γίνεται σύντομη αναφορά σε στοιχεία της διατριβής αυτής που κρίνονται απαραίτητα για την περαιτέρω κατανόηση του τελικού στόχου.

Κεφάλαιο 2: Ανασκόπηση υφιστάμενης πλατφόρμας

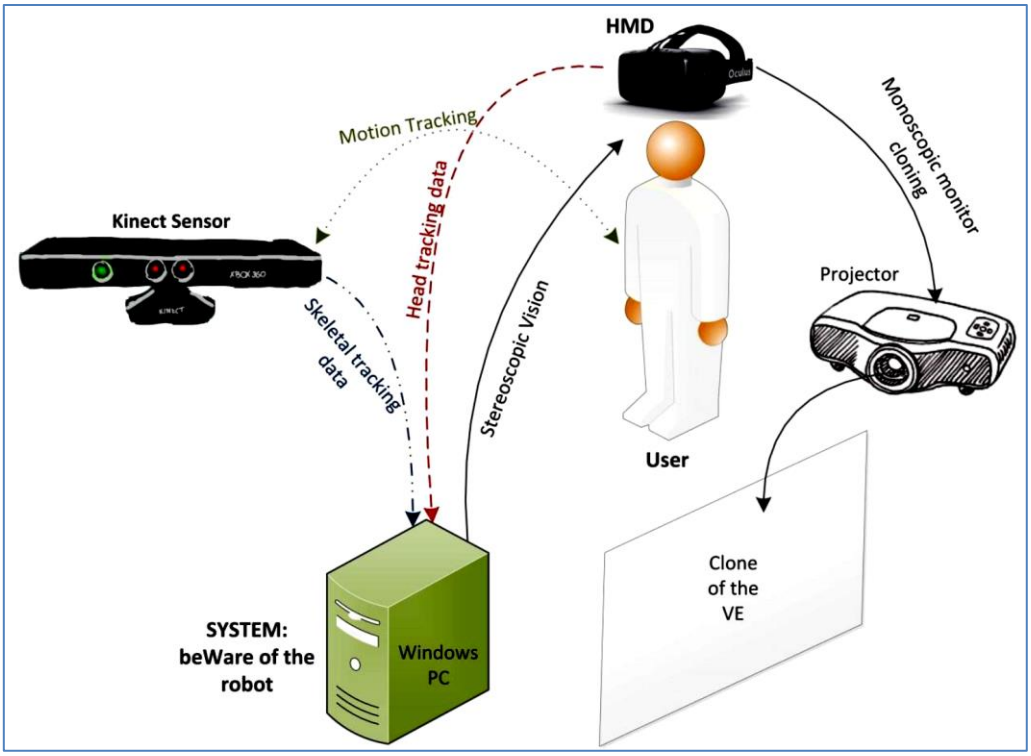
2.1 Περιγραφή

Το εικονικό περιβάλλον που δημιουργήθηκε, αποτέλεσμα της διδακτορικής διατριβής [2], ονομάζεται «*beWare of the Robot v2.0*» και είναι βασισμένο στην πλατφόρμα Unity 3d. Τα περισσότερα στοιχεία που απαρτίζουν το εικονικό αυτό περιβάλλον (πχ. τρισδιάστατα μοντέλα, μοντέλο ρομποτικού βραχίονα Stäubli™ RX90L, μοντέλο ανθρωποειδούς (avatar), με ένα σκελετό διπόδου (biped) και υφή δέρματος, ανταλλαγή δεδομένων με τις συσκευές tracking σε πραγματικό χρόνο, κώδικες σε C#, δυναμικό φωτισμό, δημιουργία σκιάσεων, κλπ.) είναι πρωτότυπα και έχουν δημιουργηθεί από το μηδέν.

Η αλληλεπίδραση των αντικειμένων του ΕΠε μεταξύ τους αλλά και με τον άνθρωπο γίνεται με τον προγραμματισμό κανόνων συμπεριφοράς που έχουν οριστεί εκ των προτέρων από τον προγραμματιστή προς την δημιουργία ενός σεναρίου. Πρέπει να αναφερθεί σε αυτό το σημείο πως βασικό πλεονέκτημα χρήσης μίας πλατφόρμας όπως το Unity είναι η ευελιξία που παρέχει στον προγραμματιστή να αλλάζει τη γεωμετρία και συνθήκες του περιβάλλοντος και να μεταβαίνει από ένα σενάριο σε άλλο άμεσα με ευκολία. Η σκηνή «*beWare of the Robot v2.0*» αποτελεί ένα από τα σενάρια που αναπτύχθηκαν στο οποίο ο βαθμός συνεργασίας είναι υψηλός.

Όσον αφορά την αλληλεπίδραση του χρήστη με το περιβάλλον και για την ρεαλιστική καταγραφή των ανθρώπινων κινήσεων χρησιμοποιείται εξοπλισμός ο οποίος καταγράφει τις κινήσεις του ανθρώπου σε πραγματικό χρόνο (Kinect) και τις δίνει σαν εισόδους στο κινηματικό μοντέλο του ανθρωποειδούς avatar. Αντίστοιχα, για την αίσθηση ρεαλισμού του

χρήστη, σαν συσκευή εξόδου χρησιμοποιείται η στερεοσκοπική μάσκα Oculus η οποία επιτρέπει την εμβύθιση στο ΕΠε. Ένα σχεδιάγραμμά των συσκευών αλληλεπίδρασης με το ΕΠε φαίνεται στην Εικόνα 2.



Εικόνα 2: Συσκευές αλληλεπίδρασης χρήστη - εικονικού περιβάλλοντος **[2]**.

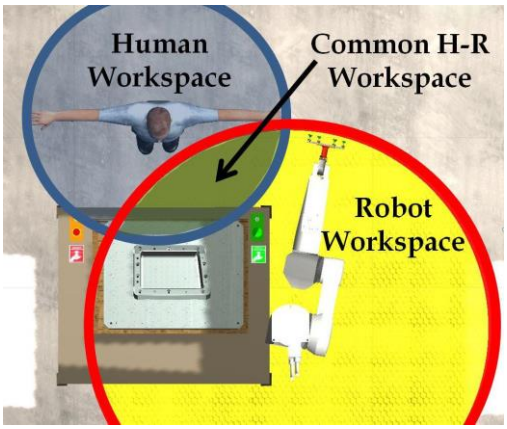
Η παρούσα εργασία χρησιμοποιεί το υφιστάμενο αυτό εικονικό περιβάλλον σα βάση και επεκτείνει τις δυνατότητες του επιτρέποντας το δυναμικό σχεδιασμό διαδρομής του τελικού σημείου δράσης του βραχίονα για την αποφυγή εμποδίων. Ο αναγνώστης μπορεί να ανατρέξει στη σχετική διδακτορική διατριβή [2] για λεπτομερή περιγραφή και ανάλυση του ΕΠε (Εικονικού Περιβάλλοντος).

2.2 Περιγραφή στοιχείων του Εικονικού Περιβάλλοντος

Η εφαρμογή επίδειξης του «*beWare of the Robot v2.0*» προσομοιώνει καθήκοντα ρομποτικής υποβοήθησης για τη συνεργατική στρώση υφασμάτων ανθρακονημάτων κατά την κατασκευή σύνθετων αεροπορικών τεμαχίων. Πιο συγκεκριμένα, υπάρχει συνεργατική προετοιμασία του ανθρακοϋφάσματος (αφαίρεση του αυτοκόλλητου) και επίσης, συνεργατική και από κοινού στρώση του ανθρακοϋφάσματος στο καλούπι.

Το πρόγραμμα ξεκινά με το ρομπότ να εκκινεί από την αρχική του θέση (home position) και να κατευθύνεται προς το τραπέζι όπου βρίσκεται το ανθρακο-ϋφασμα. Μόλις το τελικό σημείο δράσης του βραχίονα βρεθεί στην επιθυμητή θέση, ενεργοποιούνται οι βεντούζες που συλλέγουν το ύφασμα και το ρομπότ συνεχίζει προς εκτέλεση της προκαθορισμένης διαδρομής προς τον άνθρωπο. Αφού το ρομπότ σταματήσει σε μία θέση με συγκεκριμένη κατεύθυνση για την διευκόλυνση του ανθρώπου, αυτός αφαιρεί ένα ειδικό αυτοκόλλητο που βρίσκεται στη κάτω πλευρά ώστε να μην κολλάνε οι επιφάνειες του ανθρακοϋφάσματος μεταξύ τους. Πλέον, ο άνθρωπος και το ρομπότ, σε πλήρη συνεργασία, απλώνουν το ανθρακοΰφασμα στο καλούπι από κοινού. Παρατηρούμε ότι υπάρχουν περιοχές όπου ο άνθρωπος και το ρομπότ πρέπει να βρίσκονται ταυτόχρονα.

Εικόνα 3: Σενάριο beWare of the Robot v2.0



Εικόνα 4: Χώροι εργασίας ανθρώπου και ρομποτικού βραχίονα

Ο χώρος εργασίας του ανθρώπου και του ρομπότ είναι κοινός χωρίς προστατευτικές μπάρες ή πλέγμα διαχωρισμού. Το κομμάτι του χώρου αυτού που συμπίπτει (common workspace) είναι ο χώρος συνεργασίας ανθρώπου-ρομπότ.

2.2.1 Ρομπότ

Ο βιομηχανικός βραχίονας που προσομοιώνεται είναι ο 6 βαθμών ελευθερίας Stäubli RX90L και υπάρχει στη φυσική του μορφή στο εργαστήριο Τεχνολογίας των Κατεργασιών του ΕΜΠ.

Η τοποθέτηση του βραχίονα στο εικονικό περιβάλλον επιλέχθηκε να είναι ανάστροφη πάνω σε μία μεταλλική κατασκευή. Η θέση αυτή βοηθά στην απελευθέρωση χρήσιμου χώρου για ελιγμούς του ρομπότ κατά τη διάρκεια εκτέλεσης της εργασίας. Σε αυτή τη φάση, το ευθύ και αντίστροφο κινηματικό μοντέλο του βραχίονα δεν αξιοποιείται αφού για την κίνηση του χρησιμοποιούνται συναρτήσεις βιβλιοθηκών της πλατφόρμας Unity.

2.2.2 Άνθρωπος – Άβαταρ

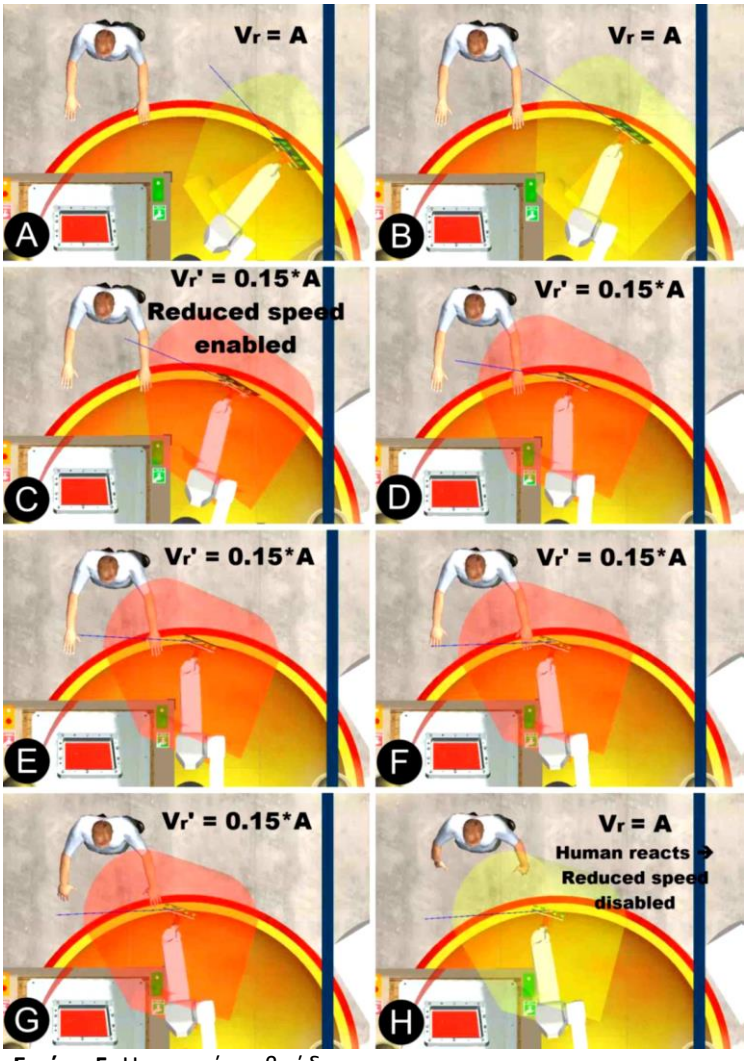
Σημαντική προσπάθεια έχει γίνει στην μοντελοποίηση των ανθρώπινων κινήσεων μέσω του ανθρωποειδούς avatar. Τονίζεται πως η εισαγωγή ενός ρεαλιστικού ανθρωποειδούς μπορεί να εντείνει την αίσθηση παρουσίας στον εικονικό κόσμο και να έχει θετικό αντίκτυπο στην αποτελεσματικότητα του ΕΠε.

Το πλέγμα του ανθρωποειδούς αποτελείται από 11000 πολύγωνα ενώ έχουν προσαρτηθεί σε αυτό ρούχα εργασίας και η υφή πραγματικού δέρματος, στοχεύοντας σε μία πιο ρεαλιστική απεικόνιση του ανθρώπινου σώματος. Πέρα από το 3d πλέγμα, το άβαταρ ενσωματώνει ένα πλήρως λειτουργικό κινηματικό μοντέλο με ένα σκελετό διπόδου το οποίο έχει σαν είσοδο το σήμα από της συσκευές καταγραφής κίνησης. Συγκεκριμένα, η κίνηση των χεριών καταγράφεται από τον αισθητήρα Kinect ενώ η κίνηση του κεφαλιού από τη στερεοσκοπική μάσκα Oculus. Επιπλέον, κατά την εκτέλεση του προγράμματος ο χρήστης έχει τη δυνατότητα θέασης του εικονικού περιβάλλοντος σε πρώτο πρόσωπο (first-person) μέσω ανάρτησης στο κεφάλι του avatar κάμερας, που αποτελεί αντικείμενο του Unity. Συνοψίζοντας, όλα τα παραπάνω στοιχεία συνεργάζονται με αποτέλεσμα μία πολύ ρεαλιστική προσομοίωση του ανθρώπινου σώματος μέσα στο ΕΠε. Ο φυσικός χρήστης αποκόπτεται εντελώς από τον πραγματικό κόσμο ενώ ταυτόχρονα οι κινήσεις του σε αυτόν αποτυπώνονται-αντιγράφονται με ακρίβεια στον εικονικό.

2.3 Τεχνικές αποφυγής συγκρούσεων

Για την αποφυγή συγκρούσεων κατά την εκτέλεση του προγράμματος, έχουν προγραμματιστεί δύο διαφορετικές τεχνικές: α) Επιβράδυνση βραχίονα, β) Υποχώρηση βραχίονα. Οι τεχνικές αυτές στηρίζονται σε συναρτήσεις εντοπισμού συγκρούσεων του Unity και δεν χρησιμοποιείται το κινηματικό μοντέλο του βραχίονα. Καταλαβαίνουμε λοιπόν ότι σε περίπτωση πρόβλεψης σύγκρουσης, ο βραχίονας πρέπει να μεταβάλλει ή και ακόμα να σταματήσει την προς εκτέλεση εργασία ωσότου το εμπόδιο πάψει να παρεμβάλλεται. Παρακάτω ακολουθεί μία συνοπτική περιγραφή των δύο αυτών τεχνικών.

2.3.1 Η τεχνική επιβράδυνσης του βραχίονα



Εικόνα 5: Η τεχνική επιβράδυνσης

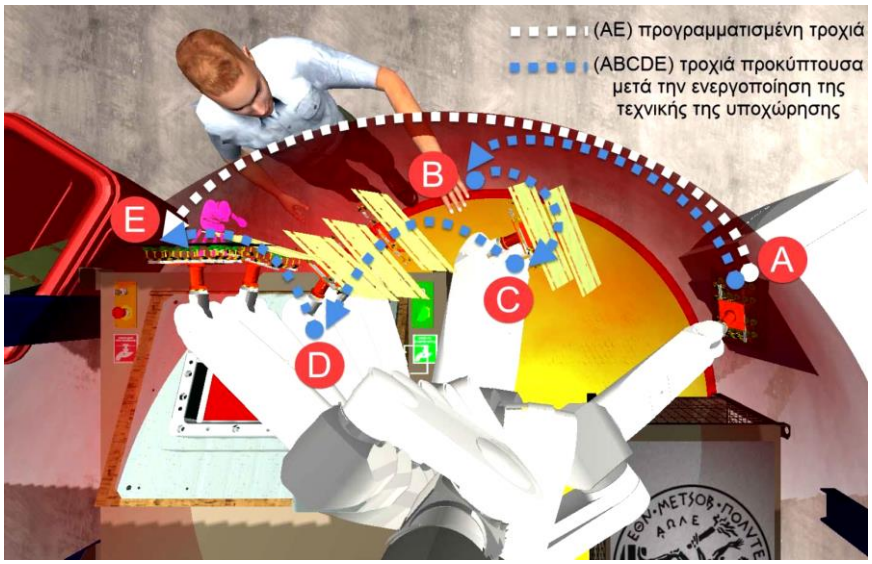
Η κίνηση του βραχίονα από το ένα σημείο στο επόμενο έχει προοριστεί στο πρόγραμμα εργασίας μέσω τιμών των γωνιών των αρθρώσεών του. Για την ομαλή μετάβαση από το ένα σημείο στο επόμενο χρησιμοποιείται γραμμική παρεμβολή για τον υπολογισμό των γωνιών όλων των ενδιάμεσων σημείων ενώ η ταχύτητα του βραχίονα μπορεί εύκολα να ορισθεί μέσω της συνάρτηση παρεμβολής του Unity. Εάν κατά την κίνηση του ρομπότ ικανοποιηθεί το κριτήριο ενεργοποίησης της επιβράδυνσης τότε, αμέσως, η γωνιακή ταχύτητα περιστροφής των αρθρώσεων του ρομπότ μειώνεται στο 15% της αρχικά

προγραμματισμένης. Η μειωμένη ταχύτητα παραμένει όσο εξακολουθεί να ικανοποιείται η συνθήκη ενεργοποίησης ενώ επανέρχεται στην αρχική της όταν αυτή παύει να ισχύει.

Η λογική της τεχνικής στηρίζεται στην υπόθεση ότι η επιβράδυνση της κίνησης του ρομπότ παρέχει τον απαιτούμενο χρόνο στον άνθρωπο να αντιληφθεί την πιθανότητα σύγκρουσης και να απομακρυνθεί από την τροχιά του ρομπότ.

2.3.2 Η τεχνική υποχώρησης του βραχίονα

Η υποχώρηση του βραχίονα σε περίπτωση πρόβλεψης σύγκρουσης, αποτελεί μία πιο πολύπλοκη τεχνική σε σύγκριση με την απλή επιβράδυνση. Η τεχνική αυτή εστιάζει στην πλήρη αποφυγή του ανθρώπου και στην παράλληλη εξακολούθηση εκτέλεσης της εργασίας.



Εικόνα 6: Η τεχνική υποχώρησης

Πιο συγκεκριμένα, σε περίπτωση εμφάνισης εμποδίου κατά μήκος της διαδρομής που πρέπει να κινηθεί ο βραχίονας ενεργοποιείται η τεχνική της υποχώρησης. Ο βραχίονας διακόπτει ακαριαία τη κίνηση που εκτελεί και ακολουθεί ένα νέο σημείο που έχει προ-καθοριστεί ανάλογα με το σενάριο εργασίας. Το σημείο αυτό θεωρείται ένα «ασφαλές σημείο» στο οποίο ο βραχίονας θα διαφύγει για την αποφυγή του ανθρώπου ενώ υπάρχουν πολλά τέτοια σημεία, προφανώς όχι άπειρα, για την κάλυψη όσο το δυνατόν περισσότερων σεναρίων. Εάν στο μεταξύ ο άνθρωπος απομακρυνθεί και δεν ανιχνευθεί εκ νέου επερχόμενη σύγκρουση, το ρομπότ θα ακολουθήσει απευθείας την τροχιά προς το

τελικό σημείο και θα ολοκληρώσει την κίνησή του. Τέλος πρέπει να σημειωθεί και εδώ ο προ-καθορισμένος και περιορισμένος χαρακτήρας αυτής της λύσης αφού ο βραχίονας μπορεί να μεταβεί στο τελικό σημείο μόνο από αυτά τα συγκεκριμένα «σημεία ασφάλειας».

Κεφάλαιο 3: Σχεδιασμός διαδρομής

3.1 Εισαγωγή

Ο σχεδιασμός κίνησης (Motion Planning) ενός ρομποτικού βραχίονα αποτελεί ένα σύνθετο πρόβλημα προς επίλυση καθώς εξαρτάται άμεσα από την γεωμετρία – κινηματικό μοντέλο του ρομπότ καθώς και τον αλγόριθμο που θα χρησιμοποιηθεί για τον σχεδιασμό της κίνησης. Ο σχεδιασμός κίνησης αποτελείται από τα επιμέρους: σχεδιασμός διαδρομής (Path Planning) και σχεδιασμός τροχιάς (Trajectory Planning). Στο παρόν κεφάλαιο αναπτύσσεται η θεωρία της μεθόδου σχεδιασμού διαδρομής RRT (Rapidly-exploring Random Tree) η οποία και υλοποιήθηκε στο εικονικό περιβάλλον συνεργασίας ανθρώπου – ρομπότ.

3.1.1 Σχεδιασμός διαδρομής (Path Planning)

Διαδρομή (path): Καμπύλη που διαγράφει το ρομπότ στο χώρο, η οποία ενώνει δύο (ή περισσότερες) ενδιάμεσες θέσεις (σημεία διέλευσης).

Σκοπός του σχεδιασμού διαδρομής είναι η εύρεση μίας διαδρομής, όσο το δυνατόν πιο σύντομη ή με άλλα κριτήρια βέλτιστη, που θα ενώνει την αρχική θέση με τη θέση στόχο. Το πρόβλημα μπορεί εύκολα να προσομοιωθεί με ένα πρόβλημα δύο διαστάσεων (2D) εύρεσης της συντομότερης διαδρομής πάνω σε ένα χάρτη μεταξύ δύο σημείων. Έχοντας

γνωστά το αρχικό και τελικό σημείο, η πλήρης επίγνωση του χάρτη (δηλαδή γνώση όλων των διαδρομών και εμποδίων) θα μας επέτρεπε να καταλήξουμε στη βέλτιστη λύση χωρίς σύγκρουση με εμπόδιο ή ακόμα και να μην καταλήξουμε αν αυτή τελικά δεν υπάρχει.
Ο σχεδιασμός διαδρομής για το τελικό σημείο δράσης ενός ρομποτικού βραχίονα μπορεί να διαφέρει από αυτόν ενός αυτόνομου ρομπότ καθώς όταν ο χώρος καταστάσεων (state space) είναι μεγάλος, μια πλήρης λύση του προβλήματος δεν είναι πάντα εφικτή.

3.1.2 Σχεδιασμός τροχιάς (Trajectory Planning)

Τροχιά (trajectory): Χρονική ακολουθία ενδιάμεσων θέσεων (στο Καρτεσιανό χώρο) ή διατάξεων (στο χώρο των αρθρώσεων)

Σκοπός του σχεδιασμού τροχιάς είναι ο υπολογισμός, συνήθως στο χώρο διάταξης (configuration space), μιας ομαλής τροχιάς από μία αρχική διάταξη σε μία τελική διάταξη – στόχο λαμβάνοντας υπόψη τα παρακάτω στοιχεία:

- Το κινηματικό μοντέλο του ρομπότ (πχ. για αποφυγή ιδιόμορφων σημείων)
- Τα φυσικά όρια των αρθρώσεων
- Την αποφυγή συγκρούσεων συνδέσμου με σύνδεσμο
- Την αποφυγή συγκρούσεων με το περιβάλλον (εμπόδια)

Γίνεται αντιληπτό, λοιπόν, ότι ο σχεδιασμός τροχιάς αφορά κυρίως την ομαλή χρονική μετάβαση από το ένα σημείο στο επόμενο και μέχρι τον τελικό στόχο, η δε πολυπλοκότητα αυτού εξαρτάται άμεσα από την γεωμετρία του βραχίονα. Στο σημείο αυτό, βέβαια, πρέπει να τονιστεί ότι ο σχεδιασμός τροχιάς γίνεται αφού έχει γίνει ο σχεδιασμός διαδρομής που θα ακολουθήσει το ρομπότ. Στην περίπτωση των ρομποτικών βραχιόνων πολλών βαθμών ελευθερίας όπου υπάρχουν πολλαπλές λύσεις (ίσως και άπειρες), ο σχεδιαστής τροχιάς (trajectory planner) πρέπει να έχει συνεχή συνεργασία με το σχεδιαστή διαδρομής (path planner) ώστε να βρεθεί η «βέλτιστη» λύση.

Για παράδειγμα για έναν ρομποτικό βραχίονα 6 βαθμών ελευθερίας: Ο σχεδιαστής διαδρομής θα εξερευνήσει όλο το χώρο εργασίας του βραχίονα και θα βρει τη συντομότερη διαδρομή για να μεταβεί το τελικό σημείο δράσης του βραχίονα (end effector) στο σημείο

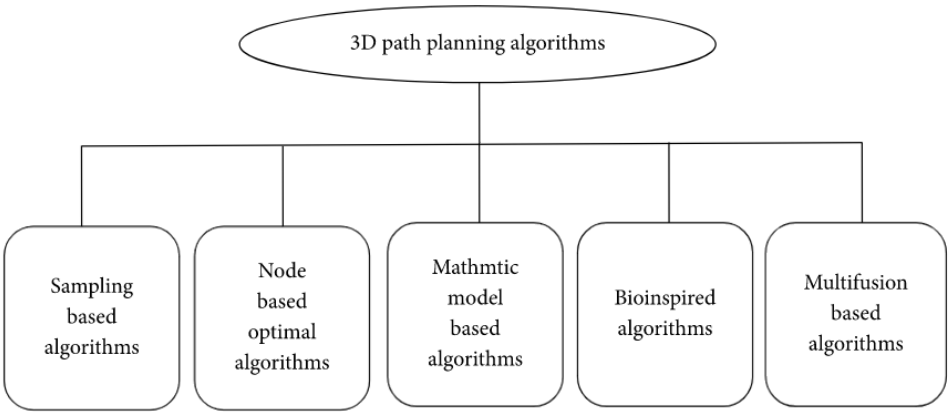
στόχο. Η διαδρομή αυτή θα διασπαστεί σε επιμέρους κομμάτια μικρών αποστάσεων και με τη σειρά του ο σχεδιαστής τροχιάς πρέπει για κάθε ένα κομμάτι να υπολογίσει τη μεταβολή στις γωνίες των αρθρώσεων του βραχίονα ώστε η μετάβαση από το ένα σημείο στο επόμενο να είναι χρονικά ομαλή. Επιπλέον, ο σχεδιαστής τροχιάς θα πρέπει να αποφύγει τυχόν συγκρούσεις των αρθρώσεων του ρομπότ με το περιβάλλον ή και με τον εαυτό του και μόνο τότε θα μπορέσει ο βραχίονας να ξεκινήσει την κίνηση του προς το τελικό στόχο.

3.2 Βιβλιογραφική ανασκόπηση

Υπάρχει μία πληθώρα ερευνητικών εργασιών πάνω στην βελτιστοποίηση ενός ασφαλούς δυναμικού σχεδιασμού κίνησης ρομποτικού βραχίονα πολλών βαθμών ελευθερίας. Η μεγάλη ποικιλία έγκειται στο γεγονός ύπαρξης πολλών αλγορίθμων σχεδιασμού διαδρομής και σχεδιασμού τροχιάς και την προσπάθεια εύρεσης του κατάλληλου συνδυασμού τους. Ακολουθεί συνοπτική βιβλιογραφική ανασκόπιση των αλγορίθμων σχεδιασμού διαδρομής και οι εφαρμογές τους.

3.2.1 Αλγόριθμοι σχεδιασμού διαδρομής

Έχουν αναπτυχθεί πάρα πολλοί αλγόριθμοι σχεδιασμού διαδρομής ήδη από τον προηγούμενο αιώνα. Μία πρώτη αναλυτική κατηγοριοποίηση των προβλημάτων σχεδίασης δρόμων καθώς και μεθοδολογίες επίλυσής τους έγινε στη μελέτη των Hwang και Ahuja [3]. Μέχρι και σήμερα η εξέλιξη αυτών των αλγορίθμων είναι ραγδαία και γίνονται πολλές προσπάθειες περαιτέρω ταξινόμησης των αλγορίθμων με βάση τα μοναδικά χαρακτηριστικά που έχει ο καθένας. Μία πιο πρόσφατη ταξινόμηση των αλγορίθμων σχεδιασμού διαδρομής παρουσιάζεται στο Σχήμα 1.



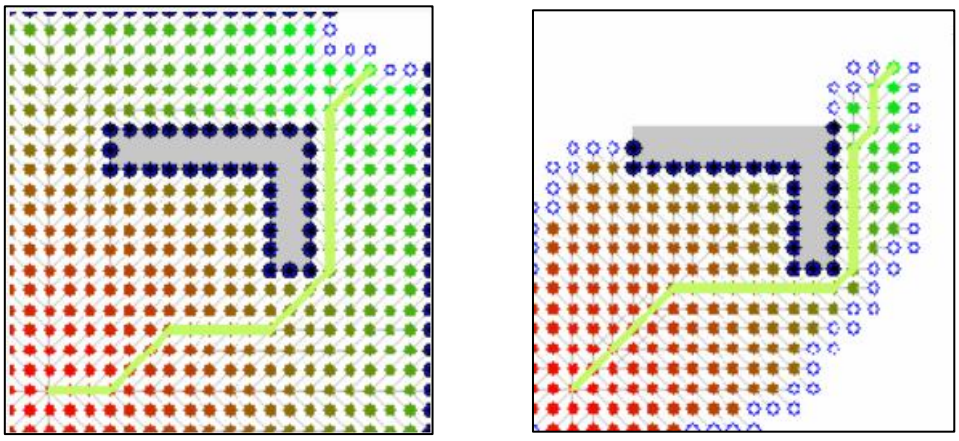
Σχήμα 1: Κατηγοριοποίηση αλγορίθμων σχεδιασμού διαδρομής κατά Liang Yang et.al **[4]**

Παρακάτω ακολουθεί γρήγορη αναφορά των χαρακτηριστικών της κάθε κατηγορίας:

1) Δειγματοληπτικοί αλγόριθμοι (Sampling based): Βασικό και κοινό στοιχείο όλων των δειγματοληπτικών αλγορίθμων είναι ο έλεγχος και η χρήση τυχαίων σημείων – υποθέσεων. Εκεί που διαφοροποιούνται είναι στον τρόπο που χρησιμοποιούνται αυτά τα τυχαία σημεία ώστε ο αλγόριθμος να φτάσει σε ένα βέλτιστο. Σημαντικό πλεονέκτημα του δειγματοληπτικού ελέγχου σημείων είναι η αποφυγή τοπικών ελαχίστων, όπου άλλου είδους αλγόριθμοι παγιδεύονται. Μπορούν περαιτέρω να χαρακτηριστούν σαν ενεργοί ή παθητικοί, από τους οποίους οι πρώτοι έχουν τη δυνατότητα να βρουν τη βέλτιστη διαδρομή από μόνοι τους. Συναντώνται κατά κόρον σε εφαρμογές πραγματικού χρόνου (real-time) καθώς επιτυγχάνουν υψηλές χρονικά αποδόσεις για στατικά αλλά και δυναμικά προβλήματα – εμπόδια.

   Η χρήση του δειγματοληπτικού αλγορίθμου RRT σε πραγματικό χρόνο παρουσιάζεται στα [5], [6]. Πιο συγκεκριμένα στο [6] γίνεται χρήση δύο διαφορετικών ροποτικών βραχιόνων όπου το ένα λειτουργεί σαν δυναμικό κινούμενο εμπόδιο με αυθαίρετη τροχιά ενώ το άλλο πρέπει να εκτελέσει μία εργασία μετάβασης από ένα αρχικό σημείο σε ένα σημείο στόχο. Επιπλέον, στα [7], [8] γίνεται χρήση των bidirectional RRT όπου η επέκταση του δέντρου γίνεται ταυτόχρονα και σε αντίθετη κατεύθυνση από δύο δέντρα με στόχο την ταχύτερη σύγκλιση του αλγορίθμου σε μία λύση. Στην κατηγορία των δειγματοληπτικών αλγορίθμων ανήκουν και οι αλγόριθμοι PRM (Probabilistic Roadmap Planner) [9].

2) Αλγόριθμος διακριτού χώρου – γράφοι (Node based): Το πλήρες περιβάλλον είναι γνωστό και διακριτοποιείται σε μορφή πλέγματος όπου οι κόμβοι έχουν συγκεκριμένη διάσταση (ανάλυση). Οι αλγόριθμοι διακριτού χώρου, χρησιμοποιούν την πληροφορία κάθε κόμβου και μετατρέπουν την απόσταση μεταξύ των κόμβων σε βάρη τα οποία και χρησιμοποιούν για την δημιουργία της διαδρομής. Προφανώς η ποιότητα του τελικού αποτελέσματος εξαρτάται από την ανάλυση διακριτοποίησης του χώρο, η οποία όσο αυξάνεται απαιτεί και μεγαλύτερο υπολογιστικό κόστος. Στην κατηγορία αυτή συγκαταλέγονται πέραν άλλων και οι γνωστοί αλγόριθμοι Dijkstra (Edsger W. Dijkstra, 1956) και A* (Peter Hart, Nils Nilsson και Bertram Raphael, 1968) μαζί με τις παραλλαγές τους.



Σχήμα 2: Αλγόριθμοι Dijkstra (αριστερά) και A* (δεξιά)

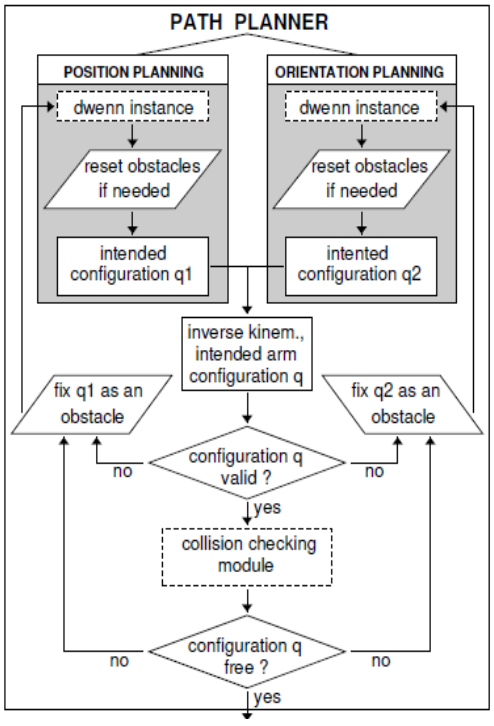
3) Αλγόριθμοι με μαθηματικά μοντέλα (Mathematic model based): Αυτού του είδους οι αλγόριθμοι έχουν ως στόχο την μαθηματική περιγραφή όλου του χώρου εργασίας καθώς και οποιονδήποτε περιορισμών, δυναμικών και κινηματικών. Θεωρούνται αρκετά αξιόπιστοι με μεγαλύτερο πλεονέκτημα την ασφάλεια που παρέχουν με την διατήρηση όλων των περιορισμών που έχουν τεθεί. Παρόλο που αλγόριθμοι αυτοί απαιτούν μεγάλη υπολογιστική ισχύ, με τη ραγδαία εξέλιξη των υπολογιστικών συστημάτων η χρήση τους ολοένα αυξάνεται και εξελίσσεται.

   Ο μαθηματικός σχεδιασμός κίνησης για ένα ανθρωπομορφικό βραχίονα 7-DOF μέσω της επίλυσης των κινηματικών μοντέλων και υπολογισμό της Ιακωβιανής του μήτρας παρουσιάζεται στο [10].

4) Βιο-μιμητικοί (Bioinspired) αλγόριθμοι : Οι αλγόριθμοι αυτοί μιμούνται βιολογικές διεργασίες με σκοπό την λύση ενός προβλήματος. Είναι χρήσιμοι σε προβλήματα που περιέχουν πολλές παραμέτρους και δεν υπάρχει αναλυτική μέθοδος που να μπορεί να βρει το βέλτιστο συνδυασμό τιμών για τις μεταβλητές ώστε το υπό εξέταση σύστημα να αντιδρά με όσο το δυνατόν περισσότερο επιθυμητό τρόπο. Χαρακτηριστικοί αλγόριθμοι αυτής της κατηγορίας είναι τα νευρωνικά δίκτυα (NN - Neural Networks) και οι γενετικοί αλγόριθμοι (GA – Genetic Algorithms). Οι τελευταίοι χρησιμοποιούν τις βιολογικές διεργασίες της εξέλιξης μέσω μετάλλαξης, φυσικής επιλογής και διασταύρωσης σε μια επαναληπτική διαδικασία. Παρότι μπορούν να λύσουν και δυναμικά προβλήματα η χρήση τους προτείνεται για offline προγραμματισμό.

Στα [11] και [12] παρουσιάζεται η χρήση γενετικού αλγορίθμου για την ελαχιστοποίηση συνεχούς πολυωνυμικής συνάρτησης $4^{ου}$ και $7^{ου}$ βαθμού, αντίστοιχα, η οποία περιγράφει τη διαδρομή των γωνιών των αρθρώσεων ενός 3-DOF ρομποτικού βραχίονα. Ο σχεδιασμός κίνησης γίνεται από σημείο σε σημείο (όχι ευθεία γραμμή απαραίτητα) και λαμβάνει υπόψη πολύπλοκες γεωμετρίες εμποδίων καθώς και όλους τους κινηματικούς και δυναμικούς περιορισμούς του ρομπότ.

Μία παραλλαγή του αλγορίθμου των νευρωνικών δικτύων χρησιμοποιείται στο [13] ενώ παράλληλα γίνεται διάσπαση του προβλήματος της λύσης ενός 7-DOF ρομποτικού βραχίονα σε δύο πιο απλά 3D προβλήματα: 1) Σχεδιασμός θέσης τελικού σημείου δράσης και 2) σχεδιασμός διεύθυνσης τελικού σημείου δράσης. Πιο αναλυτικά, χρησιμοποιούνται δύο οντότητες του αλγορίθμου DWENN (Dynamic Wave Expansion Neural Network) (Σχήμα 3) με την κάθε μία να επιλύει το επιμέρους 3D πρόβλημα. Η δυσκολία έγκυται στο γεγονός ότι παρότι τα επιμέρους προβλήματα είναι μαθηματικώς ανεξάρτητα, αυτά αναφέρονται στο ίδιο φυσικό πρόβλημα.



Σχήμα 3: Σχεδιασμός κίνησης 7-DOF βραχίονα κατά **[13]**

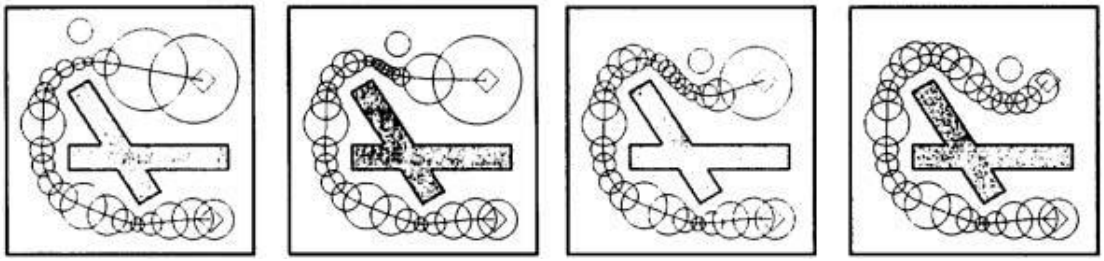
5) Συνδυαστικοί αλγόριθμοι (Multifusion): Σε αυτή τη κατηγορία ανήκουν οι αλγόριθμοι που αποτελούν αποτέλεσμα συνδυασμού άλλων επιμέρους αλγορίθμων. Οι μεθοδολογίες αυτές κερδίζουν ολοένα και περισσότερο έδαφος, καθώς συνδυάζουν τα πλεονεκτήματα των επιμέρους αλγορίθμων βελτιώνοντας έτσι την απόδοση της σχεδίασης διαδρομής σε πραγματικό χρόνο. Ένα γνωστό παράδειγμα τέτοιου αλγορίθμου είναι η χρήση αλγορίθμου δυναμικών πεδίων (Potential field) σε συνδυασμό με συναρτήσεις πλοήγησης (navigation functions) για την αποφυγή παγίδευσης σε τοπικά ελάχιστα. Άλλα παραδείγματα υβριδικών αλγορίθμων μπορούμε να δούμε στη δουλειά των K.Oh et al [14], όπου γίνεται χρήση διανυσμάτων - Support Vector Machine (SVM) – για την βέλτιστη κατεύθυνση δημιουργίας ενός δέντρου RRT με στόχο τη μείωση του υπολογιστικού χρόνου. Παρόμοια, στο [15], γίνεται χρήση διατάξεων – κλειδιών για την καθοδήγη επέκτασης του δέντρου και την τελική ένωση της αρχικής διάταξης του βραχίονα με την τελική διάταξη – στόχο.

Η ολοκληρωτική κατηγοριοποίηση των αλγορίθμων σχεδιασμού διαδρομής είναι αρκετά πολύπλοκη καθώς συνεχώς αναπτύσσονται νέες μέθοδοι και υβρίδια για τη βελτιστοποίηση τους. Αξίζει να αναφερθούμε στους παρακάτω αλγορίθους οι οποίοι εστιάζουν στον ταχύτερο σχεδιασμό διαδρομής για μεγάλους χώρους καταστάσεων πολλών διαστάσεων σε ένα δυναμικά και απρόβλεπτα μεταβαλλόμενο περιβάλλον. Ο καθένας από αυτούς του αλγορίθμους προσπαθεί να λύσει το πρόβλημα του ταχέως δυναμικού σχεδιασμού διαδρομής με τελείως διαφορετικό τρόπο.

- Αλγόριθμος Ελαστικής ζώνης (Elastic Band). [16] [17]

  Ο αλγόριθμος της ελαστικής ζώνης αναπτύχθηκε από τους Quinlan and Khatib [16] με στόχο την πλήρωση του χάσματος μεταξύ της κατασκευής διαδρομής και του ελέγχου του ρομπότ, με έναν ομαλό και ασφαλή τρόπο. Η ιδέα πίσω από τον αλγόριθμο της ελαστικής ζώνης έχει να κάνει με την ελαστική παραμόρφωση μίας δεδομένης διαδρομής, βάσει τεχνητών δυνάμεων, με στόχο την αποφυγή εμποδίων και την εξομάλυνση της. Ο αλγόριθμος της ελαστικής ζώνης αποσκοπεί στην μίμηση της συμπεριφοράς ενός λάστιχου που παραμορφώνεται, εξαιτίας ενός συνόλου δυνάμεων που ασκούνται σ’ αυτό. Οι δυνάμεις αυτές είναι γενικά δύο ειδών, εσωτερικές ελκτικές δυνάμεις συστολής και εξωτερικές απωστικές δυνάμεις, που

παραμορφώνουν το λάστιχο, μέχρι να φτάσει σε ένα επίπεδο ηρεμίας. Το κλειδί για την υλοποίηση του αλγορίθμου της ελαστικής ζώνης, αποτελεί η φούσκα (bubble) η οποία ορίζεται ως το μέγιστο τοπικό, προσβάσιμο τμήμα του ελεύθερου χώρου, γύρω από μία κατάσταση q_i. Το μέγεθος της φούσκας είναι ανάλογο της απόστασης του ρομπότ σε μία κατάσταση q_i από το κοντινότερο εμπόδιο, ενώ ουσιαστικά δηλώνει το σύνολο των δυνατών καταστάσεων q στις οποίες μπορεί να διέλθει το ρομπότ από μία κατάσταση q_i. Η ελαστική ζώνη κατασκευάζεται, αρχικά βάσει του ολικού μονοπατιού που παράγεται από έναν αλγόριθμο κατασκευής μονοπατιών (πχ. A*), τοποθετώντας μία φούσκα σε κάθε ενδιάμεσο σημείο του μονοπατιού.



Σχήμα 4: Παραμόρφωση ελαστικής ζώνης, βάσει ελκτικών δυνάμεων μεταξύ των φουσκών και απωστικών δυνάμεων από το στατικό και το δυναμικό εμπόδιο **[16]**.

- Αλγόριθμος Ελαστικής λωρίδας (Elastic strip). [18] [19]

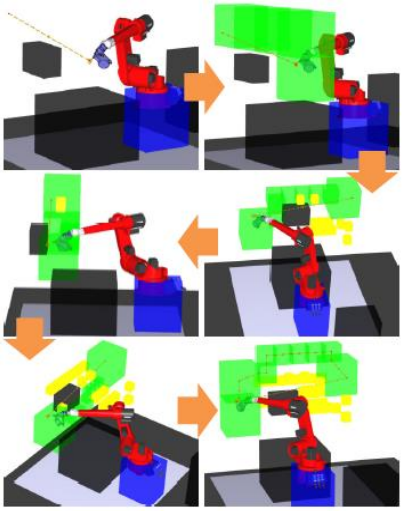
  Ο αλγόριθμος Ελαστικής λωρίδας αποτελεί μία αναβάθμιση του αλγορίθμου Ελαστικής ζώνης και τον βελτιώνει με τρεις ανεξάρτητους τρόπους: 1) Η αναπαράσταση του ελεύθερου χώρου γίνεται στο χώρο εργασίας (workspace) και όχι στο χώρο διάταξης (configuration space) βελτιώνοντας έτσι σημαντικά την προσαρμογή του αλγορίθμου σε προβλήματα πολλών βαθμών ελευθερίας. 2) Λαμβάνει υπόψη τη δυναμική του ρομποτικού βραχίονα και επιτρέπει την ενσωμάτωση περιορισμών εργασίας (task-level) κατά την αποφυγή εμποδίων. 3) Περιλαμβάνει ένα μηχανισμό επιδιόρθωσης της διαδρομής για την αποφυγή μετατροπής της σε μη βέλτιστη. Παράδειγμα, η περίπτωση που ένα εμπόδιο περάσει σε κάποιο σημείο της διαδρομής. Η ελαστική λωρίδα αποκόπτεται στα δύο στο σημείο που περνάει το εμπόδιο. Μόλις το εμπόδιο εξαφανιστεί, αυτή επιδιορθώνεται με τη χρηση των εσωτερικών δυνάμεων. Σε διαφορετική περίπτωση μη χρήσης αυτής της μεθόδου, θα πρέπει κάθε φορά που ένα εμπόδιο

παρουσιάζετε να γίνετε χρήση ενός εξωτερικού αλγορίθμου σχεδιασμού διαδρομής (πχ. A*) δημιουργώντας έτσι μη βέλτιστες λύσεις.

- Αλγόριθμος σχεδιασμού διαδρομής με αποσύνθεση (Decomposition based). [20] [21]

  Διασπούν το πρόβλημα σχεδιασμού κίνησης σε δύο επιμέρους προβλήματα, ένα στο χώρο εργασίας και ένα στο χώρο διάταξεων του ρομπότ.

  Η εργασία [22] παρουσιάζει τη χρήση του αλγορίθμου για ένα 6-DOF ρομποτικό βραχίονα. Με τη χρήση αλγορίθμου διακριτού χώρου δημιουργείται ο χώρος κίνησης του βραχίονα ενώ ο υπολογισμός των ενδιάμεσων σημείων απο το αρχικό και τελικό σημείο γίνεται με αλγορίθμους παρεμβολής και χρήση των κινηματικών μοντέλων του βραχίονα. Κατά τη διακριτοποίηση του χώρου κάθε όγκος (μεγέθους ανάλογο της ανάλυσης που έχει οριστεί) μπορεί να πάρει 3 τιμές: Ελεύθερος, Εμπόδιο, Μικτός ανάλογα από το ποσοστό κάλυψης του χώρου από εμπόδια. Ο τελικός χώρος κίνησης του βραχίονα αποτελείται από του «ελευθερους» όγκους.

  Σχήμα 5: Κατά **[22]**. Δημιουργία χώρου κίνησης (πράσινο) και υπολογισμός ενδιάμεσων σημείων. Δημιουργία εικονικών εμποδίων (κίτρινο) σε περίπτωση μη εύρεσης λύσης για τα ενδιάμεσα σημεία.

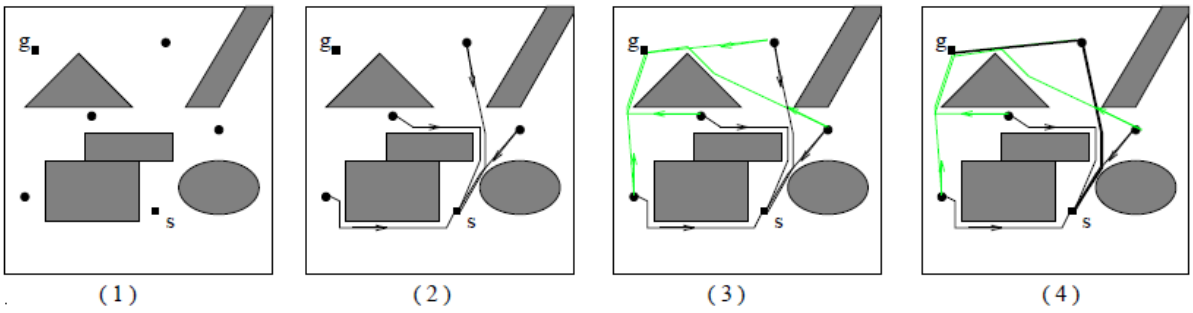
- PRM για μεταβαλλόμενα περιβάλλοντα (PRM-Changing Environments/PRMCE). [23]

  Ο αλγόριθμος PRMCE προσπαθεί να επιτύχει το σχεδιασμό διαδρομής σε πραγματικό χρόνο με τη χρήση δημιουργίας ενός προ-υπολογισμένου (preprocessing) χάρτη. Ο προ-υπολογισμός εκτελείται την πρώτη φορά που τρέχει ο αλγόριθμος για ένα συγκεκριμένο ρομποτικό βραχίονα και μπορεί να πάρει αρκετό χρόνο λόγο των πολλαπλών λύσεων που υπάρχουν. Βασίζεται στις ίδιες αρχές με τον αλγόριθμο PRM αλλά επεκτείνεται για δυναμικές εφαρμογές πραγματικού χρόνου.

- Flexible Anytime Dynamic PRM (FADPRM). [24]

Ο αλγόριθμος FADPRM προσπαθεί να επιτύχει δυναμικό σχεδιασμό διαδρομής χρησιμοποιώντας μία ήδη υπολογισμένη διαδρομή την οποία διορθώνει όπου είναι απαραίτητο. Για την διόρθωση γίνεται ένας ταχύτατος υπολογισμός μίας μη βέλτιστης λύσης η οποία εν συνέχεια βελτιώνεται σταδιακά όσο δεν ικανοποιείται ένα χρονικό κριτήριο. Αποτελεί συνδυασμό του αλγορίθμου AD* (Anytime Dynamic A* και του PRM.

Επιπλέον, οι Caigong Qin και Dominik Henrich [25] προτείνουν την χρήση ενός αλγόριθμου τυχαίας παράλληλης εξερεύνησης (randomized parallel search). Έχοντας το αρχικό και τελικό σημείο, η ιδέα είναι η δημιουργία τυχαίων «ενδιάμεσων» σημείων και η διάσπαση του προβλήματος σε δύο βήματα. Πρώτο βήμα είναι η παράλληλη εύρεση ελεύθερης διαδρομής από τα ενδιάμεσα αυτά σημεία προς το αρχικό και δεύτερο βήμα η παράλληλη εύρεση ελεύθερης διαδρομής από τα ενδιάμεσα σημεία προς το τελικό. Το τελικό μονοπάτι εκλέγεται μέσω κριτηρίου ελαχιστοποίησης της απόστασης. Η προσέγγιση αυτή μπορεί να επεκταθεί και σε περισσότερα βήματα με στόχο την εξάλειψη των περιπτώσεων τοπικών ελαχίστων και παγίδευσης του αλγορίθμου, ο οποίος στην απλή του μορφή απλά θα επεκτεινόταν σε μία ευθεία γραμμή από το αρχικό σημείο προς το τελικό. Παρόλα αυτά η δημιουργία των τυχαίων ενδιάμεσων σημείων δεν είναι απλή υπόθεση και προτείνεται η χρήση των γενικευμένων διαγραμμάτων Voronoi.



Σχήμα 6: Σχεδιασμός διαδρομής 2 βημάτων κατά **[25]:** 1) Δημιουργία ενδιάμεσων τυχαίων σημείων, 2) Παράλληλη εξερεύνηση ελεύθερης διαδρομής από τα ενδιάμεσα σημεία προς το αρχικό, 3) Παράλληλη εξερεύνηση ελεύθερης διαδρομής από τα ενδιάμεσα σημεία προς το τελικό, 4) Επιλογή της τελικής διαδρομής με βάση κριτήρια (πχ. η πρώτη λύση που θα βρεθεί).

3.3 Αλγόριθμος RRT

Στην παρούσα εργασία χρησιμοποιήθηκε ο απλός αλγόριθμος RRT (Rapidly-exploring Random Trees) ο οποίος θεωρείται αρκετά αποδοτικός στον σχεδιασμό διαδρομής και ιδιαίτερα σε προβλήματα πολλών παραμέτρων – διαστάσεων.

Όπως προαναφέρθηκε οι δειγματοληπτικοί αλγόριθμοι, αντί να εξερευνήσουν όλο το χώρο εργασίας για όλες τις πιθανές λύσεις, σχεδιάζουν πιθανές διαδρομές προσθέτοντας κάθε φορά ένα κόμβο-σημείο σε ένα δέντρο μέχρι να βρεθεί μια λύση ή να λήξει το χρονικό όριο που έχει τεθεί. Αυτοί οι αλγόριθμοι χαρακτηρίζονται ως πιθανοτικά ολοκληρωμένοι (probabilistic complete) καθώς η πιθανότητα εύρεσης μιας λύσης τείνει στο 1 όταν ο χρόνος τείνει στο άπειρο. Παρόλα αυτά, δεν μπορούν να προσδιορίσουν την ύπαρξη ή μη λύσης.

3.3.1 Ψευδοκώδικας

Ο βασικός αλγόριθμος RRT στην απλή του μορφή μπορεί εύκολα να περιγραφεί με τη χρήση ψευδοκώδικα. Έστω T η δομή του δέντρου και q_{init} η αρχική του κατάσταση (σημείο έναρξης). Τα βήματα του αλγορίθμου είναι:

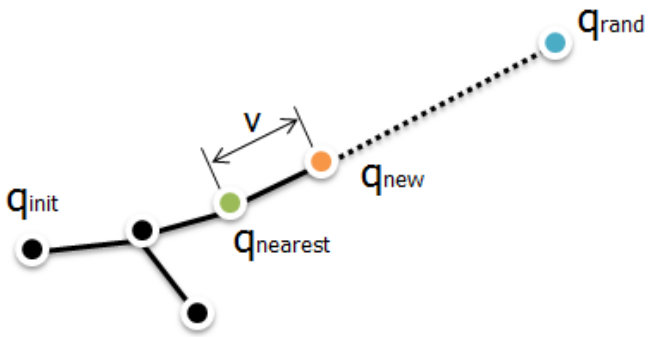
1) Δημιουργία ενός κενού δέντρου
2) Προσθήκη αρχικού σημείου στο δέντρο
3) Όσο δεν έχει βρεθεί ο τελικός στόχος (ή παρέλθει το χρονικό όριο)
   3.1Επιλογή ενός τυχαίου σημείου q_{rand}
   3.2Εύρεση στο δέντρο του κοντινότερου σημείου (q_{near}) με το q_{rand}
   3.3Έλεγχος συγκρούσεων στο μονοπάτι που ενώνει τα q_{near} - q_{rand}
   3.4Ένωση των σημείων αν δεν υπάρχει σύγκρουση

Ακολουθεί ο ψευδοκώδικας του αλγορίθμου καθώς και η αναπαράσταση επέκτασης ενός RRT (Σχήμα 7).

```
RRT(q_init) {
    T.init(q_init);
    for k = 1 to K do {
    q_rand = RANDOM_CONFIG();
    EXTEND(T, q_rand) }
}
```

Η συνάρτηση EXTEND περιλαμβάνει τον έλεγχο συγκρούσεων και αποφασίζει αν ο νέος κόμβος θα απορριφθεί ή θα προστεθεί στο δέντρο.

Το δέντρο ξεκινά από μία αρχική κατάσταση και επεκτείνεται σταδιακά μέχρι να βρει μία διαδρομή προς το σημείο στόχο. Σε κάθε βήμα επιλέγεται ένα τυχαίο σημείο του χώρου. Αν αυτό το σημείο βρίσκεται σε περιοχή χωρίς εμπόδιο τότε ο αλγόριθμος ψάχνει να βρει τον κοντινότερο κόμβο του δέντρου και ελέγχει αν αυτά τα δύο σημεία μπορούν να ενωθούν. Για την ένωση των δύο σημείων υπάρχει ένα προκαθορισμένο βήμα κάλυψης της απόστασης ανάλογα με την ανάλυση που θέλουμε να έχει ο εντοπισμός ενδιάμεσων εμποδίων. Όσο δεν υπάρχει σύγκρουση, τα σημεία του κάθε βήματος ενώνονται με το προηγούμενο τους, επεκτείνοντας έτσι το δέντρο. Ο αλγόριθμος ολοκληρώνεται είτε όταν βρεθεί μία διαδρομή που ενώνει το αρχικό σημείο με το σημείο-στόχο, είτε όταν έχει τελειώσει το όριο που έχει τεθεί (χρονικό ή αριθμός επαναλήψεων).



Σχήμα 7: Αναπαράσταση επέκτασης RRT

Οι βασικές υπορουτίνες που χρησιμοποιούνται από τον αλγόριθμο RRT μπορούν να κατηγοριοποιηθούν σε:

- Υπορουτίνα εύρεσης του επόμενου σημείου για προσθήκη στο δέντρο.
- Υπορουτίνα επέκτασης δέντρου - σύνδεσης του σημείου με το υπόλοιπο δέντρο.
- Υπορουτίνα αξιολόγησης και βελτιστοποίησης της διαδρομής.

3.3.2 Υπορουτίνα εύρεσης σημείου - κόμβου

Η πιο κλασσική μέθοδος εύρεσης κόμβων προς προσθήκη στο δέντρο είναι με τη χρήση αλγορίθμου τυχαιότητας. Έχοντας ορίσει και οριοθετήσει τον χώρο εργασίας με τα

κατάλληλα όρια, ένας αλγόριθμος τυχαιότητας παράγει τυχαία σημεία τα οποία και εξετάζονται περαιτέρω για τη σύνδεσή τους στο δένδρο. Παρόμοιες μεθοδολογίες περιλαμβάνουν επίσης την επιλογή τυχαίων σημείων αλλά μέσα σε μία περιοχή κοντά στον εκάστοτε αρχικό κόμβο και όχι σε όλο το χώρο εργασίας.

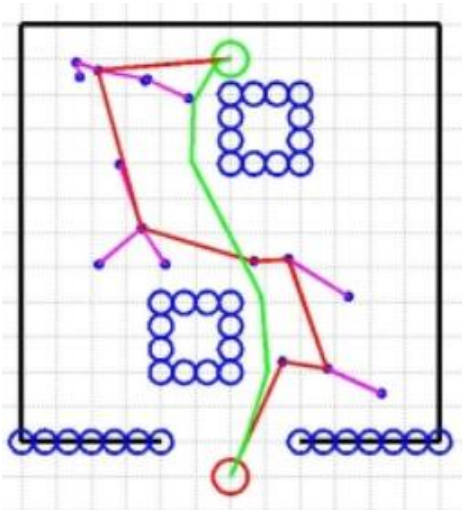
3.3.3 Υπορουτίνα επέκτασης δέντρου

Η υπορουτίνα αυτή καθορίζει αν ο νέος κόμβος τηρεί τα κριτήρια για την προσθήκη του στο δέντρο. Κάθε νέος κόμβος συνδέεται με τον κοντινότερο προϋπάρχοντα κόμβο του δέντρου. Αυτό γίνεται εύκολα υπολογίζοντας την απόσταση όλων των κόμβων του δέντρου από τον καινούριο κόμβο και επιλέγοντας την μικρότερη εξ αυτών. Η μεθοδολογία αυτή δεν εγγυάται την εύρεση της συντομότερης διαδρομής στο χώρο, παρόλα αυτά υπάρχουν διάφορες παραλλαγές του αλγορίθμου (πχ. RRT*) που εστιάζονται στην ελαχιστοποίηση της απόστασης της διαδρομής.

Προτού προστεθεί οποιοδήποτε σημείο στο δέντρο, αυτό εξετάζεται για τυχόν συγκρούσεις με το περιβάλλον. Η διαδικασία εύρεσης συγκρούσεων κατά τον σχεδιασμό διαδρομής δεν είναι απλή ενώ η πολυπλοκότητα αυξάνεται ακόμα περισσότερο για τους ρομποτικούς βραχίονες οι οποίοι μπορεί να έχουν συγκρούσεις και με τον εαυτό τους. Το αποτέλεσμα είναι ότι η υπορουτίνα εύρεσης συγκρούσεων συνήθως καταλαμβάνει τον περισσότερο υπολογιστικό χρόνο.

3.3.4 Υπορουτίνα βελτιστοποίησης διαδρομής

Καθώς η τελική διαδρομή αποτελεί αποτέλεσμα τυχαίας δειγματοληψίας στο χώρο, είναι πολύ πιθανό να είναι μη βέλτιστη και μη ομαλή ανά σημεία. Η υπορουτίνα βελτιστοποίησης διαδρομής μπορεί να ομαλοποιήσει δραστικά αυτή τη διαδρομή χρησιμοποιώντας συνήθως τεχνικές παρεμβολής (πολυωνυμικές, splines, κλπ.). Προσοχή χρειάζεται κατά την διάρκεια βελτιστοποίησης ώστε η νέα ομαλή διαδρομή να είναι εξίσου απαλλαγμένη από συγκρούσεις.

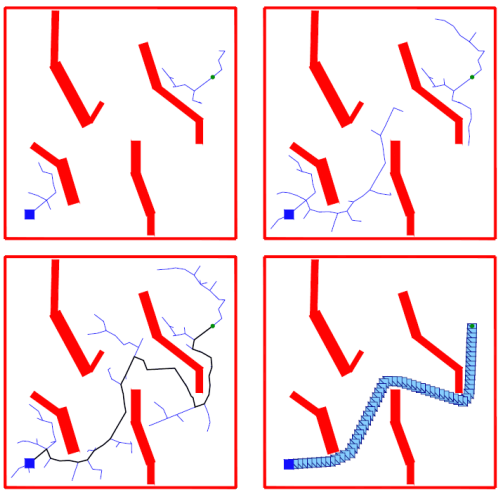


Εικόνα 7: Παράδειγμα διαδρομής: αρχική (κόκκινο), βελτιστοποιημένη (πράσινο) **[26]**

3.4 Παραλλαγές αλγορίθμου RRT

Υπάρχει μία συνεχής προσπάθεια βελτίωσης του αλγορίθμου RRT με σκοπό τη γρήγορη εύρεση μίας βέλτιστης διαδρομής κατά την διάρκεια σχεδιασμού. Έχει αποδειχθεί ότι κάτω από «κλασσικούς» περιορισμούς και συνθήκες, καθώς το κόστος εύρεσης της βέλτιστης διαδρομής αυξάνεται , ο αλγόριθμος RRT σχεδόν σίγουρα συγκλίνει σε μία μη βέλτιστη λύση. Πάνω σε αυτό το πρόβλημα, έχουν αναπτυχθεί πολλές παραλλαγές το αλγορίθμου κάποιες εκ των οποίων είναι οι εξής:

- **RRT-connect / bidirectional RRTs:** Έχοντας ως δεδομένο το αρχικό και το τελικό σημείο, το δέντρο αρχίζει και επεκτείνεται παράλληλα και από τα δύο αυτά σημεία δημιουργώντας έτσι δύο κλάδους. Σε συνδυασμό με τον ορισμό της κατεύθυνσης επέκτασης από τον ένα κλάδο στον άλλο, αυτού του είδους οι αλγόριθμοι μπορούν πολύ γρήγορα να φτάσουν σε λύση [27].

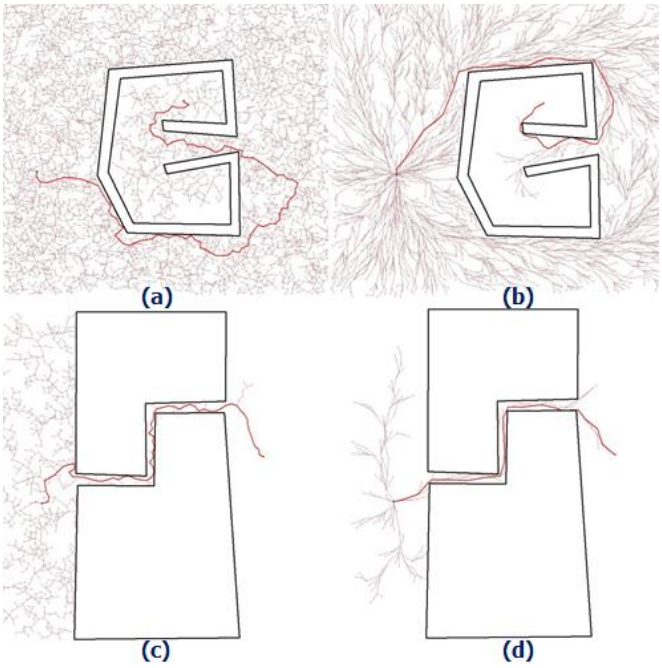


Σχήμα 8: RRT-connect

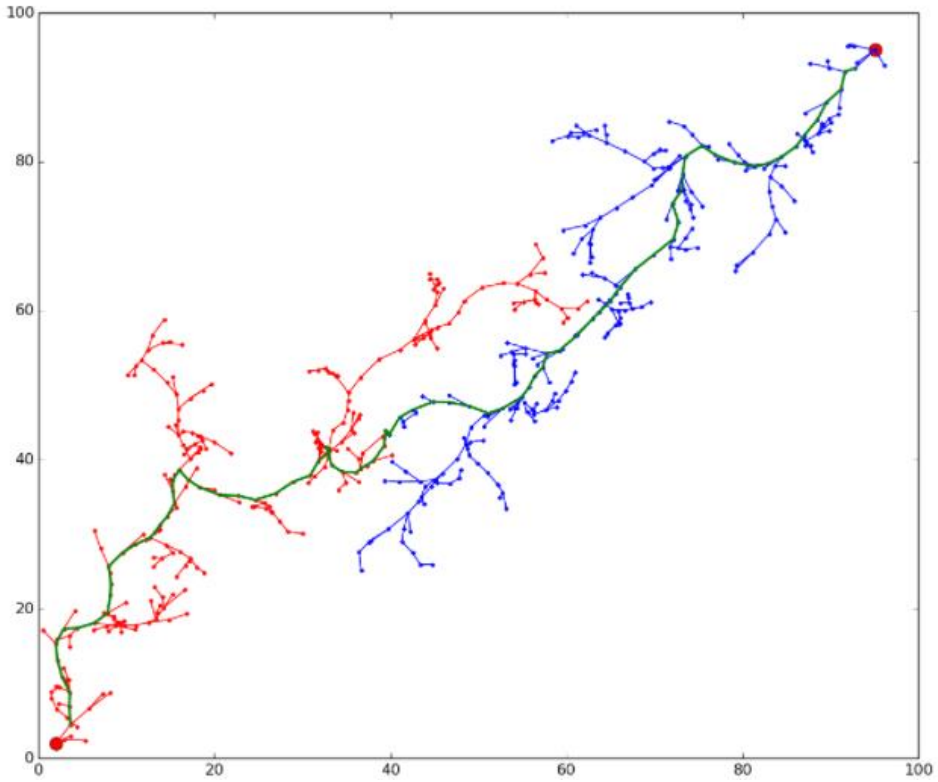
- **RRT***: Ο αλγόριθμος RRT* είναι σχεδόν ίδιος με τον RRT με την κύρια διαφορά να έγκειται στον τρόπο επέκτασης του δέντρου. Ο RRT* στοχεύει στη ελαχιστοποίηση της απόστασης των κόμβων του δέντρου μέσα

από την ρουτίνα επέκτασης και όχι βελτιστοποίησης της διαδρομής όπως στον RRT. Αποτέλεσμα είναι ο RRT* να προτιμάται και να επιστρέφει μία «καλύτερη» διαδρομή.

- **RRT*FN**: Μπορεί να θεωρηθεί ως παραλλαγή του RRT*. Βασικό χαρακτηριστικό του είναι ο περιορισμός του αριθμού των κόμβων που προστίθενται στο δέντρο και άρα η χρήση λιγότερης μνήμης και υπολογιστικών πόρων.
- **A*-RRT** και **A*-RRT***: Αποτελούν αλγορίθμους δύο φάσεων. Η πρώτη φάση αφορά τα προβλήματα μικρών διαστάσεων και την εύρεση μίας αρχικής πιθανής διαδρομής με τη χρήση αλγορίθμου με γράφους όπως ο A*. Η δεύτερη φάση αφορά τα προβλήματα μεγαλύτερων διαστάσεων όπου και χρησιμοποιείται ο αλγόριθος RRT/RRT*.
- Συνδυασμός των παραπάνω, κλπ.



Σχήμα 9: a) RRT σε παγίδα, b) RRT* σε παγίδα, c) RRT σε λαβύρινθο, d) RRT* σε λαβύρινθο **[28]**



Εικόνα 8: Bidirectional RRT **[29]**

Κεφάλαιο 4: Ελεγκτής σχεδιασμού κίνησης

4.1 Εισαγωγή

Η παρούσα εργασία επικεντρώνεται στον προγραμματισμό ενός δυναμικού ελεγκτή σχεδιασμού κίνησης (motion planner - controller), εστιάζοντας στο κομμάτι του σχεδιασμού διαδρομής με τη χρήση του απλού αλγορίθμου RRT μέσα στο εικονικό περιβάλλον του Unity. Η χρήση μίας πλατφόρμας όπως το Unity έχει πολλά πλεονεκτήματα καθώς εμπεριέχει έτοιμες βιβλιοθήκες για την αναπαράσταση όγκων και την αλληλεπίδραση μεταξύ τους. Πιο συγκεκριμένα οι συναρτήσεις ελέγχου συγκρούσεων, που αποτελούν βασικό κομμάτι του αλγορίθμου RRT, είναι ενσωματωμένες στο Unity επιτρέποντας έτσι στον προγραμματιστή να επικεντρωθεί στη δημιουργία του αλγορίθμου σχεδιασμού κίνησης του ρομπότ. Οι ενσωματωμένες δυνατότητες του Unity δεν περιορίζονται μόνο στον έλεγχο συγκρούσεων, αλλά επεκτείνονται με πολλαπλές βιβλιοθήκες που αφορούν γεωμετρικά χαρακτηριστικά, μαθηματικά μοντέλα και συναρτήσεις, σχεδιαστικά και γραφικά μοντέλα και μεθόδους γραφικής απεικόνισης του τελικού αποτελέσματος στις δύο (2D) ή και τρεις διαστάσεις (3D).

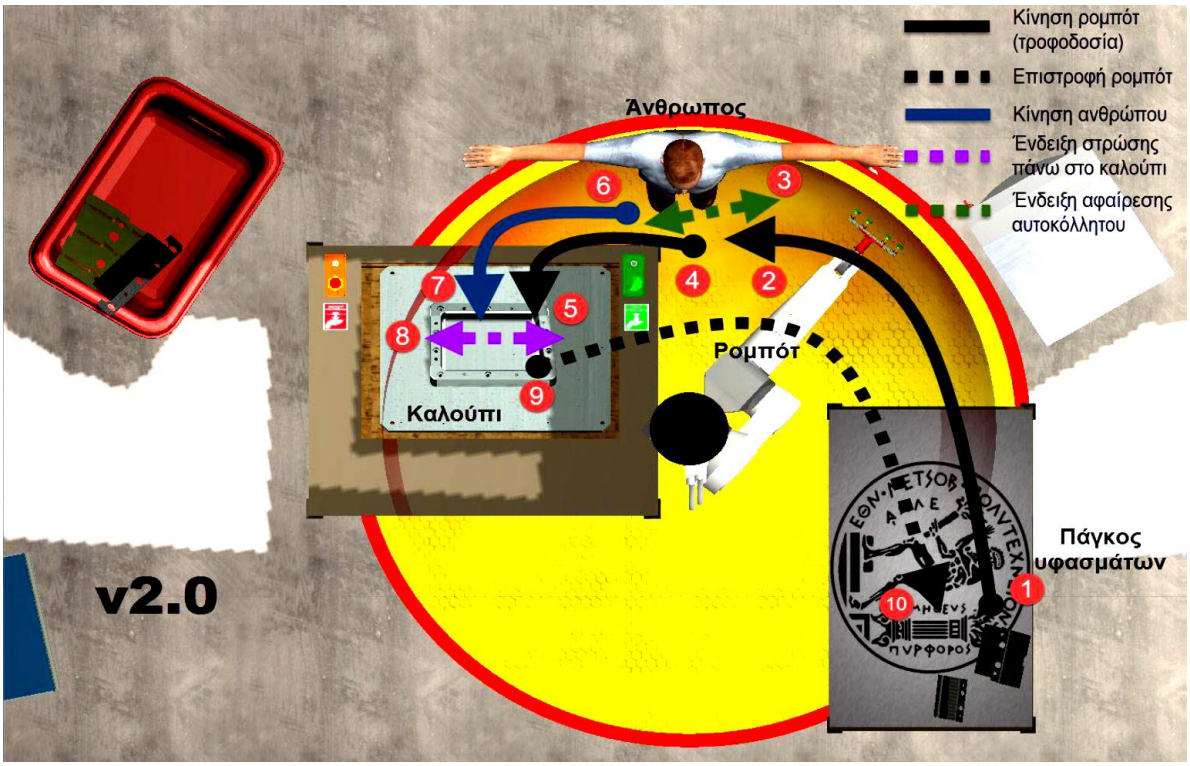
4.1.1 Ορισμός προβλήματος

Προτού ξεκινήσει ο προγραμματισμός του δυναμικού ελεγκτή απαιτείται ο ορισμός του προβλήματος προς λύση καθώς και καταγραφή του υπάρχοντος τρόπου διαχείρισής του.

Ήδη, και κυρίως σε βιομηχανικά περιβάλλοντα, η χρήση ρομποτικών βραχιόνων διέπεται από πολύ αυστηρά κριτήρια απόδοσης. Στόχος με την χρήση των ρομπότ είναι η αύξηση της

παραγωγής, η μείωση του κόστους, η μείωση των νεκρών χρόνων και η διατήρηση της ποιότητας. Εύκολα καταλαβαίνει κανείς λοιπόν, ότι σε ένα περιβάλλον συνεργασίας ανθρώπου ρομπότ αυτά τα κριτήρια δεν είναι δυνατό να διατηρηθούν σε αποδοτικά επίπεδα αν δεν υπάρχει ένας τρόπος ασφαλούς συνεργασίας. Ο παράγοντας «άνθρωπος» και «ασφάλεια» πάντα αποτελούν προτεραιότητα και διέπονται από αυστηρούς κανόνες.

Στο υπάρχον εικονικό περιβάλλον, ο χώρος εργασίας του ρομπότ και του ανθρώπου, πέρα από το κοινό κομμάτι που μοιράζονται (χώρος συνεργασίας), δεν είναι ούτε προκαθορισμένος ούτε χωρισμένος με ασφαλιστικές διατάξεις (μπάρες, πλέγματα κλπ). Αυτό σημαίνει πως ο άνθρωπος έχει μεν τη δυνατότητα να κινηθεί ελεύθερα αλλά πάντα με κίνδυνο την σύγκρουσή του με το ρομπότ πχ. «ανθρώπινο» λάθος. Οι υπάρχοντες μηχανισμοί αποφυγής συγκρούσεων περιορίζονται στην α) Επιβράδυνση του ρομπότ κατά την ανίχνευση πιθανής σύγκρουσης, β) Υποχώρηση του ρομπότ έναντι του ανθρώπου. Και οι δύο τεχνικές, μειώνουν δραστικά την απόδοση του ρομπότ προς όφελος της ασφάλειας του ανθρώπου ενώ μόνο το ευθύ κινηματικό μοντέλο του βραχίονα χρησιμοποιείται για την μετάβαση από σημείο σε σημείο (προκαθορισμένο πρόγραμμα).



Εικόνα 9:. Αρίθμηση εργασιών συνεργασίας ανθρώπου–ρομπότ με χρονική σειρά **[2]**

4.1.2 Στόχος του ελεγκτή σχεδιασμού κίνησης

Ο ελεγκτής σχεδιασμού κίνησης που θα προγραμματίσουμε πρέπει να παρέχει την ίδια - ή και ακόμα καλύτερη - ασφάλεια για τον άνθρωπο ενώ παράλληλα να διατηρεί την παραγωγικότητα του ρομπότ σε ικανοποιητικά επίπεδα. Αυτό απαιτεί προσεχτικό σχεδιασμό, αυστηρά κριτήρια και πολλές ώρες προσομοιώσεων ώστε να δοκιμαστούν όσο το δυνατό πιο πολλά πιθανά σενάρια σύγκρουσης.

Στόχος λοιπόν του ελεγκτή σχεδιασμού κίνησης είναι με σειρά προτεραιότητας:

1. Ασφάλεια ανθρώπου – Αποφυγή εμποδίων σε πραγματικό χρόνο
2. Εύρεση λύσης και ολοκλήρωση της εργασίας
3. Ταχύτητα εργασίας

Αντίστοιχα, για την επίτευξη των παραπάνω απαιτήσεων, ο αλγόριθμος που θα επιλέξουμε προς προγραμματισμό θα πρέπει:

1. Να σχεδιάζει διαδρομές ελεύθερες συγκρούσεων σε πραγματικό χρόνο. Απαιτείται μείωση της πιθανότητας σύγκρουσης στο 0% ακόμα και με δικλείδες ασφαλείας (πχ. σταμάτημα του ρομπότ στην περίπτωση μη εύρεσης λύσης).
2. Να μην κινδυνεύει από παγίδες και τοπικά ελάχιστα, βρίσκοντας έτσι πάντα λύση όπου αυτό είναι εφικτό.
3. Να έχει γρήγορη απόκριση σε αλλαγές του περιβάλλοντος και άρα να επιτρέπει μεγαλύτερες ταχύτητες εργασίας.

Ο αλγόριθμος RRT καλύπτει σε μεγάλο βαθμό τις παραπάνω απαιτήσεις. Παρότι κάποιες παραλλαγές του (RRT, RRT*FN) θεωρούνται πιο αποδοτικές, η απλότητα στην υλοποίησή του μας οδηγεί στην τελική επιλογή αυτού έναντι των άλλων.

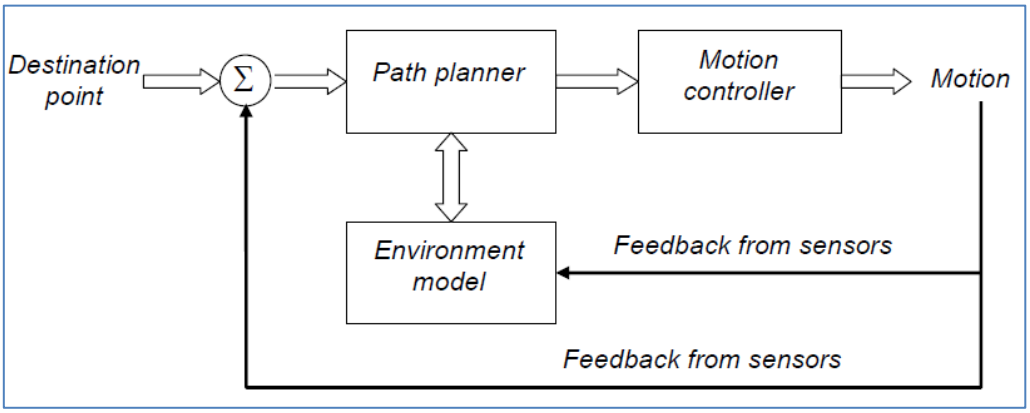
4.2 Αρχιτεκτονική κώδικα

Σημαντικό ρόλο στην απόδοση του αλγορίθμου αλλά και γενικά του προγραμματισμού ενός εικονικού περιβάλλοντος παίζει η αρχιτεκτονική του κώδικα. Ο συνολικός ελεγκτής σχεδιασμού κίνησης έχει χωριστεί σε διαφορετικά αρχεία (αρχεία .cs) ανάλογα με την

λειτουργία που επιτελούν με σκοπό την βελτιστοποίηση εκτέλεσης του κώδικα και την εύκολη «ανάγνωση» και διόρθωση του από τον χρήστη. Τα αρχεία που χρησιμοποιούνται είναι τα εξής :

1. RobotMover.cs
2. RobotController.cs
3. RRTController.cs
4. TrajectoryPlanner.cs
5. HelperFunctions.cs
6. DisplayEndEffectorCoordinates.cs

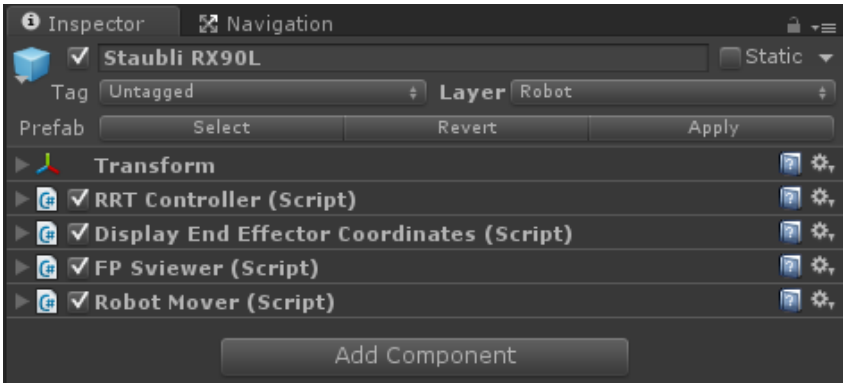
Η λογική που συνδέει όλα τα παραπάνω κομμάτια κώδικα, αποτελεί την αρχιτεκτονική του γενικού ρομποτικού ελεγκτή και παίζει σημαντικό ρόλο στην αποδοτική λειτουργία του. Ο σωστός τρόπος χρήσης της κάθε λειτουργίας και ο χρόνος στον οποίο καλούνται οι απαραίτητες συναρτήσεις μπορούν να μειώσουν δραστικά την άσκοπη χρήση υπολογιστικών πόρων και να αυξήσουν την απόκριση του συστήματος. Στο Σχήμα 10 παρουσιάζεται γραφικά το διάγραμμα ροής του ελεγκτή.



Σχήμα 10: Αρχιτεκτονική δυναμικού ελεγκτή σχεδιασμού κίνησης

Πατώντας πάνω στο αντικείμενο Staubli RX90L εμφανίζονται δεξιά στον inspector όλα τα scripts που αφορούν τον ρομποτικό βραχίονα (Εικόνα 10). Παρατηρείται ότι δεν εμφανίζονται όλα τα προαναφερόμενα αρχεία κώδικα. Αυτό συμβαίνει γιατί τα αντικείμενα(Objects) *TrajectoryPlanner.cs*, *RobotController.cs* και *HelperFunctions.cs* δεν κληρονομούν την συμπεριφορά mono του Unity. Σε αυτό το σημείο πρέπει να αναφερθεί

ότι η συμπεριφορά mono αποτελεί λειτουργία του Unity και παρέχει στα αντικείμενα έτοιμες συναρτήσεις και λειτουργίες. Χαρακτηριστικές συναρτήσεις mono είναι οι: Awake(), Start(), Update(), κλπ. Τα αντικείμενα που δεν κληρονομούν τη συμπεριφορά mono, καλούνται κατά τη διάρκεια εκτέλεσης του προγράμματος μόνο όταν απαιτηθεί, χωρίς έτσι να επιβαρύνουν υπολογιστικά το περιβάλλον.



Εικόνα 10: Αρχεία κώδικα με συμπεριφορά Mono

4.3 Περιγραφή αντικειμένων

Παρότι τα αντικείμενα(scripts) του ρομποτικού βραχίονα συνδέονται άμεσα μεταξύ τους, το κάθε ένα από αυτά επιτελεί μία διαφορετική λειτουργία απαραίτητη για τον σχεδιασμό κίνησης. Στη παρούσα ενότητα γίνεται μία πιο αναλυτική παρουσίαση του κάθε αντικειμένου με έμφαση στα αντίστοιχα κομμάτια κώδικα που κρίνονται σημαντικά.

4.3.1 RobotMover.cs

Το αντικείμενο *RobotMover.cs* αποτελεί τον κορμό του προγράμματος. Εδώ ορίζονται και αρχικοποιούνται όλες οι μεταβλητές-αντικείμενα που θα χρησιμοποιηθούν κατά την εκτέλεση όπως πχ. δημιουργία αντικειμένων, παράμετροι βραχίονα, μεταβλητές εισόδου και εξόδου του προγράμματος, κλήση των απαραίτητων συναρτήσεων, γραφική απεικόνιση μεταβλητών σε πραγματικό χρόνο, κλπ.

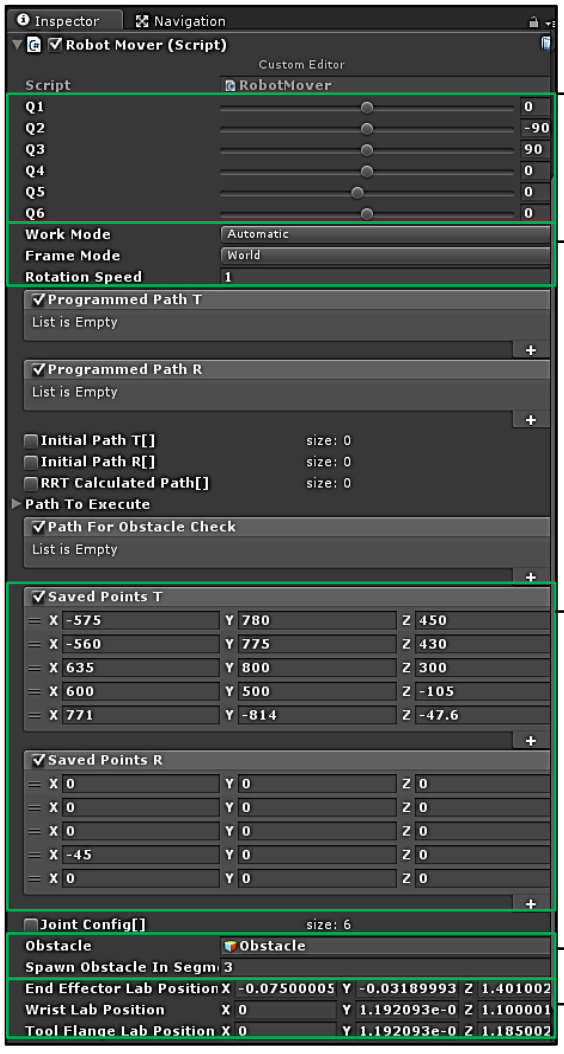
Με λίγα λόγια, το αντικείμενο *RobotMover.cs* ορίζει τη ροή του προγράμματος. Καθώς εκτελείται μέσα στην Update() συνάρτηση του Unity, είναι σημαντικό να γίνονται μόνο υπολογισμοί που αφορούν τη ροή του προγράμματος και όσο το δυνατόν λιγότεροι υπολογισμοί μαθηματικών μοντέλων. Υπάρχουν διάφορες μέθοδοι βελτιστοποίησης του

χρόνου εκτέλεσης των πράξεων όπως πχ. χρήση multithreading, παρόλα αυτά δεν αποτελούν αντικείμενο της παρούσας εργασίας.

4.3.1.1 Επιλογές – Ρυθμίσεις Χρήστη

Ανοίγοντας το μενού του αντικειμένου *RobotMover.cs* στον inspector του Unity, ο χρήστης έχει την επιλογή εισαγωγής διαφόρων ρυθμίσεων που αφορούν κυρίως την κίνηση του ρομπότ και το τρόπο εισόδου των εντολών (Εικόνα 11). Να υπενθυμίσουμε ότι οποιαδήποτε αλλαγή στον inspector που γίνεται κατά τη διάρκεια εκτέλεσης του προγράμματος, με τον τερματισμό του χάνεται και οι τιμές επιστρέφουν στις αρχικές προ εκτέλεσης.

Πέρα από τα πεδία εισαγωγής και ρύθμισης παραμέτρων του inspector, ο χρήστης έχει την επιτήρηση μεταβλητών του προγράμματος όπως πχ. τις συντεταγμένες των σημείων της διαδρομής, τις συντεταγμένες του τελικού σημείου δράσης, κλπ.



Τιμές των αρθρώσεων σε πραγματικό χρόνο

Επιλογή λειτουργίας κίνησης βραχίονα.
- **Automatic**: Χρήση ελεγκτή σχεδιασμού κίνησης
- **Manual**: Κίνηση βραχίονα από το χρήστη σε καρτεσιανές συντεταγμένες ή στο σύστημα αρθρώσεων.

Δυνατότητα εισαγωγής προκαθορισμένων σημείων στο χώρο για κίνηση στο καρτεσιανό σύστημα συντεταγμένων (Manual mode).

Ορισμός τμήματος για δημιουργία εμποδίου.

Συντεταγμένες αρθρώσεων σε πραγματικό χρόνο.

Εικόνα 11: Ρυθμίσεις αντικειμένου *RobotMover.cs*

4.3.1.2 Ανασκόπηση λειτουργιών

a) Λειτουργίες κίνησης: Ο χρήστης έχει την δυνατότητα ορισμού του τρόπου κίνησης του βραχίονα ανάλογα με της επιλογές *Work Mode* και *Frame Mode.* Πιο αναλυτικά:

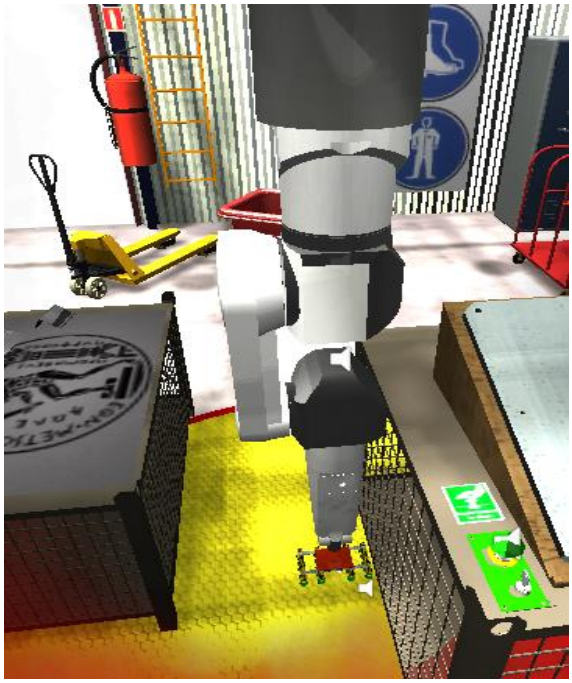
   a) Work Mode:

      - Automatic: Ο βραχίονας εκτελεί το προκαθορισμένο πρόγραμμα που έχει οριστεί μέσα στον κώδικα (πχ. εκτέλεση εργασίας) με πλήρως αυτόνομο τρόπο. Ενεργοποιείται ο ελεγκτής σχεδιασμού κίνησης.
      - Manual: Ο βραχίονας κινείται μόνο ύστερα από εντολή του χρήστη ανάλογα με το *Frame Mode* που έχει επιλεγεί.

   b) Frame Mode:

      - World: Ο χρήστης κινεί το τελικό σημείο του βραχίονα (end effector) σε καρτεσιανές συντεταγμένες μέσω των κουμπιών q,w,e,a,s,d για μετατοπίσεις X, Y, Z και o,p,k,l,n,m για περιστροφές γύρω από τους άξονες X, Y, Z αντίστοιχα.
      - Joint: Ο χρήστης κινεί την κάθε άρθρωση του βραχίονα μέσω της ρύθμισης της μπάρας ολίσθησης στον Inspector του Unity (Εικόνα 11).

c) Συντομεύσεις κίνησης: Κατά τη διάρκεια εκτέλεσης του προγράμματος ο χρήστης έχει τη δυνατότητα να αποθηκεύσει το σημείο που βρίσκεται το τελικό σημείο του robot για χρήση του σε μετέπειτα χρόνο. Με το πάτημα του κουμπιού «=» στο πληκτρολόγιο το σημείο αποθηκεύεται και οι συντεταγμένες του φαίνονται στον πίνακα *Saved Points T* και *Saved Points R* στον inspector. Επιπλέον με την χρήση του κουμπιού «space» ο βραχίονας επιστρέφει στην αρχική του θέση (home) (Εικόνα 12).



Εικόνα 12: Αρχική θέση ρομπότ (home position)

d) Έλεγχος απαίτησης χρήσης αντίστροφου κινηματικού μοντέλου: Κατά τη διάρκεια εκτέλεσης του προγράμματος και μόνο όταν ο χρήστης εισάγει μία νέα εντολή κίνησης

στο καρτεσιανό σύστημα συντεταγμένων, ενεργοποιείται ο ρομποτικός ελεγκτής (RobotController.cs) που περιέχει, πέραν άλλων, το αντίστροφο κινηματικό μοντέλο. Ο έλεγχος που γίνεται αφορά το σημείο στόχο (νέα εντολή) σε σχέση με το παρόν σημείο που βρίσκεται το τελικό άκρο του βραχίονα. Σε περίπτωση διαφοράς των τιμών τότε ενεργοποιείται το αντίστροφο κινηματικό μοντέλο και υπολογίζονται οι τιμές των γωνιών των αρθρώσεων που πρέπει να επιτύχει το ρομπότ. Τονίζεται πως ο υπολογισμός του αντίστροφου κινηματικού μοντέλου έχει μεγάλο υπολογιστικό κόστος και πρέπει να εκτελείται μόνο όταν είναι απαραίτητο.

e) Έλεγχος εμποδίων: Όταν το ρομπότ βρίσκεται στην λειτουργία Automatic, πρέπει να έχει 'επίγνωση' του περιβάλλοντος ώστε να αποφύγει τυχόν εμπόδια. Ο έλεγχος εμποδίων σε πρώτη φάση, γίνεται στο αρχικά προκαθορισμένο μονοπάτι που ακολουθεί ο βραχίονας προς εκτέλεση της εργασίας.

   Για τον εντοπισμό εμποδίων, το Unity παρέχει ορισμένες λειτουργίες εύρεσης συγκρούσεων οι οποίες είναι αρκετά χρήσιμες πχ. Linecast, CheckSphere, κλπ. Και εδώ, όπως με το αντίστροφο κινηματικό μοντέλο, απαιτείται ελαχιστοποίηση του απαραίτητου αριθμού ελέγχων για εμπόδια προς αποφυγή με στόχο τη μείωση του χρόνου εκτέλεσης του προγράμματος (frames per second - fps). Ας σημειωθεί, πως η τιμή fps υποδηλώνει πόσες φορές τρέχει ο κώδικας της συνάρτησης Update() σε ένα δευτερόλεπτο. Είναι προφανές ότι ο έλεγχος εμποδίων δεν έχει νόημα να εκτελείται πολλές φορές ανά δευτερόλεπτο καθώς εξετάζουμε ένα φυσικό σύστημα όπου οι μεταβολές γίνονται σε κλίμακα δευτερολέπτων και όχι κλασμάτων δευτερολέπτου. Για την εκτέλεση του ελέγχου εμποδίων σε προκαθορισμένη περίοδο και όχι συνεχόμενα, χρησιμοποιείται ο παρακάτω κώδικας:

```
if (Time.time > NextTime)
{
   NextTime = Time.time + PauseTime;
   ObstacleDetector (pathToExecute);
}
```

   όπου η μεταβλητή *PauseTime* ορίζει το χρόνο αναμονής μεταξύ δύο διαδοχικών ελέγχων εμποδίων.

Στη παρούσα εργασία ο χρόνος αναμονής ορίστηκε σε 1 δευτερόλεπτο. Προφανώς όσο πιο μεγάλη υπολογιστική ισχύ διατίθεται τόσο μπορεί να μειώνεται η τιμή αναμονής χωρίς να επηρεάζεται ο χρόνος εκτέλεσης του συνολικού προγράμματος.

Ο έλεγχος εμποδίων μπορεί να γίνει αρκετά πολύπλοκος και με μεγάλο υπολογιστικό κόστος. Παρατηρούμε ότι καλείται με όρισμα την προς εκτέλεση διαδρομή:

```
ObstacleDetector (pathToExecute);
```

Αυτό σημαίνει ότι σε κάθε κύκλο ελέγχου η διαδρομή μπορεί να διαφέρει με ποικίλους τρόπους. Πρέπει να ληφθεί υπόψη ότι μπορεί να μικραίνει και να αυξάνεται ταυτόχρονα λόγω της κάλυψης ενός τμήματος από τον βραχίονα αλλά και λόγω της προσθήκης άλλου τμήματος για αποφυγή τυχόν εμποδίων. Επιπλέον, πέρα από τον έλεγχο της διαδρομής, γίνεται έλεγχος για εμπόδια πάνω στα σημεία στόχους (μη ύπαρξη λύσης) και έλεγχος για συγκρούσεις των αρθρώσεων του βραχίονα.

f) Ενημέρωση - ανανέωση της διαδρομής και της τροχιάς: Σε περίπτωση εύρεσης εμποδίου ή σύγκρουσης, καλούνται οι ελεγκτές RRTController.cs και TrajectoryPlanner.cs οι οποίοι υπολογίζουν τη νέα διαδρομή και τροχιά του βραχίονα. Η νέα διαδρομή αποθηκεύεται σε ένα πίνακα 6D σημείων (μετατοπίσεις και περιστροφές) και με τη σειρά του αυτός ο πίνακας αντικαθιστά, στον αρχικό πίνακα της κύριας διαδρομής, το κομμάτι που αντιστοιχούσε στη σύγκρουση.
g) Απεικόνιση διαδρομής: Η προς εκτέλεση διαδρομή απεικονίζεται σε πραγματικό χρόνο μέσα στο εικονικό περιβάλλον.

4.3.2 RobotController.cs

Περιέχει το ευθύ και αντίστροφο κινηματικό μοντέλο του βραχίονα Staubli RX90L. Όπως προαναφέρθηκε, ο ρομποτικός ελεγκτής καλείται από άλλες συναρτήσεις μόνο όταν είναι απαραίτητος ο υπολογισμός των συντεταγμένων ενός σημείου στο χώρο (ευθύ μοντέλο) ή οι γωνίες των αρθρώσεων (αντίστροφο μοντέλο).

4.3.2.1 Ανασκόπηση λειτουργιών

1) Ευθύ κινηματικό μοντέλο: Το ευθύ κινηματικό μοντέλο του βραχίονα έχοντας γνωστές τις γωνίες των αρθρώσεων, υπολογίζει τις συντεταγμένες του τελικού σημείου δράσης. Η συνάρτηση που υλοποιήθηκε είναι η:

```
List<Vector4> FKcontroller(jointValues, desiredJointPos = 0)
```

όπου:

- Το όρισμα *jointValues* είναι απαραίτητο και αποτελεί ένα πίνακα με τις τιμές όλων των γωνιών των αρθρώσεων.

- Το όρισμα *desiredJointPos* είναι προαιρετικό και παίρνει τιμές από 1 μέχρι 6. Ο αριθμός αυτός υποδηλώνει την άρθρωση της οποίας θέλουμε να υπολογίσουμε τη θέση στο χώρο. Σε περίπτωση που ο χρήστης δεν δηλώσει κάποια άρθρωση, η προκαθορισμένη τιμή της μεταβλητής αυτής είναι 0 (μηδέν) και η συνάρτηση επιστρέφει της καρτεσιανές συντεταγμένες του τελικού σημείου δράσης. Υπενθυμίζεται, ότι για την εύρεση συγκρούσεων σε ολόκληρο το βραχίονα χρειάζεται να είναι γνωστή η θέση της κάθε άρθρωσης στο χώρο.

2) Αντίστροφο κινηματικό μοντέλο: Υπολογίζει τις τιμές γωνιών που πρέπει να έχει η κάθε άρθρωση με δεδομένο μόνο τις συντεταγμένες του τελικού σημείου δράσης. Η συνάρτηση είναι η:

```
List<float[]> IKcontroller(T, R,  mode=null)
```

Όπου

- *T* οι μετατοπίσεις του σημείου (X, Y, Z)

- *R* οι περιστροφές A, B, C γύρω από τους άξονες X, Y, Z αντίστοιχα

- *mode* ο τρόπος μετάβασης από το ένα σημείο στο επόμενο. Σε περίπτωση που επιθυμούμε η μετάβαση από σημείο σε σημείο να γίνεται σε μία ευθεία τότε *mode=LIN*.

3) Γραμμική παρεμβολή: Όταν η παράμετρος *mode* έχει οριστεί σε *LIN* τότε ο ρομποτικός ελεγκτής μέσω γραμμικής παρεμβολής υπολογίζει τις γωνίες των αρθρώσεων για κάθε διαδοχικό σημείο μέχρι και το τελικό. Σε σύγκριση με τον απλό υπολογισμό των αρθρώσεων στο τελικό σημείο-στόχο όπου το αντίστροφο κινηματικό μοντέλο τρέχει

μόνο μία φορά, η γραμμική μετάβαση από σημείο σε σημείο μπορεί να είναι υπολογιστικά ακριβή. Προς μείωση των περιττών υπολογισμών, μπορεί να χρησιμοποιηθεί η μέθοδος «περιορισμένης απόκλισης» (Bounded deviation method) η οποία περιγράφεται αναλυτικά στην επόμενη υποενότητα.

4) Κριτήρια επιλογής λύσεων αντίστροφου κινηματικού μοντέλου: Είναι γνωστό ότι μπορεί να υπάρχουν πολλαπλές λύσεις ή ακόμα και άπειρες κατά την επίλυση του αντίστροφου κινηματικού προβλήματος. Η επιλογή της τελικής λύσης προς εκτέλεση γίνεται με συγκεκριμένα κριτήρια από τον ρομποτικό ελεγκτή και ανάλογα με τη λειτουργία του την κάθε χρονική στιγμή.

   - Χειροκίνητη χρήση: Καθώς ο χρήστης μετακινεί το ρομπότ από μία θέση στην επόμενη, η επιλογή των τελικών γωνιών των αρθρώσεων γίνεται με βάση το ελάχιστο άθροισμα των τετραγώνων των διαφορών των γωνιών της θέσης-στόχου με τις προηγούμενες.

$$\sum_{i=1}^{6}\left(q_{i\_prev} - q_{i\_new}\right)^2$$

   Το κριτήριο αυτό, αποδίδει πολύ καλά όταν υπάρχει τουλάχιστον μία λύση όπου όλες οι τιμές των γωνιών είναι μέσα στα όρια των αρθρώσεων.

   - Αυτόματη χρήση: Κατά την εκτέλεση σε αυτόματη λειτουργία το παραπάνω κριτήριο ίσως να μην είναι αρκετό καθώς μπορεί να οδηγεί κάποια άρθρωση του βραχίονα στο όριό της. Εδώ εισάγεται το κριτήριο αποφυγής ορίων των αρθρώσεων ως αντικειμενική συνάρτηση της απόστασης από τα μηχανικά όρια των αρθρώσεων και ορίζεται ως [30]:

$$\Phi(q) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{q_i - \bar{q}_i}{q_{iM} - q_{im}}\right)^2$$

   Όπου:

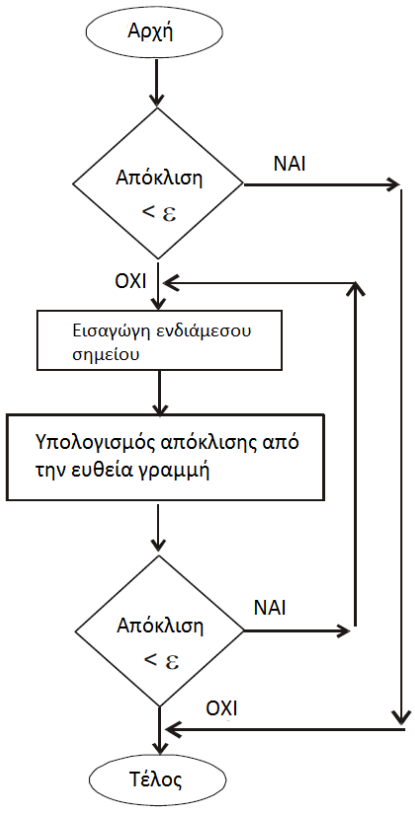
   - q_{iM}, q_{im} το μέγιστο και ελάχιστο όριο της άρθρωσης i
   - $\bar{q}_i$ η μέση τιμή της εμβέλειας της άρθρωσης i

   Επειδή, παρόλα αυτά, επιθυμούμε το ρομπότ να μεταβαίνει από την μία θέση στην επόμενη ομαλά, τελικά γίνεται χρήση και των δύο προαναφερόμενων κριτηρίων με τη χρήση βαρών.

4.3.2.2 Μέθοδος περιορισμένης απόκλισης – Bounded Deviation

Ο αλγόριθμος της περιορισμένης απόκλισης χρησιμοποιείται για την εκτέλεση μίας κατά προσέγγιση ευθύγραμμης κίνησης του τελικού σημείου δράσης ενός βραχίονα. Η μέθοδος προτάθηκε από τον Taylor [31] και μειώνει σημαντικά τους υπολογισμούς που απαιτούνται σε πραγματικό χρόνο, δημιουργώντας μόνο όσα ενδιάμεσα σημεία χρειάζονται ώστε η απόκλιση της προσεγγιστικής ευθύγραμμης κίνησης του τελικού σημείου δράσης του βραχίονα από την πραγματική ευθεία να μην υπερβαίνει ένα προκαθορισμένο όριο.

Έχει παρατηρηθεί ότι η απόκλιση πιθανότατα αποκτά την μεγαλύτερη τιμή της στο μέσο της απόστασης που πρέπει να διανυθεί. Ο αλγόριθμος περιορισμένης απόκλισης ξεκινάει με τον υπολογισμό της απόκλισης στο μέσο και σε περίπτωση που αυτή υπερβαίνει μια τιμή (σφάλμα) ε, προσθέτει ένα ενδιάμεσο σημείο. Η διαδικασία επαναλαμβάνεται για όλα τα καινούρια ευθύγραμμα τμήματα που δημιουργούνται και μέχρι η μετάβαση από το αρχικό στο τελικό σημείο να μην αποκλίνει περισσότερο από το σφάλμα ε. Παρακάτω παρουσιάζεται και το διάγραμμα ροής (Σχήμα 11), καθώς επίσης ακολουθεί η αναλυτική περιγραφή του αλγορίθμου.



Σχήμα 11: Διάγραμμα ροής αλγορίθμου περιορισμένης απόκλισης

Βήματα αλγορίθμου περιορισμένης απόκλισης:

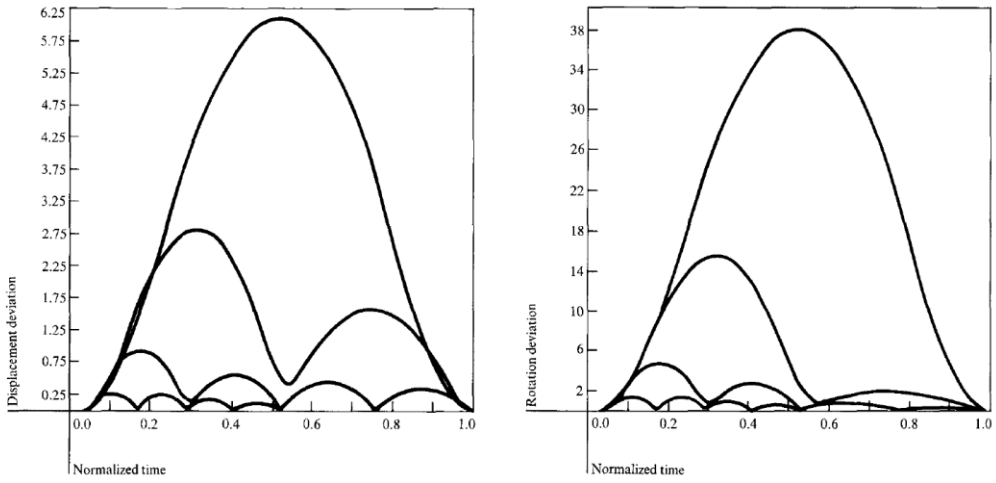
1) Επιλογή της τιμής της μέγιστης επιτρεπτής απόκλισης – σφάλμα ε > 0.
2) Δεδομένων του αρχικού (P_0) και τελικού σημείου (P_1) και με χρήση του αντίστροφου κινηματικού μοντέλου υπολογίζουμε τις τιμές των αρθρώσεων για το κάθε σημείο, Q_0 και Q_1 αντίστοιχα όπου Q πίνακας που περιέχει τις τιμές όλων των αρθρώσεων ($\{q_1,q_2,q_3,q_4,q_5,q_6\}$).
3) Υπολογισμός του μέσου σημείου στο χώρο των αρθρώσεων:

$$Q_m = \frac{Q_0 + Q_1}{2}$$

4) Υπολογισμός των καρτεσιανών συντεταγμένων του τελικού σημείου δράσης για τις τιμές αρθρώσεων Q_m, με χρήση του ευθύ κινηματικού μοντέλου. Έστω P_m.
5) Υπολογισμός του πραγματικού μέσου σημείου:

$$P_M = \frac{P_0 + P_1}{2}$$

6) Αν η διαφορά $\|P_M - P_m\| \leq \varepsilon$ τότε τέλος αλγορίθμου.
7) Αν η διαφορά $\|P_M - P_m\| \geq \varepsilon$ τότε εισαγωγή του σημείου P_M σαν ενδιάμεσου και δημιουργία δύο νέων ευθύγραμμων τμημάτων $\{P_0, P_M\}$ και $\{P_M, P_1\}$.
8) Επανάληψη βημάτων 1 έως 6 μέχρι όλα τα νέα ευθύγραμμα τμήματα να έχουν απόκλιση μικρότερη του ε.



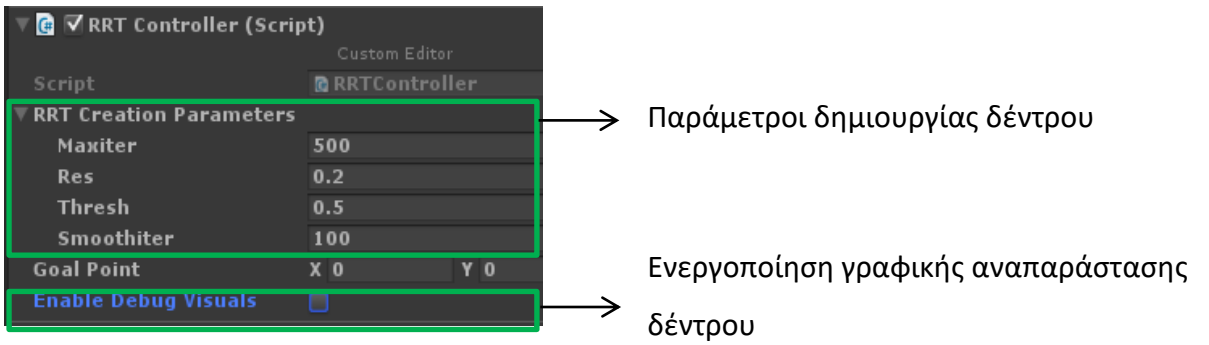
Σχήμα 12: Σφάλματα μετατόπισης (αριστερά) και περιστροφής (δεξιά) σε ένα ευθύγραμμο τμήμα με παρεμβολή ενδιάμεσων σημείων 0, 1, 2 και 3 επιπέδων **[31]**.

4.3.3 RRTController.cs

Ο ελεγκτής RRT περιέχει όλες τις απαραίτητες συναρτήσεις για τη δημιουργία του δέντρου και την εύρεση μίας διαδρομής χωρίς συγκρούσεις. Καλείται από το κυρίως πρόγραμμα μόνο όταν ανιχνευθεί εμπόδιο στην αρχικά προκαθορισμένη διαδρομή ή σε κάποια από τις αρθρώσεις του βραχίονα.

4.3.3.1 Επιλογές – Ρυθμίσεις Χρήστη

Επιλέγοντας στον Inspector τον ελεγκτή RRT ο χρήστης μπορεί να εισάγει τις παραμέτρους δημιουργίας του δέντρου καθώς επίσης και να ενεργοποιήσει την γραφική αναπαράσταση του δέντρου στο 3D εικονικό περιβάλλον. Οι διαθέσιμες ρυθμίσεις φαίνονται στην Εικόνα 13.



Εικόνα 13: Ρυθμίσεις ελεγκτή RRT

4.3.3.2 Ανασκόπηση λειτουργιών

1) Κλάσεις δέντρου και κόμβου: Ένα δέντρο αποτελείται από κόμβους οι οποίοι ενώνονται συνήθως με βάση το κριτήριο ελαχιστοποίησης της απόστασης μεταξύ τους. Για την δημιουργία του δέντρου και της τελικής διαδρομής, ο κάθε κόμβος αποτελεί μόνος του μία δομή (struct) η οποία περιέχει πληροφορίες.

   - Δομή δέντρου: Το δέντρο περιέχει δύο λίστες. Μία με όλους τους κόμβους που έχουν προστεθεί κατά την διαδικασία επέκτασής του και μία μόνο με τους κόμβους που απαρτίζουν την τελική διαδρομή που έχει επιλεγεί. Η δεύτερη λίστα είναι προφανώς υποσύνολο της πρώτης.

   - Δομή κόμβου: Ένας κόμβος περιέχει «τοπική» πληροφορία που αφορά α) τις συντεταγμένες του στο χώρο και β) τον αριθμό του προηγούμενου κόμβου με τον οποίο

ενώνεται. Σημειώνεται η αύξουσα αρίθμηση των κόμβων που προστίθενται στο δέντρο ξεκινώντας από το μηδέν.

2) Παράμετροι δημιουργίας δέντρου: Ο χρήστης έχει τη δυνατότητα να ορίσει βασικές παραμέτρους της δημιουργίας του RRT επηρεάζοντας άμεσα την ανάλυση και χρονική απόκριση του αλγορίθμου. Πιο αναλυτικά:

   - Παράμετρος **Maxiter**: Ορίζει τον αριθμό των μέγιστων επαναλήψεων προσπάθειας επέκτασης του δέντρου. Ανάλογα με το περιβάλλον και τα χαρακτηριστικά του προβλήματος (πχ. όγκος χώρου προς εξερεύνηση, μέγεθος εμποδίων, κλπ.) ο αριθμός αυτός πρέπει να είναι ικανοποιητικά μεγάλος ώστε ο αλγόριθμος να μην τερματίζει πρόωρα. Από την άλλη, η συχνή εξάντληση του ορίου αυτού μπορεί να υποδηλώνει τη χρήση μη βέλτιστων τιμών για τις επόμενες παραμέτρους.

   - Παράμετρος **Res**: Ορίζει την ακτίνα της σφαίρας του κάθε κόμβου. Καθώς αναφερόμαστε σε ένα φυσικό πρόβλημα όπου τα εμπόδια έχουν όγκο και δεν είναι σημειακά, έτσι και ο κάθε κόμβος που δημιουργείται αποτελεί έναν όγκο σφαίρας ακτίνας R. Ανάλογα με το μέγεθος των εμποδίων προς αποφυγή, ο χρήστης μπορεί να επηρεάσει άμεσα την ανάλυση του συστήματος ορίζοντας την παράμετρο αυτή.

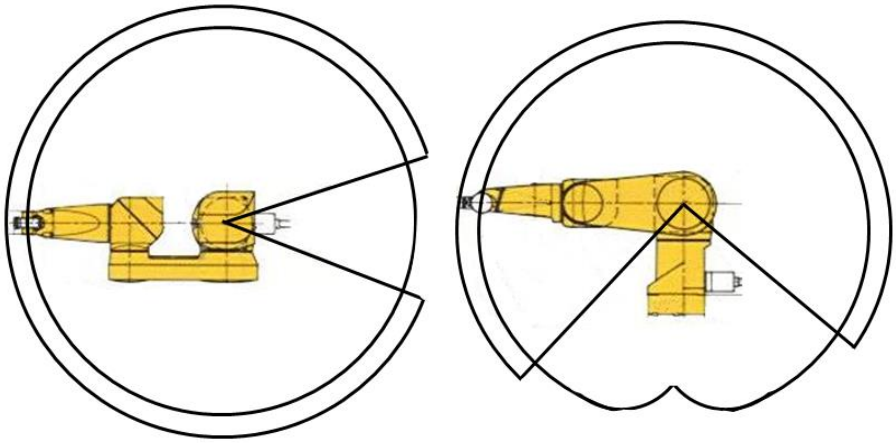
   - Παράμετρος **Thresh**: Καθώς είναι γνωστό το τελικό σημείο, για να επιταχύνουμε την εκτέλεση του αλγορίθμου, ορίζουμε μία μέγιστη απόσταση μεταξύ του πιο πρόσφατου τυχαίου κόμβου του δέντρου και του κόμβου στόχου, κάτω από την οποία ο αλγόριθμος τερματίζει επιτυχώς. Προφανώς αυτή η απόσταση δεν πρέπει να έχει μεγάλη τιμή καθώς αυξάνονται οι πιθανότητες ύπαρξης εμποδίων στο ενδιάμεσο. Παρόλα αυτά, ο αλγόριθμος κάνει έναν τελικό έλεγχο προτού επιβεβαιώσει τον επιτυχή τερματισμό. Η χρήση αυτής της παραμέτρου, που στην ουσία δημιουργεί μια σφαίρα ακτίνας *thresh* γύρω από κάθε κόμβο, μειώνει δραστικά το χρόνο εκτέλεσης του προγράμματος καθώς «παρακάμπτει» το δειγματοληπτικό χαρακτήρα του αλγορίθμου κοντά στο σημείο-στόχο.

   - Παράμετρος **Smoothiter**: Ορίζει το μέγιστο αριθμό των επαναλήψεων προσπάθειας επέκτασης του αλγορίθμου. Αυτή η παράμετρος είναι κυρίως απαραίτητη για τον τερματισμό του αλγορίθμου σε περίπτωση μη ύπαρξης λύσης. Όπως αναφέραμε στο προηγούμενο κεφάλαιο, ο αλγόριθμος RRT αδυνατεί να εντοπίσει την μη ύπαρξη λύσης και άρα είναι απαραίτητος ο προκαθορισμένος τερματισμός του. Μία βέλτιστη τιμή αυτής της παραμέτρου εξαρτάται από τις προηγούμενες παραμέτρους και συνήθως

ορίζεται πειραματικά (πχ. μέγιστος αριθμός προσπαθειών επέκτασης σε περίπτωση ύπαρξης λύσης).

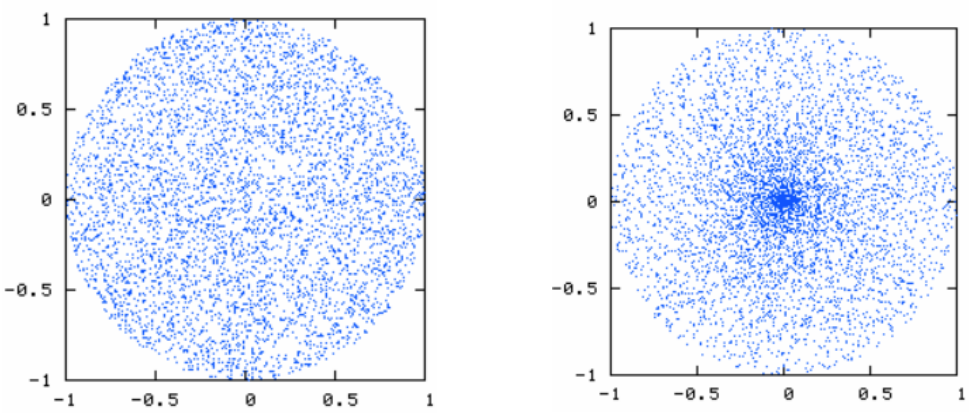
3) Αλγόριθμος δειγματοληψίας σημείων και φιλτράρισμα: Η δειγματοληψία σημείων κατά την επέκταση του δέντρου γίνεται με τη χρήση αλγορίθμου τυχαιότητας. Καθώς ο χώρος εργασίας πλήρους επιδεξιότητας του βραχίονα πλησιάζει αυτόν μιας σφαίρας (Σχήμα 13), ο χώρος δημιουργίας των τυχαίων σημείων μπορεί να περιορισθεί σημαντικά. Το πρόβλημα ανάγεται πια σε τυχαία δημιουργία κόμβων μέσα στον όγκο σφαίρας με γνωστή ακτίνα (όσο το μήκος του βραχίονα πλήρως εκτεταμένου). Η συνάρτηση που χρησιμοποιείται είναι η:

SphereVector3Randomizer(Vector3 Center, float Radius)

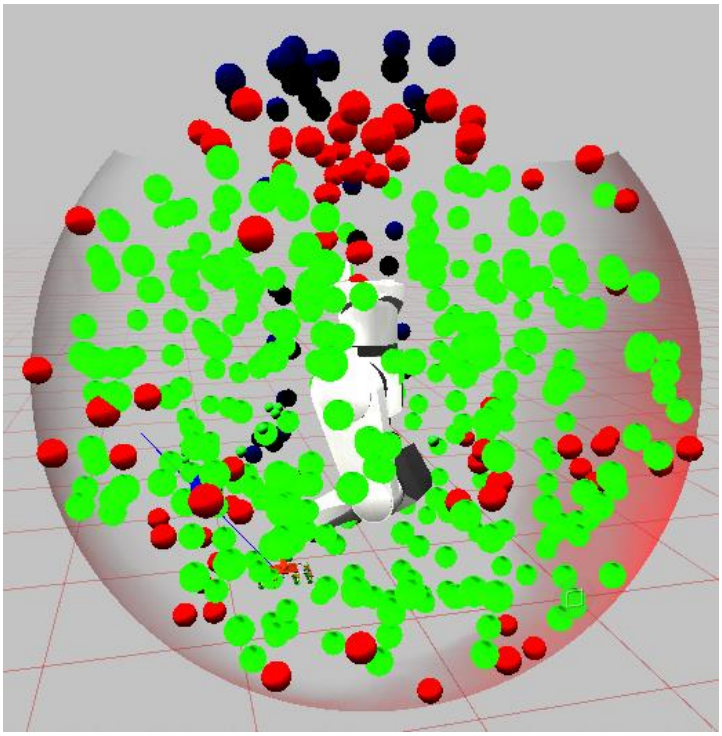


Σχήμα 13: Κάτοψη (αριστερά) και πλάγια όψη (δεξιά) χώρου εργασίας βραχίονα

Είναι σημαντικό να αναφερθεί πως ο αλγόριθμος δειγματοληψίας πρέπει να υιοθετεί ομοιόμορφη κατανομή στην επιλογή των κόμβων, ώστε να αυξηθεί η απόδοση του αλγορίθμου RRT. Στο Σχήμα 14 φαίνεται η διαφορά της ομοιόμορφης κατανομής από μία με συσσώρευση στο κέντρο. Στην Εικόνα 14 φαίνεται η κατανομή που δημιουργήθηκε από τον δειγματοληπτικό αλγόριθμο.



Σχήμα 14: Ομοιόμορφη κατανομή (αριστερά) και κεντρικά συσσωρευμένη (δεξιά) **[32]**



Εικόνα 14: Κατανομή κόμβων του δειγματοληπτικού αλγορίθμου που χρησιμοποιήθηκε

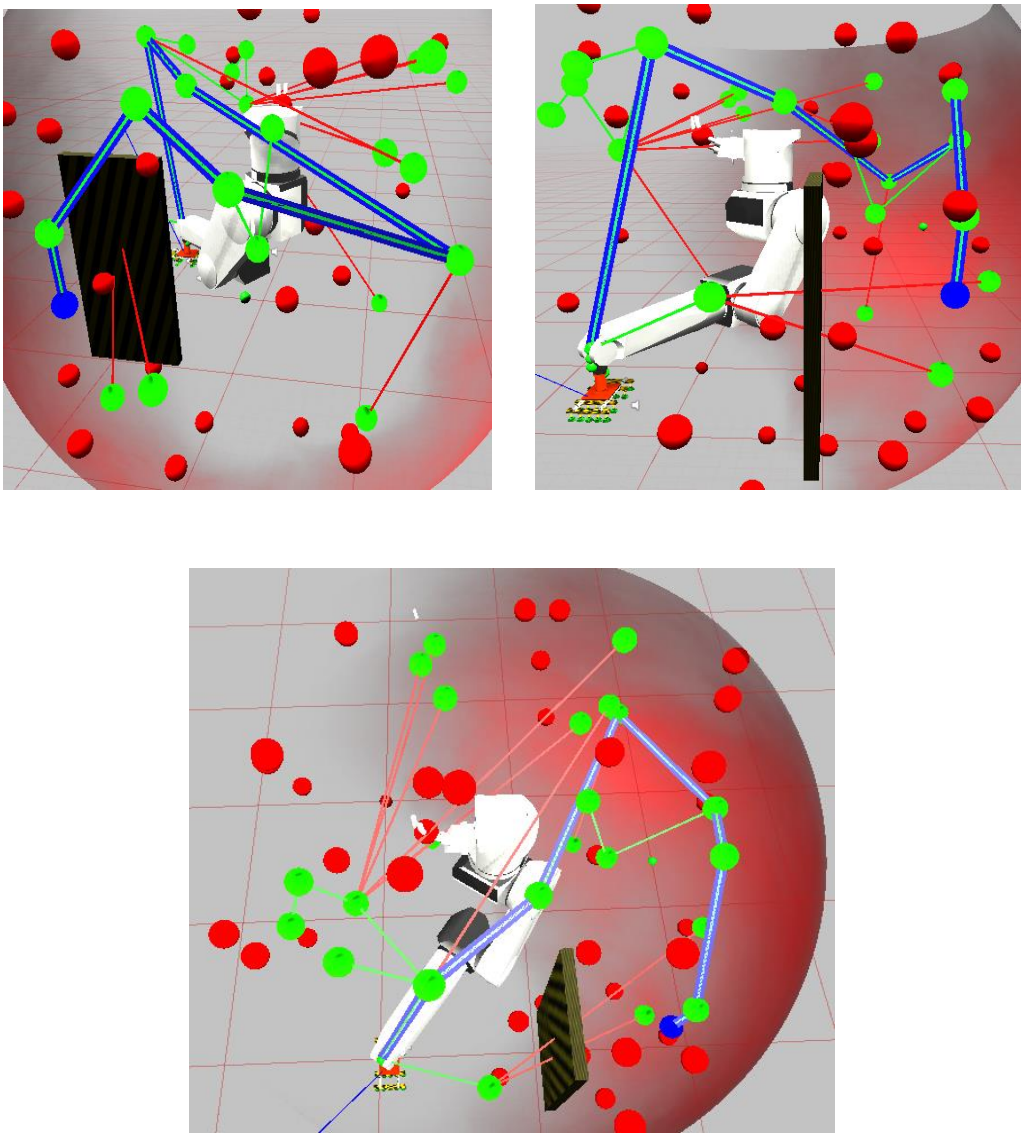
4) <u>Δημιουργία δέντρου:</u> Για τη δημιουργία του δέντρου, όπως προαναφέραμε, χρησιμοποιείται ο απλός αλγόριθμος RRT. Για την εύρεση συγκρούσεων χρησιμοποιούνται συναρτήσεις του Unity και πιο συγκεκριμένα:

   a) *CheckSphere()* για τους κόμβους.

   b) *Linecast()* για την ένωση των κόμβων και τον έλεγχο του χώρου μεταξύ τους.

   Η χρήση αυτών των βασικών γεωμετρικών σχημάτων, παρόλο που μειώνει την ακρίβεια των γεωμετρικών μεγεθών (πχ. ορθογώνιο εργαλείο ρομπότ), επιτυγχάνει ραγδαία μείωση στη χρήση υπολογιστικών πόρων και διευκολύνει στον αρχικό, τουλάχιστον, προγραμματισμό του περιβάλλοντος. Για παράδειγμα, η χρήση ενός ορθογώνιου

πλέγματος για εύρεση συγκρούσεων θα απαιτούσε έλεγχο και για περιστροφές (στους τρεις άξονες) του όγκου αυτού και άρα αύξηση του αριθμού των ελέγχων για κάθε κόμβο από έναν σε, ίσως, δεκάδες. Γενικά, η αντικατάσταση πολύπλοκων πλεγμάτων με βασικά σχήματα πλέγματος προσφέρει πολλά πλεονεκτήματα με μικρό συνήθως κόστος σε ακρίβεια.



Εικόνα 15: Διάφορες οπτικές γωνίες ενός απλοποιημένου RRT δέντρου

Στην Εικόνα 15 δίνεται ένα απλοποιημένο σενάριο δημιουργίας δέντρου και επεξηγείται παρακάτω. Έστω ότι στόχος του βραχίονα είναι η κατάληξη του τελικού σημείου δράσης του στον μπλε κόμβο. Ανάμεσα υπάρχει ένα εμπόδιο και ο βραχίονας δεν μπορεί να εκτελέσει επιτυχώς την, σε αντίθετη περίπτωση, ευθύγραμμη κίνηση προς το στόχο. Για το δέντρο που δημιουργείται έχουμε:

- **Μπλε σφαίρα**: Κόμβος – στόχος.

- **Κόκκινη σφαίρα**: Κόμβος που βρίσκεται σε σύγκρουση ή στο όριο του χώρου εργασίας πλήρους επιδεξιότητας.

- **Πράσινη σφαίρα**: Κόμβος που είναι ελεύθερος συγκρούσεων και υποψήφιος για προσθήκη στο δέντρο.

- **Κόκκινη γραμμή**: Αδυναμία ένωσης κόμβου με το δέντρο.

- **Πράσινη γραμμή**: Ένωση κόμβου με το δέντρο.

- **Μπλε γραμμή**: Τελική διαδρομή.

Σημειώνεται ότι, παρόλο που η τελική διαδρομή είναι ελεύθερη συγκρούσεων, απέχει πολύ από το να είναι βέλτιστη.

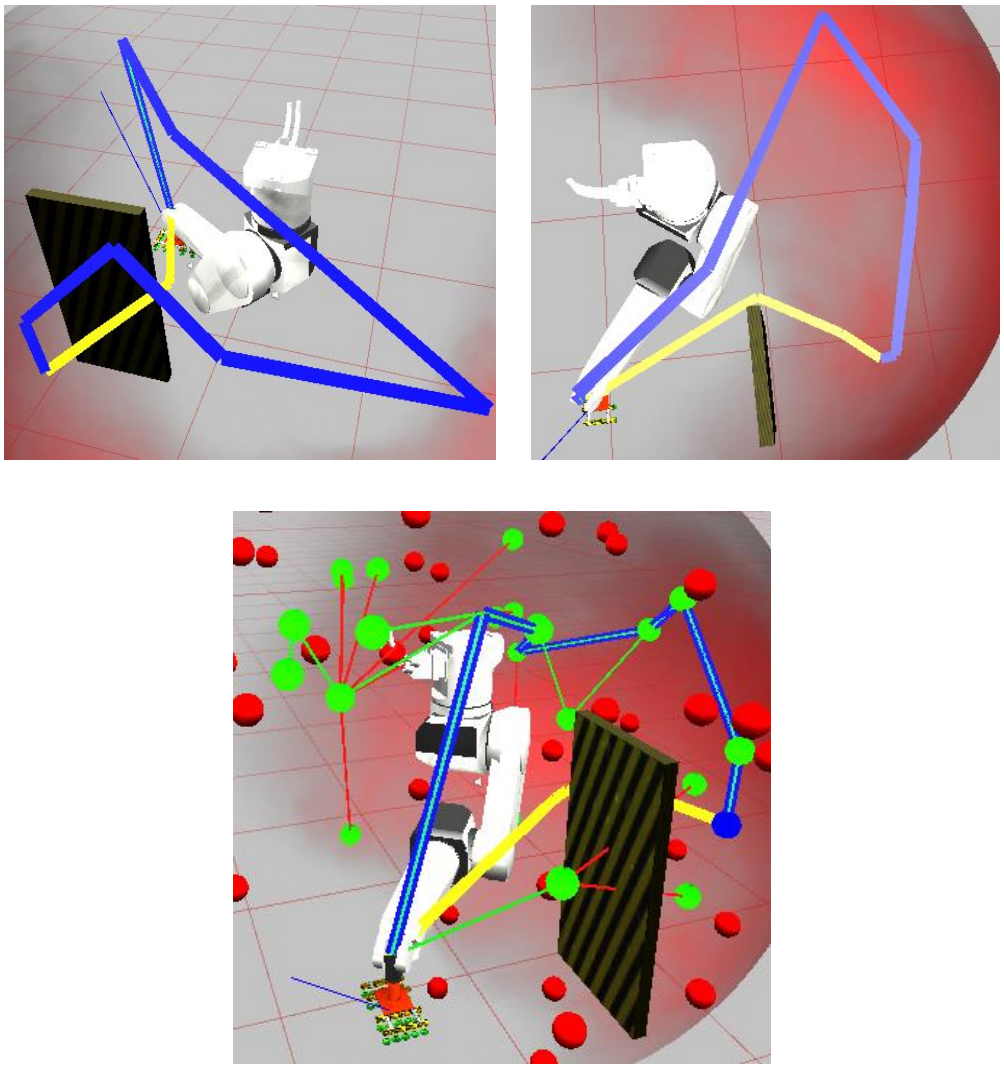
5) Βελτιστοποίηση διαδρομής: Επόμενο βήμα του *RRTController.cs* είναι η βελτιστοποίηση της διαδρομής, ώστε να μπορεί να εκτελεσθεί από τον βραχίονα. Σύνηθες κριτήριο βελτιστοποίησης είναι η ελαχιστοποίηση της απόστασης που πρέπει να διανυθεί και η μείωση των απότομων «στροφών» (αλλαγών κατεύθυνσης).

Ο αλγόριθμος βελτιστοποίησης είναι στην ουσία μία επαναληπτική μέθοδος γραμμικής παρεμβολής όπου επιλέγονται τυχαία σημεία μεταξύ κόμβων και γίνεται έλεγχος αν μπορούν να ενωθούν σε έναν για να μειώσουν το συνολικό μήκος της διαδρομής. Χρειάζεται μεγάλη προσοχή ώστε να γίνεται έλεγχος εμποδίων σε κάθε επανάληψη του αλγορίθμου. Επίσης, είναι συνήθως ανεπιθύμητο η διαδρομή να είναι πολύ κοντά σε εμπόδιο και για αυτό ορίζεται μία νοητή σφαίρα ακτίνας r της οποίας το κέντρο κινείται πάνω στη διαδρομή. Ο αλγόριθμος τερματίζει όταν η σφαίρα αυτή έρθει σε επαφή με κάποιο αντικείμενο και έτσι εγγυάται πως η απόσταση της τελικής βελτιστοποιημένης διαδρομής από οποιοδήποτε εμπόδιο θα είναι κατ' ελάχιστο r.

Στην Εικόνα 16 φαίνεται το αποτέλεσμα του αλγορίθμου βελτιστοποίησης για το προηγούμενο παράδειγμα δημιουργίας της διαδρομής από τον αλγόριθμο RRT. Έχουμε:

- **Μπλε γραμμή**: Αρχική διαδρομή μη βελτιστοποιημένη.

- **Κίτρινη γραμμή**: Τελική βελτιστοποιημένη διαδρομή έτοιμη προς εκτέλεση από τον βραχίονα.



Εικόνα 16: Βελτιστοποιημένη τελική διαδρομή (κίτρινη γραμμή)

6) Ενεργοποίηση οπτικής αναπαράστασης της δημιουργίας του δέντρου: Ο χρήστη έχει τη δυνατότητα οπτικής αναπαράστασης της δημιουργίας του δέντρου ώστε να έχει εποπτεία κατά την εκτέλεση του προγράμματος. Με την ενεργοποίηση της επιλογής *Enable Debug Visuals* στον inspector του Unity, ενεργοποιούνται τα γραφήματα των εικόνων 15 και 16. Προτείνεται η χρήση αυτής της λειτουργίας να γίνεται μόνο όταν είναι απαραίτητη καθώς έρχεται με μεγάλο υπολογιστικό κόστος.

4.3.4 TrajectoryPlanner.cs

Έχοντας υπολογίσει μία διαδρομή μετάβασης στον τελικό στόχο χωρίς συγκρούσεις, ο βραχίονας πρέπει να γνωρίζει τη θέση όχι μόνο του τελικού σημείου δράσης αλλά και όλων των υπόλοιπων αρθρώσεων του. Με άλλα λόγια πρέπει να γίνει ο σχεδιασμός τροχιάς του βραχίονα που από μόνος του αποτελεί ένα πολύπλοκο πρόβλημα και τεράστιο τομέα

έρευνας. Στα πλαίσια της παρούσας εργασίας έμφαση δόθηκε στο κομμάτι του σχεδιασμού διαδρομής ενώ ο ελεγκτής σχεδιασμού τροχιάς *TrajectoryPlanner.cs* περιέχει μόνο βασικές λειτουργίες ελέγχου δυνατότητας εκτέλεσης της διαδρομής και όχι εύρεσης λύσης.

4.3.4.1 Ανασκόπηση λειτουργιών

1) Έλεγχος συγκρούσεων λοιπών αρθρώσεων βραχίονα: Βασική λειτουργία του ελεγκτή είναι ο έλεγχος συγκρούσεων όλων των αρθρώσεων του βραχίονα με το περιβάλλον. Κάθε σημείο της διαδρομής περιέχει σαν πληροφορία όλες τις αποδεκτές λύσεις του αντίστροφου κινηματικού μοντέλου και είναι στη διάθεση του ελεγκτή σχεδιασμού τροχιάς να ελέγξει και να επιλέξει την βέλτιστη ανάλογα με τα κριτήρια που έχουν τεθεί. Πέρα από τις συγκρούσεις ο ελεγκτής πρέπει παράλληλα να σχεδιάζει ομαλές τροχιές και να μην οδηγεί τις αρθρώσεις του βραχίονα σε όρια ή ιδιομορφίες κατά μήκος όλης της διαδρομής. Σε περίπτωση μη εύρεσης λύσης ο ελεγκτής σχεδιασμού τροχιάς πρέπει να ενεργοποιεί τον ελεγκτή σχεδιασμού διαδρομής για εκ νέου σχεδιασμό της διαδρομής στα σημεία του προβλήματος. Το πρόβλημα αυτό είναι αρκετά πολύπλοκο και δεν αναλύεται στην παρούσα εργασία.
2) Προσομοίωση θέσης-πόζας βραχίονα πριν την εκτέλεση: Ο σχεδιασμός τροχιάς απαιτεί προ-υπολογισμό της θέσης του βραχίονα κατά μήκος της διαδρομής, καθώς η μετάβαση στο επόμενο σημείο είναι αλληλένδετη με τον γεωμετρία του βραχίονα στην αμέσως προηγούμενη. Εν τέλει, ο σχεδιασμός τροχιάς απαιτεί την εποπτεία της γεωμετρίας του βραχίονα κατά μήκος όλης της διαδρομής προς εκτέλεση.

4.3.5 HelperFunctions.cs

Το αντικείμενο *HelperFunctions.cs* περιέχει δομές, κλάσεις και συναρτήσεις που χρησιμοποιούνται επανειλημμένα από όλα τα υπόλοιπα αντικείμενα – ελεγκτές.

4.3.5.1 Ανασκόπηση λειτουργιών

1) Δομές: Το κάθε σημείο – κόμβος είναι στην ουσία μία δομή που περιέχει πληροφορίες. Στη ρομποτική, όταν αναφερόμαστε σε σημεία υπάρχουν πολλαπλοί τρόποι περιγραφής τους ανάλογα με το σύστημα αναφοράς. Έτσι έχουν

δημιουργηθεί δομές για το καρτεσιανό σύστημα συντεταγμένων αλλά και το σύστημα αρθρώσεων του ρομπότ.

2) Κλάσεις: Η διαδρομή προς εύρεση είναι μία κλάση. Αποτελείται από επιμέρους τμήματα τα οποία ανήκουν στη δική τους κλάση και με τη σειρά τους αποτελούνται από σημεία που πάλι έχουν τη δική τους κλάση αλλά και δομή όπως αναφέρθηκε προηγουμένως.
3) Συναρτήσεις: Υπάρχουν συναρτήσεις που χρησιμοποιούνται σε διαφορετικά αρχεία και σημεία του κώδικα και έχουν δημιουργηθεί για την μαθηματική –συνήθως- επίλυση προβλημάτων. Χαρακτηριστικό παράδειγμα είναι οι συναρτήσεις *AlmostEqual()* οι οποίες χρησιμοποιούνται για την σύγκριση μεγεθών. Τύποι μεταβλητών όπως Vector3(), Float, Quaternion, κλπ. είναι αδύνατο να συγκριθούν μεταξύ τους με τη χρήση τελεστών όπως το «==» και απαιτούν άλλη μεθοδολογία.

Κεφάλαιο 5: Εκτέλεση προσομοιώσεων

5.1 Εισαγωγή

Στο παρόν κεφάλαιο θα γίνει παρουσίαση των αποτελεσμάτων του δυναμικού σχεδιασμού της κίνησης του βραχίονα σε διάφορα σενάρια εμποδίων. Το κεντρικό σενάριο εργασίας είναι η συνεργασία του βραχίονα με τον άνθρωπο στην στρώση ανθρακοϋφασμάτων σε καλούπι. Παρόλα αυτά και για την διευκόλυνση των εικονικών πειραμάτων, οι προσομοιώσεις εστιάζονται στην γενική αποφυγή εμποδίων σε πραγματικό χρόνο.

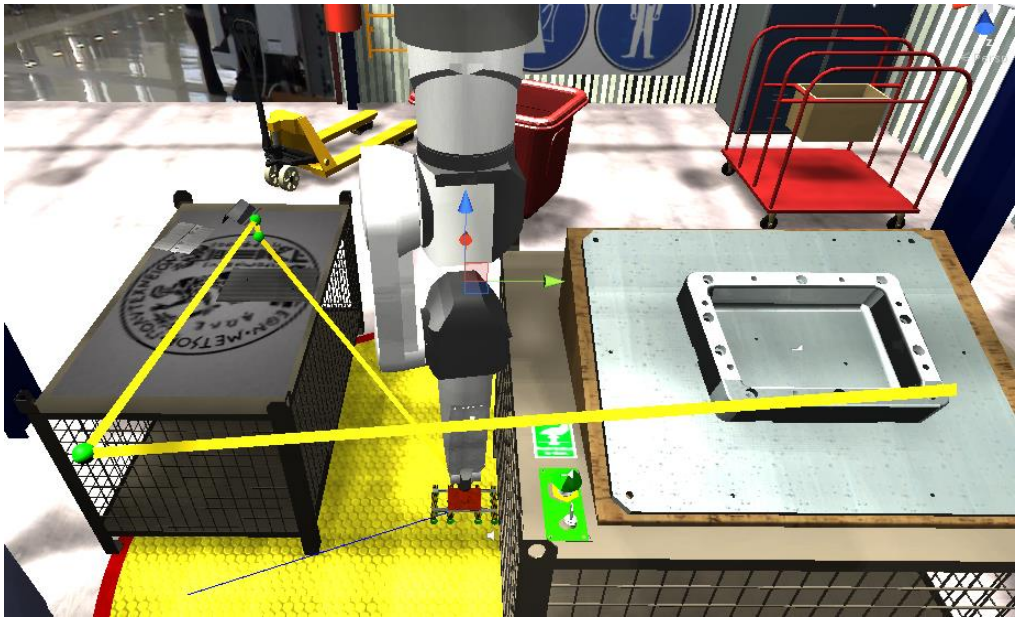
5.2 Το σενάριο

Κατά την έναρξη του προγράμματος ο βραχίονας ξεκινάει την εκτέλεση της εργασίας που έχει προγραμματιστεί. Το πρόγραμμα αποτελείται από προκαθορισμένα σημεία στο χώρο στα οποία ο βραχίονας πρέπει να μεταβεί διαδοχικά ώστε να ολοκληρώσει την εργασία. Τα σημεία αυτά θεωρούνται, πέρα από το τελικό σημείο προορισμού, σαν επιμέρους σημεία στόχοι και η χρήση τους αποτελεί κυρίως θέμα ασφάλειας αλλά και ευστάθειας του βραχίονα. Για παράδειγμα, η χρήση τους οδηγεί τον βραχίονα μέσω μίας επιθυμητής διαδρομής στο τελικό σημείο, ενώ παράλληλα δίνει χρόνο στον άνθρωπο να ολοκληρώσει την εργασία που του αναλογεί προτού αναλάβει το ρομπότ. Γενικά, σε πραγματικές ρομποτικές εφαρμογές και όταν οι αποστάσεις είναι σχετικά μεγάλες, τα ενδιάμεσα σημεία είναι πάγια τακτική που χρησιμοποιείται για την διατήρηση της ομαλής μετάβασης του ρομπότ.

Στην Εικόνα 17 φαίνεται με κίτρινη γραμμή η προκαθορισμένη διαδρομή που πρέπει να ακολουθήσει ο βραχίονας. Οι πράσινοι κόμβοι αποτελούν τα ενδιάμεσα σημεία από τα

οποία πρέπει να μεταβεί ο βραχίονας προς το τελικό σημείο στόχο. Όπως αναφέραμε σε προηγούμενο κεφάλαιο, ο βραχίονας χρησιμοποιεί το αντίστροφο κινηματικό μοντέλο για να μετατρέψει τις καρτεσιανές συντεταγμένες των σημείων σε γωνίες αρθρώσεων και να αρχίσει την κίνηση. Η προκαθορισμένη αυτή διαδρομή, σε ιδανικές συνθήκες χωρίς εμπόδια, μπορεί να εκτελεστεί ομαλά από τον βραχίονα ενώ κάθε ένα δευτερόλεπτο γίνεται έλεγχος για τυχόν εμπόδια.

Παρότι το σενάριο που εξετάζουμε τελειώνει με την ολοκλήρωση της μετάβασης στο τελικό σημείο, ο βραχίονας θα συνεχίσει και θα εκτελέσει την επόμενη εργασία που του έχει ανατεθεί (πχ. νέα κίνηση, αναμονή, κλπ.). Η προκαθορισμένη διαδρομή της εικόνας αντιστοιχεί σε μεγάλο εύρος κίνησης των αρθρώσεων του βραχίονα και επαρκεί για τη δοκιμή και επιβεβαίωση του ελεγκτή σχεδιασμού κίνησης.



Εικόνα 17: Προκαθορισμένο μονοπάτι εργασίας βραχίονα – σενάριο δοκιμής.

5.3 Μέθοδος προσομοίωσης εμποδίων

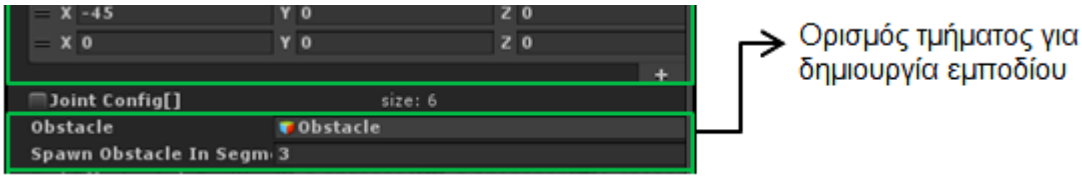
Το περιβάλλον του Unity που έχει δημιουργηθεί αποτελεί έναν εικονικό χώρο όπου η μοναδική μη προκαθορισμένη μεταβολή προέρχεται από τον χρήστη και είναι η κίνηση του ανθρώπου - avatar. Οι εντολές εισάγονται κυρίως μέσω του Kinect, όπου η κίνηση του avatar αναπαριστά με μεγάλη ακρίβεια και πιστότητα την πραγματική κίνηση του

ανθρώπου (πχ. κίνηση αγκώνα). Δυστυχώς η χρήση του Kinect κατά την επικύρωση του κώδικα είναι δύσκολη και για αυτό το λόγο, στο παρόν σενάριο, σαν εμπόδιο δεν χρησιμοποιείται το avatar αλλά άλλες τεχνητές οντότητες πιο εύκολες στη δημιουργία και χρήση τους. Θεωρήθηκε ότι αυτή η υποκατάσταση δεν επηρεάζει με οποιοδήποτε τρόπο την απόδοση και αποτέλεσμα του αλγορίθμου.

5.3.1 Δημιουργία στατικών εμποδίων

Στην Εικόνα 17 παρατηρείται ότι: Αριστερά υπάρχει το τραπέζι αποθήκευσης – αναμονής λήψης του ανθρακοϋφάσματος και δεξιά το τραπέζι εναπόθεσης και στρώσης του σε καλούπι. Και τα δύο αποτελούν στατικά εμπόδια στο χώρο εργασίας του βραχίονα τα οποία όμως αποφεύγονται εξ ορισμού. Παρόλα αυτά, σε περίπτωση αναδιάταξής τους ο αλγόριθμος σχεδιασμού κίνησης θα τα αποφύγει – αν και μία αναδιάταξη των απαραίτητων αυτών οντοτήτων θα απαιτούσε και επαναπρογραμματισμό του ρομπότ για την επιτυχή εκτέλεση της εργασίας.

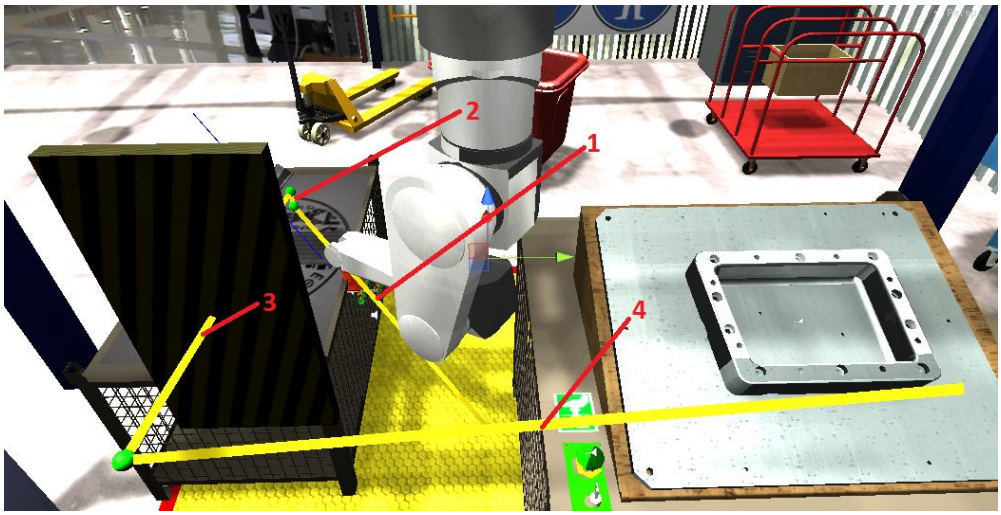
Ο πρώτος έλεγχος αποφυγής εμποδίων αναφέρεται στην εμφάνιση ενός τρίτου στατικού εμποδίου και τη μελέτη της συμπεριφοράς του ελεγκτή σχεδιασμού κίνησης. Ο χρήστης έχει τη δυνατότητα ορισμού του σημείου δημιουργίας του εμποδίου, το οποίο εμφανίζεται πέντε δευτερόλεπτα από την έναρξη του προγράμματος (Εικόνα 18).



Εικόνα 18: Ορισμός τμήματος δημιουργίας στατικού εμποδίου

Ως τμήμα ορίζεται το ευθύγραμμο τμήμα που παρεμβάλλεται μεταξύ δύο κόμβων. Στην προκειμένη περίπτωση σε σύνολο τεσσάρων τμημάτων γίνεται επιλογή δημιουργίας εμποδίου στο τρίτο εξ αυτών (Εικόνα 19).

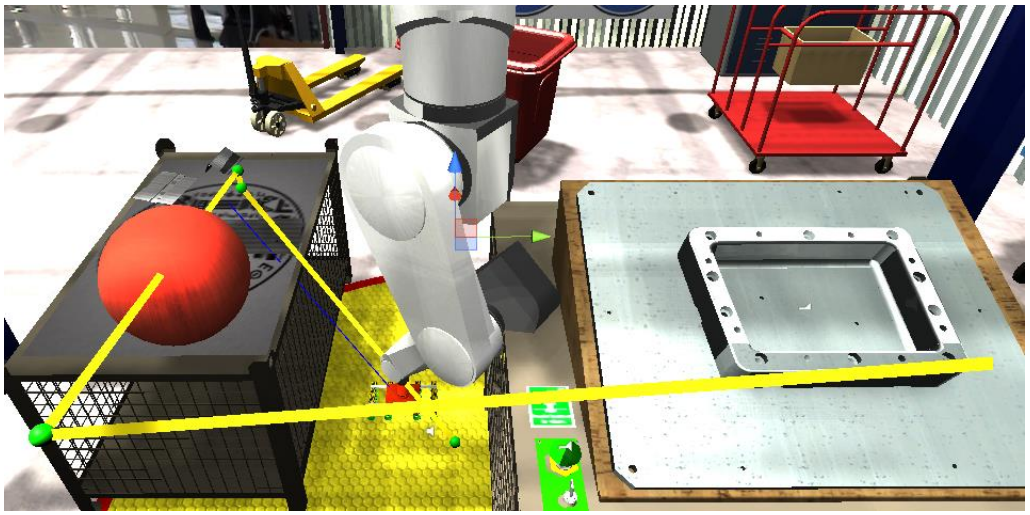
.



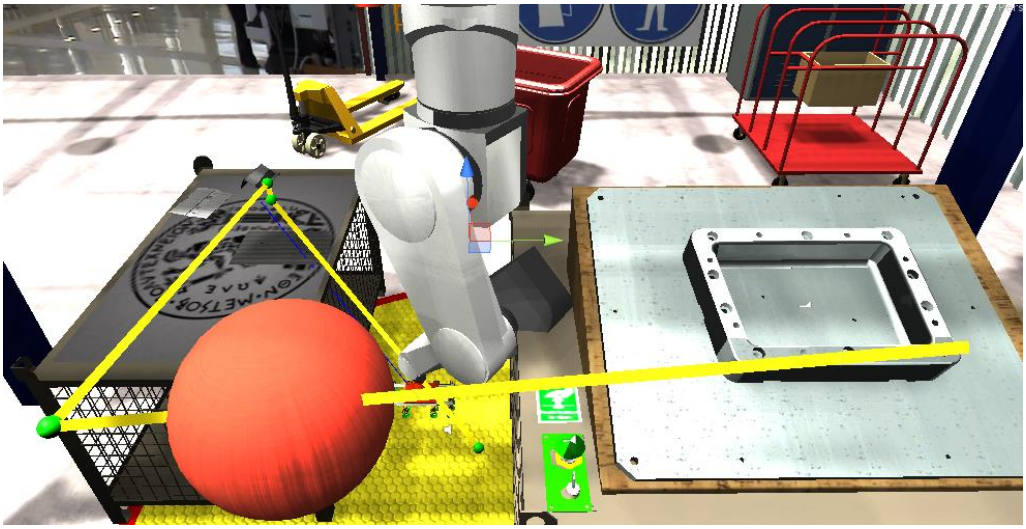
Εικόνα 19: Δημιουργία στατικού εμποδίου στο τρίτο τμήμα της διαδρομής.

5.3.2 Δημιουργία δυναμικών εμποδίων

Πέρα από την αποφυγή στατικών εμποδίων έμφαση δίνεται και σε δυναμικά εμπόδια τα οποία μπορεί να εμφανίζονται σε οποιοδήποτε τμήμα της διαδρομής, ακόμα και στο εκ νέου δημιουργημένο από την πρώτη αποφυγή του στατικού εμποδίου. Για την δυναμική προσομοίωση εμποδίων έχει δημιουργηθεί μία σφαίρα - εμπόδιο η οποία ακολουθεί το κέρσορα του ποντικιού. Ο χρήστης μπορεί να τοποθετήσει το εμπόδιο στο σημείο που επιθυμεί με την κίνηση του ποντικιού για τους δύο άξονες του επιπέδου και τη ροδέλα του ποντικιού για τον τρίτο άξονα (βάθος).



Εικόνα 20: Δημιουργία δυναμικού εμποδίου στο τρίτο τμήμα της διαδρομής.



Εικόνα 21: Δημιουργία δυναμικού εμποδίου στο τέταρτο τμήμα της διαδρομής.

5.4 Αποφυγή εμποδίων τελικού σημείου δράσης

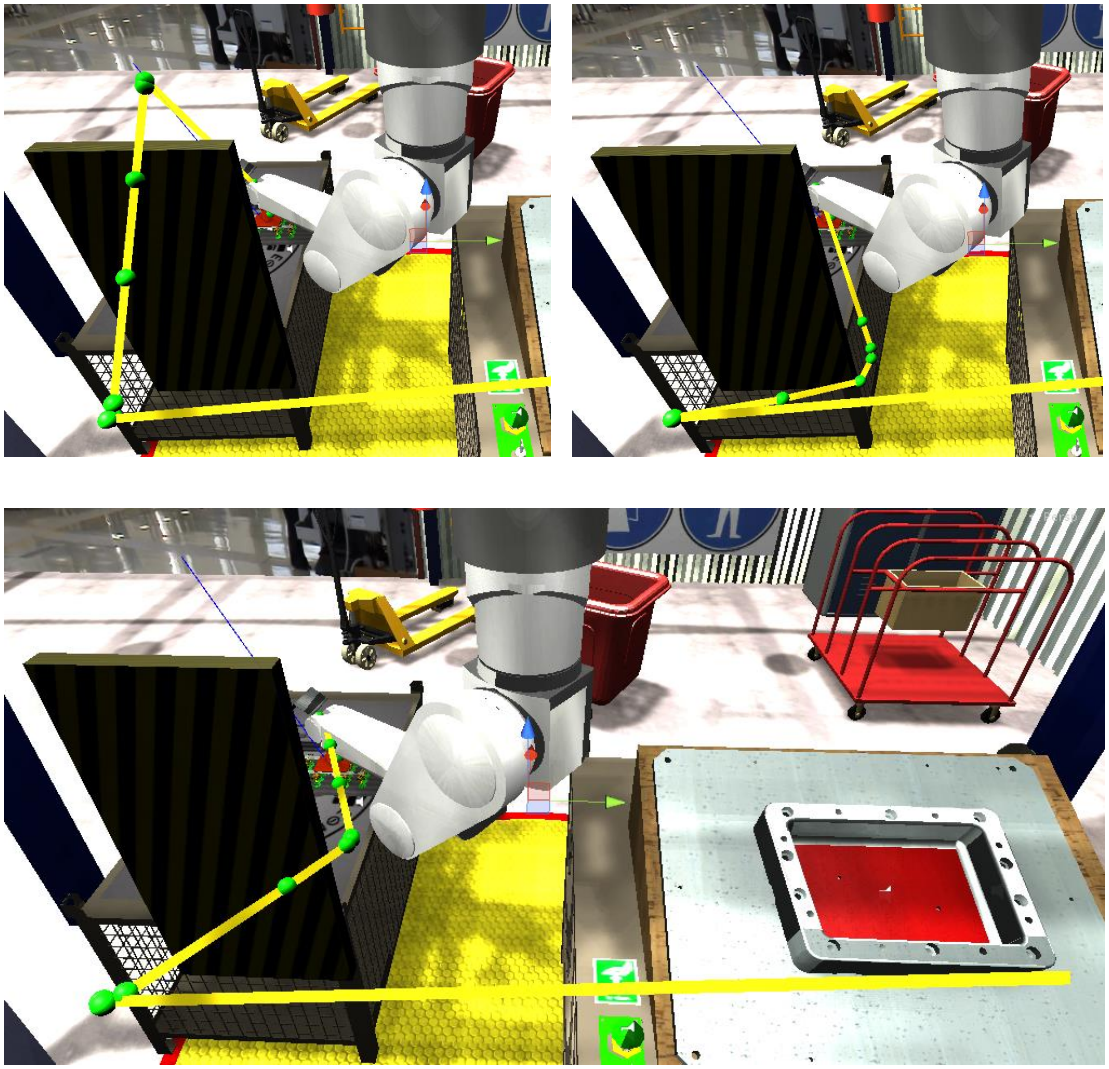
Ο χρόνος εύρεσης λύσης αλλά και το τελικό αποτέλεσμα εξαρτώνται κυρίως από τον συνδυασμό των παραμέτρων που έχουν οριστεί. Τα στατικά εμπόδια δεδομένου ότι παραμένουν «ενεργά» καθ' όλη την διάρκεια εκτέλεσης του προγράμματος υπολογίζονται μία φορά και σπάνια μπορούν να επηρεάσουν αρνητικά το αποτέλεσμα. Αντιθέτως τα δυναμικά εμπόδια, αν θεωρήσουμε ένα ακραίο σενάριο πολλαπλής εμφάνισης κατά μήκος της διαδρομής, μπορούν να δημιουργήσουν μεγάλες αποκλίσεις από την αρχική διαδρομή και διάφορα προβλήματα στο σχεδιασμό τροχιάς. Είναι πολύ σημαντικό λοιπόν, να υπάρχει εξ αρχής ένας σχεδιασμός δράσης του ελεγκτή κίνησης για όλες τις περιπτώσεις όπως πχ. απόφαση αν η νέα διαδρομή θα αλλάζει πίσω στην αρχική όταν το δυναμικό εμπόδιο εξαφανίζεται ή αν ο ελεγκτής ελέγχει όλο τη διαδρομή ή μόνο το τμήμα που κινείται το ρομπότ, κλπ.

Αξίζει να σημειωθεί εδώ, όπως προαναφέραμε στο κεφάλαιο 3, ότι οι λύσεις του αλγορίθμου RRT στηρίζονται στον δειγματοληπτικό έλεγχο του χώρου και άρα δεν έχουν εγγυημένη βέλτιστη λύση. Για αυτό το λόγο, η κάθε ξεχωριστή προσπάθεια αποφυγής του ίδιου εμποδίου, στατικού ή δυναμικού ή και συνδυασμού, μπορεί να έχει πολλές διαφορετικές λύσεις.

Στις παρακάτω υποενότητες παρουσιάζεται η συμπεριφορά του ελεγκτή σχεδιασμού διαδρομής για όλες τις περιπτώσεις εμποδίων.

5.4.1 Αποφυγή στατικών εμποδίων

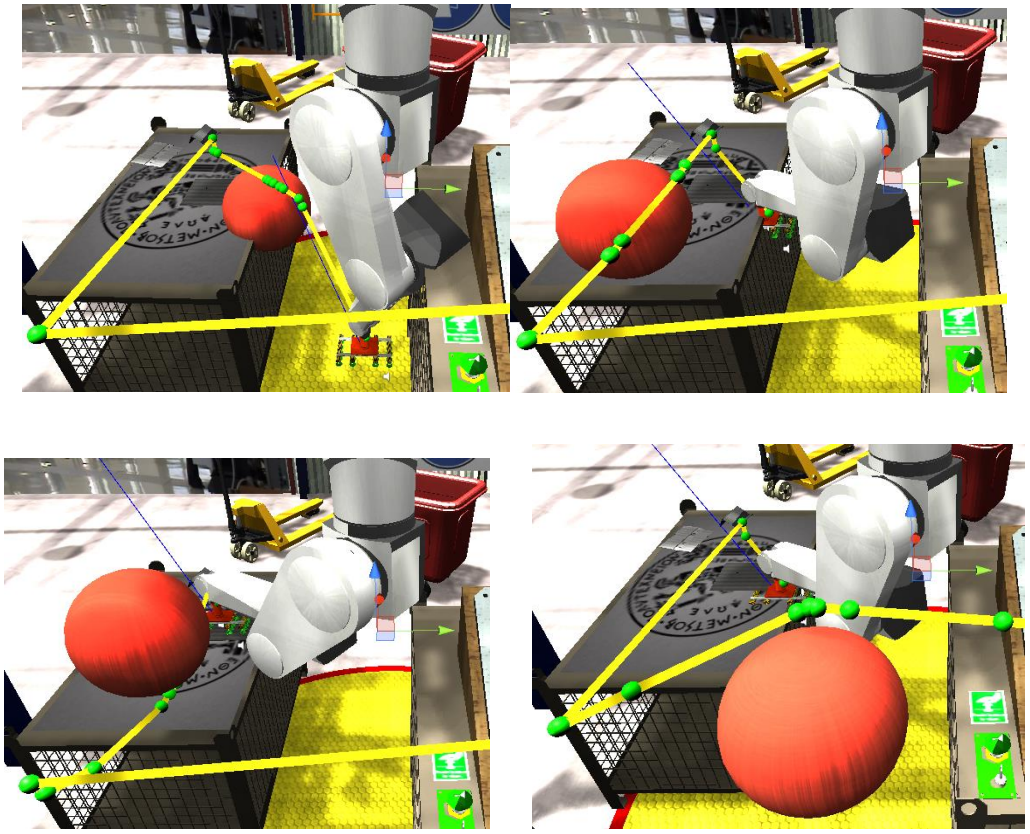
Στο σενάριο της Εικόνα 20 το στατικό εμπόδιο έχει εμφανιστεί στο τρίτο τμήμα της διαδρομής. Παρότι ο βραχίονας μπορεί να εκτελεί την κίνηση σε προηγούμενο τμήμα, ο ελεγκτής σχεδιασμού διαδρομής ελέγχει ολόκληρο τη διαδρομή και προ-υπολογίζει όλα τα επιμέρους τμήματα. Η τακτική αυτή ίσως να μην θεωρείται βέλτιστη από την σκοπιά του υπολογιστικού κόστους. Για παράδειγμα, ένα εμπόδιο στο τελευταίο τμήμα της διαδρομής ίσως να έχει εξαφανιστεί μέχρι ο βραχίονας να έχει προσεγγίσει αρκετά αυτό το τμήμα. Παρόλα αυτά, για την παρούσα εργασία, που σαν στόχο έχει την μελέτη της συμπεριφοράς του αλγορίθμου RRT, η τακτική αυτή θεωρείται ικανοποιητική. Η εκτέλεση του αλγορίθμου και μερικά από τα αποτελέσματα αποφυγής του ίδιου εμποδίου φαίνονται παρακάτω στην Εικόνα 22.



Εικόνα 22: Διαφορετικά σενάρια αποφυγής του ίδιου στατικού εμποδίου

5.4.2 Αποφυγή δυναμικών εμποδίων

Το δυναμικό εμπόδιο μπορεί να εμφανιστεί σε οποιοδήποτε τμήμα της διαδρομής. Σε περίπτωση που εμποδίζει κάποιον από τους κόμβους-στόχους, ο αλγόριθμος αδυνατεί να βρει λύση και σταματάει προτού ο βραχίονας ξεκινήσει να εκτελεί το συγκεκριμένο τμήμα που καταλήγει σε αυτόν τον κόμβο. Με την πλήρη ακινητοποίηση του βραχίονα εξασφαλίζεται η αποφυγή συγκρούσεων και ταυτόχρονα ειδοποιείται έμμεσα ο χειριστής ότι μπαίνει σε χώρο στον οποίο ο βραχίονας θέλει να καταλήξει. Μόλις το εμπόδιο απομακρύνεται, συνεχίζει κανονικά η εκτέλεση του αλγορίθμου και ο σχεδιασμός της διαδρομής.



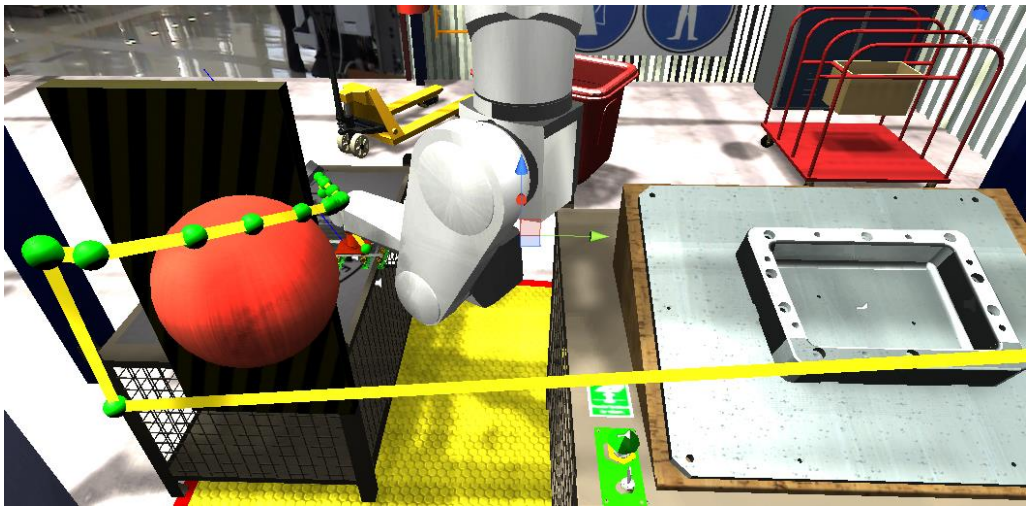
Εικόνα 23: Αποφυγή δυναμικού εμποδίου στα τμήματα 1, 3 και 4 της αρχικής διαδρομής.

5.4.3 Αποφυγή συνδυασμού εμποδίων

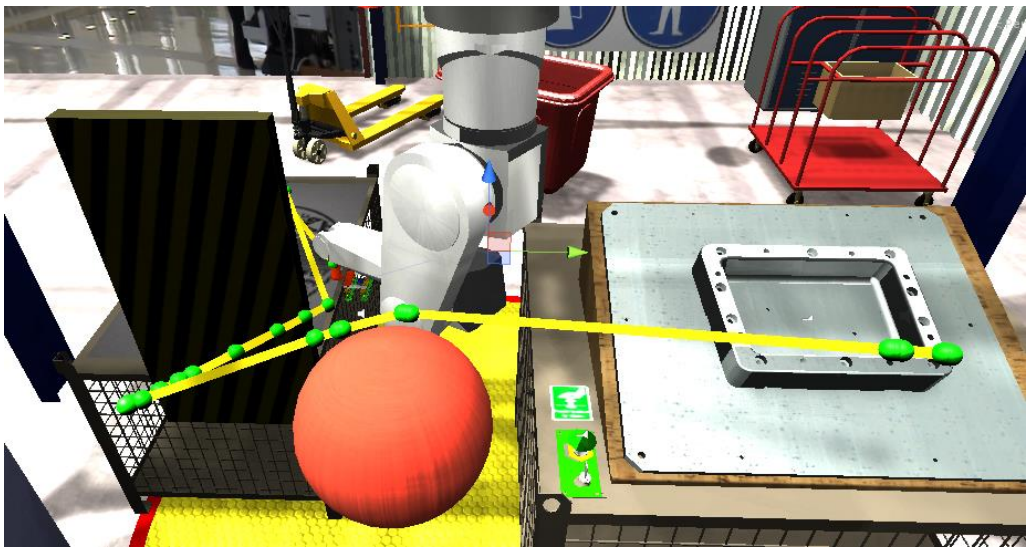
Τέλος, προσομοιώνεται η περίπτωση ύπαρξης ταυτόχρονα στατικού και δυναμικού εμποδίου το οποίο αποτελεί και το πιο πολύπλοκο σενάριο. Σε αυτή τη περίπτωση είναι απαραίτητο ο βραχίονας να γνωρίζει ποιο εμπόδιο είναι στατικό και ποιο δυναμικό ώστε να μπορέσει να διαχειριστεί κατάλληλα της δυναμικές αλλαγές στη διαδρομή. Ο τρόπος υλοποίησης στη παρούσα εργασία αλλάζει μόνιμα τη διαδρομή σε κάθε εμφάνιση του

δυναμικού εμποδίου και άρα υπάρχει περίπτωση, όταν το δυναμικό εμπόδιο εξαφανιστεί, η διαδρομή να είναι μη βέλτιστο. Οι συνδυασμοί του τρόπου εμφάνισης των εμποδίων μπορεί να είναι πάρα πολλοί και ίσως χρειάζεται διαφορετική διαχείριση για κάθε περίπτωση.

Για παράδειγμα, στην Εικόνα 24 το δυναμικό εμπόδιο έχει επηρεάσει εκ νέου τη διαδρομή που δημιουργήθηκε από την αποφυγή του στατικού εμποδίου. Στην περίπτωση που το εμπόδιο εξαφανιστεί, η διαδρομή της εικόνας θα παραμείνει η ίδια. Σε ένα ιδανικό σενάριο, η διαδρομή θα έπρεπε να αλλάζει ξανά στη προηγούμενη λύση (του στατικού εμποδίου μόνο) αλλά αυτό θα σήμαινε πως πρώτα έχει αποθηκεύτει. Υπενθυμίζεται ότι η κάθε εκ νέου εκτέλεση του αλγορίθμου RRT πιθανότατα δίνει και διαφορετική λύση.



Εικόνα 24: Αποφυγή στατικού και δυναμικού εμποδίου στο ίδιο τμήμα.



Εικόνα 25: Αποφυγή στατικού και δυναμικού εμποδίου στα τμήματα 3 και 4 αντίστοιχα.

Κεφάλαιο 6:Συμπεράσματα

6.1 Ανασκόπηση

Το όλο εγχείρημα υλοποίησης ενός ελεγκτή σχεδιασμού κίνησης ρομποτικού βραχίονα είναι αρκετά πολύπλοκο και απαιτεί τη ρύθμιση και εξέταση πολλών παραμέτρων καθώς και τη γνώση προγραμματιστικών μεθοδολογιών ώστε να θεωρηθεί επιτυχές. Δυστυχώς, στα πλαίσια μιας διπλωματικής εργασίας ο χρόνος είναι περιορισμένος και για αυτό κάποιες παραδοχές – απλοποιήσεις θεωρούνται απαραίτητες. Ταυτόχρονα όμως, το ίδιο εγχείρημα είναι αρκετά ενδιαφέρον. Η δυνατότητα συνεργασίας ανθρώπου – ρομπότ με εξασφαλισμένη την ασφάλεια του ανθρώπου αλλά και του εξοπλισμού θα αποτελέσει, αν και ακόμα είναι σε πρώιμο στάδιο, ένα μεγάλο άλμα εξέλιξης στη βιομηχανία.

6.1.1 Η πλατφόρμα Unity

Η χρήση της πλατφόρμας Unity μας βοηθάει να επικεντρωθούμε στο υπολογιστικό κομμάτι του αλγορίθμου και με ευκολία να προσομοιώσουμε τα αποτελέσματα σε εικονικό περιβάλλον και μάλιστα με εμβύθιση σε αυτό (χρήση στερεοσκοπικής μάσκας). Οι ενσωματωμένες λειτουργίες και συναρτήσεις δίνουν ευελιξία στο τρόπο προγραμματισμού και διαχείρισης του περιβάλλοντος.

6.1.2 Προγραμματισμός του ελεγκτή

Ο προγραμματισμός έγινε στη γλώσσα C#. Η ποιότητα του κώδικα επηρεάζει άμεσα την απόδοση του συνολικού εγχειρήματος καθώς ο χρόνος απόκρισης του ελεγκτή σχεδιασμού κίνησης είναι μέγιστης σημασίας για ένα σύστημα προσομοίωσης σε πραγματικό χρόνο. Στα πλαίσια της εργασίας έγινε προσπάθεια τήρησης κλασσικών μεθόδων προγραμματισμού

αλλά με κύριο στόχο την απλοποίηση του κώδικα. Μέθοδοι όπως διαμοιρασμός υπολογισμών και παράλληλη εκτέλεση (Multithreading), αν και θα προσέφεραν σημαντικά πλεονεκτήματα στο χρόνο εκτέλεσης, δεν χρησιμοποιήθηκαν. Επιπλέον, η μη προκαθορισμένη συμπεριφορά – κίνηση του βραχίονα και το χωρίς περιορισμούς τριών διαστάσεων περιβάλλον, απαιτεί την δημιουργία και διαχείριση πολλών παραμέτρων και πινάκων για τη σωστή εκτέλεση της εργασίας.

6.1.3 Ελεγκτής σχεδιασμού κίνησης

Για την δημιουργία του ελεγκτή σχεδιασμού κίνησης χρησιμοποιήθηκαν το ευθύ και αντίστροφο κινηματικό μοντέλο του βραχίονα, ο αλγόριθμος RRT, συναρτήσεις εντοπισμού στατικών και δυναμικών εμποδίων στο χώρο αλλά και στις αρθρώσεις του ρομπότ, βοηθητικές συναρτήσεις υπολογισμών, διαδραστικό μενού για είσοδο και χειρισμό από το χρήστη, κλπ. Έμφαση δόθηκε στον ελεγκτή σχεδιασμού διαδρομής ο οποίος και ολοκληρώθηκε. Ο ελεγκτής σχεδιασμού τροχιάς αποτελεί από μόνος του ένα μεγάλο κομμάτι έρευνας και περιορίστηκε στην εύρεση εμποδίων σε όλες τις αρθρώσεις του βραχίονα και απόρριψη των μη αποδεκτών λύσεων του αντίστροφου κινηματικού μοντέλου.

6.2 Αποτελέσματα

Ο ελεγκτής σχεδιασμού κίνησης που υλοποιήθηκε μπορεί δυναμικά να σχεδιάζει εκ νέου διαδρομές για την ταυτόχρονη εκτέλεση της εργασίας και την αποφυγή συγκρούσεων με οποιοδήποτε εμπόδιο. Ο αλγόριθμος RRT επιτρέπει μικρούς χρόνους απόκρισης (< 1 sec), που όμως εξαρτώνται από το ρυθμό ανανέωσης (frame rate), με χαμηλό σχετικά υπολογιστικό κόστος ενώ παράλληλα εξασφαλίζει την ακεραιότητα του ανθρώπου αλλά και του εξοπλισμού. Ο χρήστης είναι σε θέση να ορίσει βασικές παραμέτρους του συστήματος όπως ανάλυση και μέγιστο χρόνο εκτέλεσης παραμετροποιώντας έτσι τον ελεγκτή ανάλογα με τις συνθήκες που επικρατούν.

Ο σχεδιασμός της διαδρομής αφορά το τελικό σημείο δράσης του βραχίονα. Για την επιτυχή ολοκλήρωση της εργασίας απαιτείται συνεργασία με τον ελεγκτή σχεδιασμού τροχιάς ο οποίος ελέγχει την ομαλή μετάβαση όλων των αρθρώσεων του βραχίονα προς το τελικό

σημείο στόχο. Ο συγκεκριμένος ελεγκτής αν και δεν παρέχει βέλτιστες λύσεις, μπορεί να ανιχνεύσει εμπόδια σε όλες τις αρθρώσεις του βραχίονα και να αποθηκεύσει για κάθε σημείο της διαδρομής μόνο τις λύσεις που ικανοποιούν την αποφυγή εμποδίων.

6.3 Περαιτέρω έρευνα - Βελτιώσεις

Η πλατφόρμα Unity μπορεί, με προσθήκες νέων λειτουργιών, να επεκτείνεται συνεχώς και να καλύπτει όλο και περισσότερα σενάρια συνεργασίας ανθρώπου-ρομπότ.

Ενδιαφέρον θα είχε η υλοποίηση διαφορετικών αλγορίθμων σχεδιασμού διαδρομής (πχ. RRT*, A*) και σύγκριση τους με τον ήδη υλοποιημένο RRT σε επίπεδο απόδοσης, υπολογιστικού κόστους και εύρεσης βέλτιστης λύσης. Η πλατφόρμα του Unity, με σωστό προγραμματισμό, μπορεί να επιτρέψει την άμεση εναλλαγή της μεθόδου σχεδιασμού διαδρομής και άρα την άμεση σύγκριση μεταξύ τους για το ίδιο ακριβώς περιβάλλον και πρόβλημα. Η πλειοψηφία της υπάρχουσας βιβλιογραφίας εξετάζει το πρόβλημα σε απλοποιημένη μορφή των δύο διαστάσεων ενώ δεν παρέχει τρόπο στον ερευνητή να κάνει εμβύθιση στο εικονικό περιβάλλον και να δει «στη πράξη» την συμπεριφορά του ελεγκτή. Ένα τέτοιο ολοκληρωμένο περιβάλλον θα μπορούσε να χρησιμοποιείται για την εκπαίδευση των χειριστών και την εξοικείωσή τους με τα ρομπότ.

Ενέργειες μπορούν επίσης να γίνουν προς την κατεύθυνση βελτίωσης της υπάρχουσας δομής του κώδικα και των σεναρίων. Σημαντική βελτίωση στην απόδοση του αλγορίθμου και την ομαλή εκτέλεση του προγράμματος θα είχε η βελτιστοποίηση του κώδικα που αναπτύχθηκε με τεχνικές παράλληλης επεξεργασίας (multithreading) καθώς και των μεθόδων εκτέλεσης των υπολογισμών. Για παράδειγμα, ο υπολογισμός ενός τμήματος της διαδρομής κάθε φορά θα μείωνε δραστικά το υπολογιστικό κόστος. Επίσης ένας πιο αναλυτικός σχεδιασμός για τη συμπεριφορά του ελεγκτή σε κάθε είδους σενάρια (ακόμα και τα πολύ σπάνια) θα οδηγούσε σε περαιτέρω βελτιστοποίηση του περιβάλλοντος. Για παράδειγμα, σε περίπτωση μη εύρεσης λύσης θα μπορούσε να γίνεται αλλαγή στο ήδη υπάρχον δέντρο και όχι εκ νέου δημιουργία του. Προφανώς η ολοκλήρωση του ελεγκτή τροχιάς θα ήταν απαραίτητη σε αυτό το σημείο.

Βιβλιογραφία

[1] L. Wan, B. Schmidt and A. Y. Neec, "Vision-guided active collision avoidance for human-robot collaborations," *Manufacturing Letters,* vol. 1, no. 1, pp. 5-8, October 2013.

[2] Η. Μάτσας, "Ανάπτυξη και αξιολόγηση διαδραστικών μοντέλων εικονικής πραγματικότητας για το σχεδιασμό ρομποτικών συστημάτων παραγωγής," Διδακτορική Διατριβή, ΕΜΠ, Αθήνα, 2015.

[3] Y. K. Hwang and N. Ahuja, "Gross Motion Planning - A Survey," *ACM Computing Surveys,* vol. 24, no. 3, pp. 219-291, 1992.

[4] L. Yang, J. Qi, D. Song and J. Xiao, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering,* vol. 5, no. ID 7426913, p. 22, 2016.

[5] D. Bertram, J. Kuffner, R. Dillmann and T. Asfour, "An Integrated Approach to Inverse Kinematics and Path Planning for Redundant Manipulators," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May 2006.

[6] C. Nagata, E. Sakamoto, M. Suzuki and S. Aoyagi, "Path Generation and Collision Avoidance of Robot Manipulator for Unknown Moving Obstacle using Real-time Rapidly-exploring Random Trees (RRT) Method," in *Service Robotics and Mechatronics*, London, Springer, 2010, pp. 335-340.

[7] J. Kuffer and S. Lavalle, "Rapidly-exploring random trees:a new tool for path planning,"

Computer Science Dept, Iowa State University, 1998.

[8] J. Kuffer and S. Lavalle, "Rapidly-exploring random trees:progress and prospect," in *Internationsl workshop on Algorithmic and computational Robotics (WAFR)*, 1999, pp. 293-309.

[9] G. Snchez and J. c. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *In Int. Symp. Robotics Research*, 2001, pp. 403-417.

[10] H. Shen, H. Wu, B. Chen, Y. Jiang and a. C. Yan, "Obstacle Avoidance Algorithm for 7-DOF Redundant Anthropomorphic Arm," *Journal of Control Science and Engineering,* vol. 2015, 2014.

[11] B. I. Kazem, A. I. Mahdi and A. T. Oudah, "Motion Planning for a Robot Arm by Using Genetic Algorithm," *Jordan Journal of Mechanical and Industrial Engineering,* vol. 2, no. 3, pp. 131-136, 2008.

[12] A. Machmudah and S. Parman, "Polynomial Joint Angle Based Motion Planning for Robotics Arm," *Asian Journal of Scientific Research,* vol. 6, no. 3, pp. 488-497, 2013.

[13] S. Klanke, D. Lebedev, R. Haschke, J. Steil and H. Ritter, "Dynamic Path Planning for a 7-DOF Robot Arm," in *Proceedings of the 2006 IEEE/RSJ,International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[14] K.Oh, E.Kim, J.P.Hwang and H.Lee, "Path plannig of a robot manipulator using retrieval RRT strategy," *World Academy of Science,Egineering and Technology,International Journal of Electrical and Computer Engineering,* vol. 1, no. 1, 2007.

[15] B. Kiss and E. Szadeczky-Kardoss, "Extension of the rapidly-exploring random trees algorithm with key configurations for nonholonomic motion planning," 2007, pp. 293-309.

[16] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in

Proceedings of the 1993 IEEE International Conference on Robotics and Automation, May 1993.

[17] S. Quinlan, "Real-time modification of collision-free paths," PhD thesis, Stanford University, Stanford, CA, USA, 1995.

[18] O. Brock and O. Khatib, "Elastic Strips: A Framework for Motion Generation in Human Environments," *The International Journal of Robotics Research,* vol. 21, no. 12, pp. 1031-1052, 2002.

[19] O. Brock, "Generating robot motion: the integration of planning and execution," PhD thesis,Stanford University, Stanford, CA, 2000.

[20] O. Brock and L. Kavraki, "Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces.," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, 2001.

[21] O. Brock and L. Kavraki, "Decomposition-based motion planning: Towards real-time planning for robots with many degrees of freedom," 2000.

[22] C. Ramer, S. Reitelshöfer and J. Franke, "A robot motion planner for 6-DOF industrial robots based on the cell decomposition of the," in *Conference Paper*, October 2013.

[23] P. Leven and S. Hutchinson, "A Framework for Real-time Path Planning in Changing Environments," *The International Journal of Robotics Research,* vol. 21, no. 12, pp. 999-1030, 2002.

[24] K. Belghith, F. Kabanza, L. Hartman and R. Nkambou, "Anytime dynamic path planning with flexible probabilistic roadmaps," in *Proceedings of the 2006 IEEE ternational Conference on Robotics and Automation*, May 2006.

[25] C. Qin and D. Henrich, "Path Planning for Industrial Robot arms: A Parallel Randomized Approach," in *International Symposium on Intelligent Robotic Systems (SIRS 96)*, Lisbon, 1996.

[26] http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html. [Online].

[27] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *2000 IEEE Int'l Conf. on Robotics and Automation (ICRA 2000)*, 2000.

[28] "Improving the Optimality of RRT: RRT*," [Online]. Available: http://joonlecture.blogspot.gr/2011/02/improving-optimality-of-rrt-rrt.html.

[29] "Kinodynamic Trajectories," [Online]. Available: http://rikkimelissa.github.io/portfolio/kinodynamic-trajectories.html.

[30] J. Wang, Y. Li and a. X. Zhao, "Inverse Kinematics and Control of a 7-DOF Redundant Manipulator Based on the Closed-Loop Algorithm," *International Journal of Advanced Robotic Systems,* vol. 7, no. 4, pp. 1-9, 2010.

[31] R. H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories," *IBM Journal of Research and Development,* vol. 23, no. 4, pp. 424-436, 1979.

[32] [Online]. Available: http://nojhan.free.fr/metah/.

[33] http://www.staubli.com. [Online].

[34] T.C.Manjunath, "Optimal Straight Line Trajectory Generation in 3D Space using Deviation Algorithm," *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering ,* vol. 2, no. 1, 2008.

[35] S. M. LaValle, "Motion Planning: The Essentials".

[36] L. Zongxing, X. Chunguang, P. Qinxue, Z. Xinyu and L. Xinliang, "Inverse Kinematic Analysis and Evaluation of a Robot for Nondestructive Testing Application," *Journal of Robotics,* vol. 2015, p. 7, 2015.

[37] https://unity3d.com/. [Online].

[38] J. J. Craig, "Trajectory Generation," in *Introduction to robotics mechanics and control 3rd_edition*, Addison-Wesley, 1989, pp. 201-220.

Παράρτημα – κώδικες

Robot Mover.cs

```
using UnityEngine;
using System;
using System.IO;
using System.Collections;
using System.Threading;
using System.Collections.Generic;
using System.Linq;


public class RobotMover :
MonoBehaviour
{

[Range (-160f,160f)] public float q1 =
0;
[Range (-227.5f,47.5f)] public float
q2 = -90;
[Range (-52.5f,232.5f)] public float
q3 = 90;
[Range (-270f,270f)] public float q4 =
0;
[Range (-105f,120f)] public float q5 =
0;
[Range (-270f,270f)] public float q6 =
0;

//Instantiate the RobotController
public RobotController
RobotControllerInstance;
//Instantiate the RRT Path Planner
private RRTController
RRTPathPlannerInstance;
private LinearTrajectoryPlanner
TrajectoryPlannerInstance;
private Camera Camera;

private bool manualMode;
private bool RRTPathPlanner;
private bool cartesianMode;
private bool jointMode;
public enum movementMode{Manual,
Automatic} ;
public enum coordSystemMode{World,
Joint} ;
public movementMode workMode;
public coordSystemMode FrameMode;

public float rotationSpeed = 1f;
//Lerp speed
private float ikSpeed = 0.005f; //TCP
moving speed
private float ikRotSpeed = 100f; //TCP
rotating speed
private  int InPositionCounter;
private  int internalPositionCounter;

public List<Vector3> programmedPathT =
new List<Vector3> ();
public List<Vector3> programmedPathR =
new List<Vector3> ();
public List<Vector3> initialPathT =
new List<Vector3> ();
public List<Vector3> initialPathR =
new List<Vector3> ();
public List<Vector3> RRTCalculatedPath
= new List<Vector3> ();
public RobotPath pathToExecute;
List<Cartesian6DConfig> userPath = new
List<Cartesian6DConfig>();
private bool obstacleExists=false;
private bool solutionExists=false;
private bool obstacleCollisionImminent
= false;
private int obstacleSegmentInPath = -
1;
private int
obstacleInternalSegmentInPath = -1;
public List<Vector3>
pathForObstacleCheck;
private float PauseTime= 1f;
private float NextTime= 0f;
public List<Vector3> savedPointsT =
new List<Vector3> ();
public List<Vector3> savedPointsR =
new List<Vector3> ();
public float[] jointConfig = new
float[6];
```

```
private Quaternion[]
jointConfigInQuaternion = new
Quaternion[6];

public GameObject Obstacle;
public int spawnObstacleInSegment = 3;
private List<GameObject>
instantiatedObjs=new List<GameObject>
();

public Vector3 endEffectorLabPosition;
public Vector3 wristLabPosition;
public Vector3 toolFlangeLabPosition;


[HideInInspector]
public Transform
endEffectorUnityPosition;
private Transform
toolFlangeUnityPosition;
public Transform wristUnityPosition;
public Transform
referenceFrameUnityPosition;

private Vector3 PrevTargetT;
private Vector3 PrevTargetR;
private Vector3 TargetT;
private Vector3 TargetR;
private bool reachedTargetJoint=false;

// Προηγουμενο πληκτρο που πατηθηκε
private int prevKeyPress = 100;

// System Sounds
private AudioSource
outOfDexterousSpace;

// Used to Rotate the Joint Objects
private Transform joint1;
private Transform joint2;
private Transform joint3;
private Transform joint4;
private Transform joint5;
private Transform joint6;

private KeyCode[] keyCodes = {
KeyCode.Alpha1,
KeyCode.Alpha2,
KeyCode.Alpha3,
KeyCode.Alpha4,
KeyCode.Alpha5,
KeyCode.Alpha6,
KeyCode.Alpha7,
KeyCode.Alpha8,
KeyCode.Alpha9,
};

void Awake ()
{
RobotControllerInstance = new
RobotController();
joint1 = GameObject.Find ("Joint
1").GetComponent<Transform> ();
joint2 = GameObject.Find ("Joint
2").GetComponent<Transform> ();
joint3 = GameObject.Find ("Joint
3").GetComponent<Transform> ();
joint4 = GameObject.Find ("Joint
4").GetComponent<Transform> ();
joint5 = GameObject.Find ("Joint
5").GetComponent<Transform> ();
joint6 = GameObject.Find ("Joint
6").GetComponent<Transform> ();

wristUnityPosition
=GameObject.Find("(#10353) RX90BL
WRIST").GetComponent<Transform>();
toolFlangeUnityPosition =
GameObject.Find ("(#11508) RX90BL TOOL
FLANGE").GetComponent<Transform> ();
endEffectorUnityPosition =
GameObject.Find
("Object_103").GetComponent<Transform>
();
referenceFrameUnityPosition =
GameObject.Find ("Joint
2").GetComponent<Transform> ();
Camera = GameObject.Find
("Camera").GetComponent<Camera> ();

outOfDexterousSpace =
GameObject.Find("Censor Beep Sound
Effect").GetComponent<AudioSource> ();

switch (workMode) {

case movementMode.Automatic:
TrajectoryPlannerInstance = new
LinearTrajectoryPlanner ();
var RRTScript =
GetComponent<RRTController> ();
RRTScript.enabled = true;
RRTPathPlannerInstance =
GameObject.Find ("Staubli
RX90L").GetComponent<RRTController>
();
RRTPathPlanner = true;
break;
case movementMode.Manual:
RRTScript =
GetComponent<RRTController> ();
RRTScript.enabled = false;
manualMode = true;

switch (FrameMode) {

case coordSystemMode.Joint:
jointMode = true;
break;
case coordSystemMode.World:
cartesianMode = true;
break;
}
break;
}


}


void Start()
{

InPositionCounter = 1;
internalPositionCounter = 1;
```

```
endEffectorLabPosition =
unitytoRobotCoordinateSystemChange
(endEffectorUnityPosition.position);
wristLabPosition =
unitytoRobotCoordinateSystemChange
(wristUnityPosition.position);
toolFlangeLabPosition =
unitytoRobotCoordinateSystemChange
(toolFlangeUnityPosition.position);

TargetT.x = 0f;
TargetT.y = 0f;
TargetT.z = 1.185f;
TargetR.x = 0f;
TargetR.y = 0f;
TargetR.z = 0f;

PrevTargetT = TargetT;
PrevTargetR = TargetR;
jointConfig =
RobotControllerInstance.IKcontroller
(TargetT, TargetR)[0];
FKsol =
RobotControllerInstance.FKcontroller
(jointConfig);

switch (workMode) {
case movementMode.Automatic:
//Insert user desired path
userPath.Add (new Cartesian6DConfig
(0f, 0f, 1.185f, 0f, 0f, 0f));
userPath.Add (new Cartesian6DConfig (-
0.560f, 0.775f, 0.450f, 0f, 0f, 0f));
userPath.Add (new Cartesian6DConfig (-
0.560f, 0.775f, 0.380f, 0f, 0f, 0f));
userPath.Add (new Cartesian6DConfig
(0.735f, 0.726f, 0.167f, 0f, 0f, 0f));
userPath.Add (new Cartesian6DConfig
(0.771f, -0.8145f, -0.0476f, 0f, 0f,
0f));


//create the List of the Path
pathToExecute = new RobotPath
(userPath);

//just to see in editor - else need to
crete custom editor
initialPathT = new
List<Vector3>(DrawCartesianTranslation
List (pathToExecute));
initialPathR = new
List<Vector3>(DrawCartesianRotationLis
t (pathToExecute));
programmedPathT = new
List<Vector3>(DrawCartesianTranslation
List (pathToExecute));
programmedPathR = new
List<Vector3>(DrawCartesianRotationLis
t (pathToExecute));


//Spawn Obstacle if defined >0 in
editor
if (spawnObstacleInSegment>0){
Invoke("ObstacleSpawner", 5);//this
will happen after 5 seconds
}
break;
}
}


//On validate allows to change joint
angles through the editor
void OnValidate(){

if ((manualMode)&&(jointMode)){
//Avoid null reference
if (RobotControllerInstance == null)
{
return;
}

else if (RobotControllerInstance !=
null) {

jointConfigInQuaternion [0] =
Quaternion.Euler
(RobotControllerInstance.rotX1,
RobotControllerInstance.rotYAll, q1);
jointConfigInQuaternion [1] =
Quaternion.Euler
(RobotControllerInstance.rotX2,
RobotControllerInstance.rotYAll, q2);
jointConfigInQuaternion [2] =
Quaternion.Euler
(RobotControllerInstance.rotX3,
RobotControllerInstance.rotYAll, q3);
jointConfigInQuaternion [3] =
Quaternion.Euler
(RobotControllerInstance.rotX4,
RobotControllerInstance.rotYAll, q4);
jointConfigInQuaternion [4] =
Quaternion.Euler
(RobotControllerInstance.rotX5,
RobotControllerInstance.rotYAll, q5);
jointConfigInQuaternion [5] =
Quaternion.Euler
(RobotControllerInstance.rotX6,
RobotControllerInstance.rotYAll, q6);
jointConfig = new float[]
{q1,q2,q3,q4,q5,q6};
}
}
}
public List<Vector4> FKsol = new
List<Vector4> ();
void Update()
{
FKsol =
RobotControllerInstance.FKcontroller
(jointConfig);

//Home Position
if (Input.GetKeyUp (KeyCode.Space))
{

//Go Home Position and reset Target
values
RobotControllerInstance.CalculateDoRea
dy();

TargetT.x = 0f;
TargetT.y = 0f;
TargetT.z = 1.185f;
```

```
TargetR.x = 0f;
TargetR.y = 0f;
TargetR.z = 0f;

PrevTargetT = TargetT;
PrevTargetR = TargetR;

jointConfig =
RobotControllerInstance.IKcontroller
(TargetT, TargetR)[0];
}

if (Input.anyKey){

SavePositions ();

// MANUAL MODE
if (manualMode) {

ManualMovement ();


for (int i = 0; i < keyCodes.Length;
i++) {
if (Input.GetKeyDown (keyCodes [i])) {
int numberPressed = i + 1;
Debug.Log ("Number Pressed: " +
numberPressed);
TargetT = savedPointsT [i] / 1000f;
TargetR = savedPointsR [i];
}
}


//First check if the Robot needs to
move
if ((TargetT != PrevTargetT) ||
(TargetR != PrevTargetR)) {
jointConfig =
RobotControllerInstance.IKcontroller
(TargetT, TargetR)[0];
q1 = jointConfig[0] ;
q2 = jointConfig[1];
q3 = jointConfig[2];
q4 = jointConfig[3];
q5 = jointConfig[4];
q6 = jointConfig[5];
if
(RobotControllerInstance.outofDexterou
sSpaceFlag) {
Debug.Log ("Out of Limits");
outOfDexterousSpace.Play ();
}
PrevTargetT = TargetT;
PrevTargetR = TargetR;

}
}
}

if (RRTPathPlanner) {

//Run the obstacle detection after
every PauseTime secs
if (Time.time > NextTime) {
NextTime = Time.time + PauseTime;
ObstacleDetector (pathToExecute);
}

//activate the RRT pathing
if ((obstacleExists) &&
(solutionExists)) {

//if obstacle is in the same segment
as the robot is then use as starting
point the current Robot position
if (obstacleCollisionImminent) {
Vector3 targetPointForPlan =
pathToExecute.SegmentID
[obstacleSegmentInPath].endPoint.carte
sian.translationVector;
RRTCalculatedPath =
RRTPathPlannerInstance.Pathplan
(toolFlangeUnityPosition.position,
robottoUnityCoordinateSystemChange
(targetPointForPlan));
}
//if obstacle is later on the path use
the closest segment startPoint
else {
Vector3 startPointForPlan =
pathToExecute.SegmentID
[obstacleSegmentInPath].startPoint.car
tesian.translationVector;
Vector3 targetPointForPlan =
pathToExecute.SegmentID
[obstacleSegmentInPath].endPoint.carte
sian.translationVector;
RRTCalculatedPath =
RRTPathPlannerInstance.Pathplan
(robottoUnityCoordinateSystemChange
(startPointForPlan),
robottoUnityCoordinateSystemChange
(targetPointForPlan));
}

//Modify Path
//Always Clean the Segment and make it
as it was in the initialization phase
int itemstoRemove =
pathToExecute.SegmentID
[obstacleSegmentInPath].segmentPoint.C
ount-1;
pathToExecute.SegmentID
[obstacleSegmentInPath].segmentPoint.R
emoveRange(1,itemstoRemove);
//Resize
pathToExecute.SegmentID
[obstacleSegmentInPath].segmentPoint.T
rimExcess ();

//Insert the RRT points
for (int i = RRTCalculatedPath.Count -
1; i >= 0; --i) {
Vector3 pointToInsert =
unitytoRobotCoordinateSystemChange
(RRTCalculatedPath [i]);
PathPoint toInsert = new PathPoint ();
toInsert.cartesian.translationVector =
pointToInsert;
pathToExecute.SegmentID
[obstacleSegmentInPath].segmentPoint.I
nsert (1, toInsert);
}
//Reset the flag as the obstacle has
been avoided
obstacleExists = false;
```

```
}

//HighLight Path
PathDraw (programmedPathT);

//if there is imminent collision(robot
and obstacle in same segment) without
solution then robot stops where it
currently is
if ((obstacleCollisionImminent) &&
!(solutionExists)) {
Debug.Log ("No solution ...STOPPING");
TargetT = toolFlangeLabPosition;
jointConfig =
RobotControllerInstance.IKcontroller
(TargetT, TargetR,"LIN")[0];
}
else if (InPositionCounter <
pathToExecute.SegmentID.Count) {
Cartesian6DConfig
PathPointLabPosition;
if (internalPositionCounter <=
pathToExecute.SegmentID
[InPositionCounter].segmentPoint.Count
) {
if (internalPositionCounter ==
pathToExecute.SegmentID
[InPositionCounter].segmentPoint.Count
) {
PathPointLabPosition =
pathToExecute.SegmentID
[InPositionCounter].endPoint.cartesian
;
}
else {
PathPointLabPosition =
pathToExecute.SegmentID
[InPositionCounter].segmentPoint
[internalPositionCounter].cartesian;
}
TargetT =
PathPointLabPosition.translationVector
;
TargetR =
PathPointLabPosition.rotationVector;
//If Target command has changed OR
Robot is not in  Target position run
the IK
if (Vector3AlmostEqual (TargetT,
PrevTargetT, 0.015f) == false ||
Vector3AlmostEqual (TargetR,
PrevTargetR, 0.15f) == false ||
Vector3AlmostEqual (TargetT,
toolFlangeLabPosition, 0.015f) ==
false) {
PrevTargetT = TargetT;
PrevTargetR = TargetR;
jointConfig =
RobotControllerInstance.IKcontroller
(TargetT, TargetR, "LIN")[0];
q1 = jointConfig[0] ;
q2 = jointConfig[1];
q3 = jointConfig[2];
q4 = jointConfig[3];
q5 = jointConfig[4];
q6 = jointConfig[5];
if
(RobotControllerInstance.outofDexterou
sSpaceFlag) {
Debug.Log ("Out of Limits");
outOfDexterousSpace.Play ();
RRTPathPlanner = false;
}
}
else {
//check if target command has been
executed
if (reachedTargetJoint == true &&
Vector3AlmostEqual (TargetT,
toolFlangeLabPosition, 0.015f)) {
//if it is a checkpoint
if (Vector3AlmostEqual (TargetT,
pathToExecute.SegmentID
[InPositionCounter].endPoint.cartesian
.translationVector, 0.005f)) {
Debug.Log ("Robot is in Node# " +
InPositionCounter + " from the total "
+ pathToExecute.SegmentID.Count + "
Nodes");
InPositionCounter++;
internalPositionCounter = 1;
}
else {
internalPositionCounter++;
}
}
}
}
}
programmedPathT = new
List<Vector3>(DrawCartesianTranslation
List (pathToExecute));
programmedPathR = new
List<Vector3>(DrawCartesianRotationLis
t (pathToExecute));
}


//All movement done here!
MoveToTarget ();

//Update coordinates of robot
endEffectorLabPosition =
unitytoRobotCoordinateSystemChange
(endEffectorUnityPosition.position);
wristLabPosition =
unitytoRobotCoordinateSystemChange
(wristUnityPosition.position);
toolFlangeLabPosition =
unitytoRobotCoordinateSystemChange
(toolFlangeUnityPosition.position);
}

//User Input
float xAxis; //Translation
float yAxis;
float zAxis;
float axAxis; //Euler angles
float ayAxis;
float azAxis;

void ManualMovement()
{
//Get Axis values from user input
```

```
zAxis = Input.GetAxis ("7th_Axis") *
ikSpeed;
xAxis = Input.GetAxis ("XRobot") *
ikSpeed;
yAxis = Input.GetAxis ("YRobot") *
ikSpeed;
axAxis = Input.GetAxis ("axRobot") *
ikSpeed;
ayAxis = Input.GetAxis ("ayRobot") *
ikSpeed;
azAxis = Input.GetAxis ("azRobot") *
ikSpeed;

if (cartesianMode) {
//Translation Axis
if (zAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetT.z += zAxis;
} else {
TargetT.z += Math.Abs(2*zAxis);
outOfDexterousSpace.Play ();
}
}

if (yAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetT.y += yAxis;
} else {
TargetT.y += Math.Abs(2*yAxis);
outOfDexterousSpace.Play ();
}
}

if (xAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetT.x += xAxis;
} else {
TargetT.x += Math.Abs(2*xAxis);
outOfDexterousSpace.Play ();
}
}

//Rotation Axis
if (axAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetR.x += axAxis * ikRotSpeed;
} else {
TargetR.x += Math.Abs(axAxis) *
ikRotSpeed;
outOfDexterousSpace.Play ();
}
}

if (ayAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetR.y += ayAxis * ikRotSpeed;
} else {
TargetR.y += Math.Abs(ayAxis) *
ikRotSpeed;
outOfDexterousSpace.Play ();
}
}

if (azAxis != 0f) {
if
(!(RobotControllerInstance.outofDexter
ousSpaceFlag)) {
TargetR.z += azAxis * ikRotSpeed;
} else {
TargetR.z += Math.Abs(azAxis) *
ikRotSpeed;
outOfDexterousSpace.Play ();
}
}
}
}


// Moves robot to next position from
the previous position
void MoveToTarget()
{
if ((RRTPathPlanner) ||
((manualMode)&&(cartesianMode))) {

//Transform Degrees to Quaternion
jointConfigInQuaternion =
RobotControllerInstance.DegreestoQuate
rnion(jointConfig);

joint1.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint1.localEulerAngles),
jointConfigInQuaternion [0],
rotationSpeed * Time.deltaTime);
joint2.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint2.localEulerAngles),
jointConfigInQuaternion [1],
rotationSpeed * Time.deltaTime);
joint3.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint3.localEulerAngles),
jointConfigInQuaternion [2],
rotationSpeed * Time.deltaTime);
joint4.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint4.localEulerAngles),
jointConfigInQuaternion [3],
rotationSpeed * Time.deltaTime);
joint5.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint5.localEulerAngles),
jointConfigInQuaternion [4],
rotationSpeed * Time.deltaTime);
joint6.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint6.localEulerAngles),
jointConfigInQuaternion [5],
rotationSpeed * Time.deltaTime);

//Calculate the new joint angles
Instantly to know the Target values
and compare below if robot is still
moving or not.
```

```
Quaternion joint1Target =
Quaternion.Lerp (Quaternion.Euler
(joint1.localEulerAngles),
jointConfigInQuaternion [0], 1);
Quaternion joint2Target =
Quaternion.Lerp (Quaternion.Euler
(joint2.localEulerAngles),
jointConfigInQuaternion [1], 1);
Quaternion joint3Target =
Quaternion.Lerp (Quaternion.Euler
(joint3.localEulerAngles),
jointConfigInQuaternion [2], 1);
Quaternion joint4Target =
Quaternion.Lerp (Quaternion.Euler
(joint4.localEulerAngles),
jointConfigInQuaternion [3], 1);
Quaternion joint5Target =
Quaternion.Lerp (Quaternion.Euler
(joint5.localEulerAngles),
jointConfigInQuaternion [4], 1);
Quaternion joint6Target =
Quaternion.Lerp (Quaternion.Euler
(joint6.localEulerAngles),
jointConfigInQuaternion [5], 1);

//true if all joints reach the target
values. Means robot arrived and
stopped moving
int counter = 0;
reachedTargetJoint =
QuaternionAlmostEqual (joint1Target,
joint1.localRotation);
if (reachedTargetJoint) {
counter++;
}
reachedTargetJoint =
QuaternionAlmostEqual (joint2Target,
joint2.localRotation);
if (reachedTargetJoint) {
counter++;
}
reachedTargetJoint =
QuaternionAlmostEqual (joint3Target,
joint3.localRotation);
if (reachedTargetJoint) {
counter++;
}
reachedTargetJoint =
QuaternionAlmostEqual (joint4Target,
joint4.localRotation);
if (reachedTargetJoint) {
counter++;
}
reachedTargetJoint =
QuaternionAlmostEqual (joint5Target,
joint5.localRotation);
if (reachedTargetJoint) {
counter++;
}
reachedTargetJoint =
QuaternionAlmostEqual (joint6Target,
joint6.localRotation);
if (reachedTargetJoint) {
counter++;
}
if (counter == 6) {
reachedTargetJoint = true;
} else {
reachedTargetJoint = false;
}
}

if ((manualMode)&&(jointMode)) {
joint1.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint1.localEulerAngles),
jointConfigInQuaternion [0], 1);
joint2.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint2.localEulerAngles),
jointConfigInQuaternion [1], 1);
joint3.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint3.localEulerAngles),
jointConfigInQuaternion [2], 1);
joint4.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint4.localEulerAngles),
jointConfigInQuaternion [3], 1);
joint5.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint5.localEulerAngles),
jointConfigInQuaternion [4], 1);
joint6.localRotation = Quaternion.Lerp
(Quaternion.Euler
(joint6.localEulerAngles),
jointConfigInQuaternion [5], 1);
}
}

//allows the user to save current
robot position during runtime by
pressing "=" button
void SavePositions()
{
if (Input.GetKeyDown(KeyCode.Equals))
{
Vector3 toolFlangeRotation = new
Vector3(RobotControllerInstance.ax,Rob
otControllerInstance.ay,RobotControlle
rInstance.az);
savedPointsT.Add(toolFlangeLabPosition
* 1000f);
savedPointsR.Add(toolFlangeRotation);
print ("Saved Position" +
toolFlangeLabPosition * 1000f + " in
slot" + savedPointsT.Count);
}
}

public static bool
Vector3AlmostEqual(Vector3 a, Vector3
b, float PercentageDifferenceAllowed){

bool isAlmostEqual = false;
if( (a - b).sqrMagnitude <= (a *
PercentageDifferenceAllowed).sqrMagnit
ude ) {
//Debug.Log( "They are almost equal.
Less than " +
PercentageDifferenceAllowed*100+  "%
different" );
isAlmostEqual = true;
}
return isAlmostEqual;
}
```

```
// Calculates the lab Coordinates
given the Unity coordinates
public Vector3
unitytoRobotCoordinateSystemChange(Vec
tor3 frame)
{
Vector3 unityFrame = frame-
referenceFrameUnityPosition.position;
Vector3 labFrame = new Vector3
(unityFrame.z, unityFrame.x, -
unityFrame.y);
return labFrame;
}

public Vector3
robottoUnityCoordinateSystemChange(Vec
tor3 frame)
{
Vector3 robotFrame = frame;
Vector3 unityFrame = new Vector3
(robotFrame.y, -robotFrame.z,
robotFrame.x) +
referenceFrameUnityPosition.position;
return unityFrame;
}

//Obstacle checks
public int ObstacleDetector(RobotPath
inputPath) {

obstacleSegmentInPath = -1;
obstacleExists = false;
obstacleCollisionImminent = false;
Vector3 currentPos =
toolFlangeLabPosition;
int currentRobotSegment = -1;
currentRobotSegment =
InPositionCounter;
Debug.Log ("Current Segment is: " +
currentRobotSegment);

// Check points of initial programmed
Path. If they are in collision then
the RRT planner cannot find new path
solutionExists = true;
//check for each segment the 1st List
element which is the user defined
points (checkpoints - robot must reach
them) starting from the current
segment of the robot
for (int i = currentRobotSegment; i <
inputPath.SegmentID.Count; i++) {
Vector3 checkPoint =
inputPath.SegmentID
[i].endPoint.cartesian.translationVect
or;
if (Physics.CheckSphere
(robottoUnityCoordinateSystemChange
(checkPoint), 0.02f, 9)) {
Debug.Log ("Target point " + i + " in
Path is in collision - Robot might
STOP");
solutionExists = false;
break;
}
}

//"Remove" from the obstacle check all
points that have been reached
for (int i = 0; i <=
currentRobotSegment; i++) {
//for the current segment that the
robot is moving remove only the points
done
if (i == currentRobotSegment) {
for (int j = 1; j <=
internalPositionCounter; j++) {
inputPath.SegmentID [i].segmentPoint
[j - 1].cartesian.translationVector =
currentPos;
}
}
//for the already executed segments
remove all points
else {
for (int j = 1; j <=
inputPath.SegmentID
[i].segmentPoint.Count; j++) {
inputPath.SegmentID [i].segmentPoint
[j - 1].cartesian.translationVector =
currentPos;
}
}
}

bool collisionInRobotLink = true;
//call the Obstacle detector of the
trajectory planner to check for
obstacles in robot links. Checks the
whole path.
collisionInRobotLink =
TrajectoryPlannerInstance.TrajectoryOb
stacleDetector (inputPath);

//Detect Obstacles
//for each Segment
for (int i = currentRobotSegment; i <
inputPath.SegmentID.Count; i++) {
//for all point in each segment
for (int j = 1; j <=
inputPath.SegmentID
[i].segmentPoint.Count; j++) {
Vector3 internalPoint1 =
inputPath.SegmentID [i].segmentPoint
[j - 1].cartesian.translationVector;
Vector3 internalPoint2;
//if point is the last in the segment
then connect it with the next segment
if (j == inputPath.SegmentID
[i].segmentPoint.Count) {
internalPoint2 = inputPath.SegmentID
[i].endPoint.cartesian.translationVect
or;
} else {
internalPoint2 = inputPath.SegmentID
[i].segmentPoint
[j].cartesian.translationVector;
}
if (Physics.Linecast
(robottoUnityCoordinateSystemChange
(internalPoint1),
robottoUnityCoordinateSystemChange
(internalPoint2), 9)) {
Debug.Log ("Obstacle Detected in Path
Segment " + i);
```

```
obstacleSegmentInPath = i;
obstacleExists = true;
if (currentRobotSegment ==
obstacleSegmentInPath) {
obstacleCollisionImminent = true;
Debug.Log (" Collision Imminent!!! (
and solutionExists = " +
solutionExists + " )");
}
i = inputPath.SegmentID.Count;
break;
}
}
}
return obstacleSegmentInPath;
}

//Spawn Obstacles for testing
void ObstacleSpawner (){
GameObject spawnedObj;
spawnedObj = (GameObject)Instantiate
((GameObject)Obstacle,
robottoUnityCoordinateSystemChange(ini
tialPathT[spawnObstacleInSegment+1]),
Quaternion.identity);
spawnedObj.transform.position =
Vector3.MoveTowards
(robottoUnityCoordinateSystemChange(in
itialPathT[spawnObstacleInSegment+1]),
robottoUnityCoordinateSystemChange
(initialPathT
[spawnObstacleInSegment]), 0.5f);
Debug.Log (" Obstacle Spawned ");
}

//Draw the path for visual debugging
void PathDraw(List<Vector3> Path){

if (instantiatedObjs.Count > 0) {
foreach (GameObject i in
instantiatedObjs) {
GameObject.Destroy (i);
}
instantiatedObjs = new
List<GameObject> ();
}

for (int d = 1; d < (Path.Count - 1);
d++) {
GameObject sphere =
GameObject.CreatePrimitive
(PrimitiveType.Sphere);
sphere.transform.position =
robottoUnityCoordinateSystemChange
(Path [d]);
sphere.transform.localScale = new
Vector3 (0.04f, 0.04f, 0.04f);
sphere.GetComponent<Renderer>
().material.color = Color.green;
sphere.layer=9;
instantiatedObjs.Add (sphere);
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.02f, 0.02f);
OKline.material.color = Color.yellow;
OKline.SetColors (Color.yellow,
Color.yellow);
OKline.SetPosition (0,
robottoUnityCoordinateSystemChange
(Path [d]));
OKline.SetPosition (1,
robottoUnityCoordinateSystemChange
(Path [d + 1]));
instantiatedObjs.Add
(OKline.gameObject);
}
}

public List<Vector3>
DrawCartesianTranslationList(
RobotPath inputPath){
int numberofSegments =
inputPath.SegmentID.Count;
List<Vector3> allPoints = new
List<Vector3> ();
for (int j = 0; j < numberofSegments;
j++) {
int numberofPoints =
inputPath.SegmentID
[j].segmentPoint.Count;
for (int i = 0; i < numberofPoints;
i++) {
Vector3 newPoint = inputPath.SegmentID
[j].segmentPoint
[i].cartesian.translationVector;
allPoints.Add (newPoint);
}
}
allPoints.Add
(inputPath.SegmentID[numberofSegments-
1].endPoint.cartesian.translationVecto
r);
return allPoints;
}

public List<Vector3>
DrawCartesianRotationList( RobotPath
inputPath){
int numberofSegments =
inputPath.SegmentID.Count;
List<Vector3> allPoints = new
List<Vector3> ();
for (int j = 0; j < numberofSegments;
j++) {
int numberofPoints =
inputPath.SegmentID
[j].segmentPoint.Count;
for (int i = 0; i < numberofPoints;
i++) {
Vector3 newPoint = inputPath.SegmentID
[j].segmentPoint
[i].cartesian.rotationVector;
allPoints.Add (newPoint);
}
}
allPoints.Add
(inputPath.SegmentID.Last().endPoint.c
artesian.rotationVector);
return allPoints;
}
}
```

Robot Controller.cs

```
using UnityEngine;
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Threading;



public class RobotController
{
#region Variables
// Robot joint angels for initial
position
[Range (-160f,160f)] public float
rotZ1 = 0;
[Range (-227.5f,47.5f)] public float
rotZ2 = -90;
[Range (-52.5f,232.5f)] public float
rotZ3 = 90;
[Range (-270f,270f)] public float
rotZ4 = 0;
[Range (-105f,120f)] public float
rotZ5 = 0;
[Range (-270f,270f)] public float
rotZ6 = 0;

     public            List<Quaternion>
IKCalculatedJointRotations    =    new
List<Quaternion>();
public bool outofDexterousSpaceFlag;
public List<float[]>
candidateJointSolutions = new
List<float[]>();

public float rotX1 = -90f;
public float rotX2 = -90f;
public float rotX3 = 0f;
public float rotX4 = 90f;
public float rotX5 = -90f;
public float rotX6 = 90f;
public float rotYAll = 0f;
public float initRotZ1 = 0f;
public float initRotZ2 = -90f;
public float initRotZ3 = 90f;
public float initRotZ4 = 0f;
public float initRotZ5 = 0f;
public float initRotZ6 = 0f;

// DO READY Rotations of joints
public Quaternion j1Def;
public Quaternion j2Def;
public Quaternion j3Def;
public Quaternion j4Def;
public Quaternion j5Def;
public Quaternion j6Def;

// Target Rotations of joints
public Quaternion j1Rot;
public Quaternion j2Rot;
public Quaternion j3Rot;
public Quaternion j4Rot;
public Quaternion j5Rot;
public Quaternion j6Rot;

//Rotation Matrix Rx
private float rx1;
private float rx2;
private float rx3;
private float rx4;
private float rx5;
private float rx6;
private float rx7;
private float rx8;
private float rx9;

//Rotation Matrix Ry
private float ry1;
private float ry2;
private float ry3;
private float ry4;
private float ry5;
private float ry6;
private float ry7;
private float ry8;
private float ry9;

//Rotation Matrix Rz
private float rz1;
private float rz2;
private float rz3;
private float rz4;
private float rz5;
private float rz6;
private float rz7;
private float rz8;
private float rz9;

//variables used for the translation
matrix
public float pwx=0f;
public float pwy=0f;
public float pwz=1.1f;
public float ax=0f;
public float ay=0f;
public float az=0f;
public float r13;
public float r23;
public float r33;
private float r11;
private float r21;
private float r31;

// All possible solutions of joints
private float q11;
private float q12;
private float q31;
private float q32;
private float q33;
private float q34;
private float q21;
private float q22;
private float q23;
private float q24;
private float q25;
private float q26;
private float q27;
private float q28;
private float q51;
private float q52;
```

```
private float q53;
private float q54;
private float q55;
private float q56;
private float q57;
private float q58;
private float q59;
private float q510;
private float q511;
private float q512;
private float q513;
private float q514;
private float q515;
private float q516;
private float q41;
private float q42;
private float q43;
private float q44;
private float q45;
private float q46;
private float q47;
private float q48;
private float q49;
private float q410;
private float q411;
private float q412;
private float q413;
private float q414;
private float q415;
private float q416;
private float q61;
private float q62;
private float q63;
private float q64;
private float q65;
private float q66;
private float q67;
private float q68;
private float q69;
private float q610;
private float q611;
private float q612;
private float q613;
private float q614;
private float q615;
private float q616;
private float q617;
private float q618;
private float q619;
private float q620;
private float q621;
private float q622;
private float q623;
private float q624;
private float q625;
private float q626;
private float q627;
private float q628;
private float q629;
private float q630;
private float q631;
private float q632;
private float q633;
private float q634;
private float q635;
private float q636;
private float q637;
private float q638;
private float q639;
private float q640;
private float q641;
private float q642;
private float q643;
private float q644;
private float q645;
private float q646;
private float q647;
private float q648;

private float q1prev = 0f;
private float q2prev= -90f;
private float q3prev= 90f;
private float q4prev= 0f;
private float q5prev= 0f;
private float q6prev= 0f;

private float minSum;
private float sum;
#endregion



//Linear Interpolator for Straight
Line Motion
public Vector3
LinearInterpolator(Vector3 startPoint,
Vector3 endPoint){
 Vector3 intermediatePoint =
Vector3.MoveTowards ( startPoint,
endPoint, Time.deltaTime * 0.1f);
 return intermediatePoint;
 }

public Vector3
AngleInterpolator(Vector3 startPoint,
Vector3 endPoint){
 Vector3 intermediateAngle =
Vector3.MoveTowards ( startPoint,
endPoint, Time.deltaTime * 7f);
 return intermediateAngle;
 }

// IK model offers the following
movement modes
// 1) Null - Robot moves in joint
space according to the IK
// 2) LIN - Robot moves in a straight
line
// 3) PLAN - Robot calculates Joints
for trajectory planning with criteria
the joints to not go close to limits
public  List<float[]>
IKcontroller(Vector3 T, Vector3 R,
string mode=null)
{
//Assign the desired euler angles
 if (mode == "LIN") {
 Vector3 intermediatePoint =
AngleInterpolator (new Vector3 (ax,
ay, az), new Vector3 (R.x, R.y, R.z));
 ax = intermediatePoint.x;
 ay = intermediatePoint.y;
 az = intermediatePoint.z;
 }
 else {
```

```
ax = R.x;
ay = R.y;
az = R.z;
}

//Create the Rotation Matrix
//Matrix Rx
rx2 = 0f;
rx3 = 0f;
rx4 = 0f;
rx5 = Mathf.Cos (ax * Mathf.Deg2Rad);
rx6 = -Mathf.Sin (ax * Mathf.Deg2Rad);
rx7 = 0f;
rx8 = Mathf.Sin (ax * Mathf.Deg2Rad);
rx9 = Mathf.Cos (ax * Mathf.Deg2Rad);

//Matrix Ry
ry1 = Mathf.Cos (ay * Mathf.Deg2Rad);
ry2 = 0f;
ry3 = Mathf.Sin (ay * Mathf.Deg2Rad);
ry4 = 0f;
ry5 = 1f;
ry6 = 0f;
ry7 = -Mathf.Sin (ay * Mathf.Deg2Rad);
ry8 = 0f;
ry9 = Mathf.Cos (ay * Mathf.Deg2Rad);

//Matrix Rz
rz1 = Mathf.Cos (az * Mathf.Deg2Rad);
rz2 = -Mathf.Sin (az * Mathf.Deg2Rad);
rz3 = 0f;
rz4 = Mathf.Sin (az * Mathf.Deg2Rad);
rz5 = Mathf.Cos (az * Mathf.Deg2Rad);
rz6 = 0f;
rz7 = 0f;
rz8 = 0f;
rz9 = 1f;

//Rx and Rz of the final rotation
matrix
r13 = ry3 * (rx1 * rz1 + rx4 * rz2 +
rx7 * rz3) + ry6 * (rx2 * rz1 + rx5 *
rz2 + rx8 * rz3) + ry9 * (rx3 * rz1 +
rx6 * rz2 + rx9 * rz3);
r23 = ry3 * (rx1 * rz4 + rx4 * rz5 +
rx7 * rz6) + ry6 * (rx2 * rz4 + rx5 *
rz5 + rx8 * rz6) + ry9 * (rx3 * rz4 +
rx6 * rz5 + rx9 * rz6);
r33 = ry3 * (rx1 * rz7 + rx4 * rz8 +
rx7 * rz9) + ry6 * (rx2 * rz7 + rx5 *
rz8 + rx8 * rz9) + ry9 * (rx3 * rz7 +
rx6 * rz8 + rx9 * rz9);
r11 = ry1 * (rx1 * rz1 + rx4 * rz2 +
rx7 * rz3) + ry4 * (rx2 * rz1 + rx5 *
rz2 + rx8 * rz3) + ry7 * (rx3 * rz1 +
rx6 * rz2 + rx9 * rz3);
r21 = ry1 * (rx1 * rz4 + rx4 * rz5 +
rx7 * rz6) + ry4 * (rx2 * rz4 + rx5 *
rz5 + rx8 * rz6) + ry7 * (rx3 * rz4 +
rx6 * rz5 + rx9 * rz6);
r31 = ry1 * (rx1 * rz7 + rx4 * rz8 +
rx7 * rz9) + ry4 * (rx2 * rz7 + rx5 *
rz8 + rx8 * rz9) + ry7 * (rx3 * rz7 +
rx6 * rz8 + rx9 * rz9);


//Transfrom from Tool position (user
input) to Wrist position (IK
calculation input) in order to run the
Inverse Kinematic using Piepers method
// Pw = P-d*a where Pw is the wrist
position
// P is the endeffector position
// d is the distance of the wrist to
the endeffector
// a is the Rz matrix [ r13
//    r23
//    r33 ]
Vector3 normalizedRz = new Vector3
(r13, r23, r33).normalized;

//LIN - Linear move in straight line
if (mode == "LIN") {
float targetpwx=T.x -
0.085f*normalizedRz.x;
float targetpwy=T.y -
0.085f*normalizedRz.y;
float targetpwz=T.z -
0.085f*normalizedRz.z;

Vector3 intermediatePoint =
LinearInterpolator (new Vector3 (pwx,
pwy, pwz), new Vector3 (targetpwx,
targetpwy, targetpwz));
pwx = intermediatePoint.x;
pwy = intermediatePoint.y;
pwz = intermediatePoint.z;
}
else {
pwx=T.x - 0.085f*normalizedRz.x;
pwy=T.y - 0.085f*normalizedRz.y;
pwz=T.z - 0.085f*normalizedRz.z;
}
#region IKControllerMath

//*** Calculation of all joint
angles***//
// Calculate q1
q11 = Mathf.Rad2Deg * Mathf.Atan2
(pwy, pwx);
q12 = Mathf.Rad2Deg * (Mathf.Atan2 (-
pwy, -pwx));

// Calculate q3
q31 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round (((Mathf.Pow(pwx, 2f) +
Mathf.Pow(pwy, 2f) + Mathf.Pow(pwz,
2f) - Mathf.Pow(0.45f, 2f)
- Mathf.Pow(0.65f, 2f)) / (2f * 0.45f
* 0.65f))*1000f)/1000f);
q32 =  q31;
q33 = 180f - Mathf.Rad2Deg *
Mathf.Asin (Mathf.Round
(((Mathf.Pow(pwx, 2f) + Mathf.Pow(pwy,
2f) + Mathf.Pow(pwz, 2f)
- Mathf.Pow(0.45f, 2f) -
Mathf.Pow(0.65f, 2f)) / (2f * 0.45f *
0.65f))*1000f)/1000f);
q34 = 180f + Mathf.Rad2Deg *
Mathf.Asin (Mathf.Round
(((Mathf.Pow(pwx, 2f) + Mathf.Pow(pwy,
2f) + Mathf.Pow(pwz, 2f)
- Mathf.Pow(0.45f, 2f) -
Mathf.Pow(0.65f, 2f)) / (2f * 0.45f *
0.65f))*1000f)/1000f);
```

```
// Calculate q2
q21 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q31 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q31 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q11 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q11 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q31 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q31 *
Mathf.Deg2Rad) * (Mathf.Cos (q11 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q11
* Mathf.Deg2Rad) * pwy))) - q31;

 q22 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q32 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q32 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q11 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q11 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q32 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q32 *
Mathf.Deg2Rad) * (Mathf.Cos (q11 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q11
* Mathf.Deg2Rad) * pwy))) - q32;

 q23 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q33 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q33 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q11 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q11 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q33 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q33 *
Mathf.Deg2Rad) * (Mathf.Cos (q11 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q11
* Mathf.Deg2Rad) * pwy))) - q33;

 q24 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q34 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q34 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q11 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q11 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q34 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q34 *
Mathf.Deg2Rad) * (Mathf.Cos (q11 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q11
* Mathf.Deg2Rad) * pwy))) - q34;

 q25 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q31 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q31 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q12 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q12 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q31 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q31 *
Mathf.Deg2Rad) * (Mathf.Cos (q12 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q12
* Mathf.Deg2Rad) * pwy))) - q31;

 q26 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q32 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q32 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q12 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q12 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q32 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q32 *
Mathf.Deg2Rad) * (Mathf.Cos (q12 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q12
* Mathf.Deg2Rad) * pwy))) - q32;

 q27 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q33 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q33 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q12 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q12 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q33 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q33 *
Mathf.Deg2Rad) * (Mathf.Cos (q12 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q12
* Mathf.Deg2Rad) * pwy))) - q33;

 q28 = (Mathf.Rad2Deg * Mathf.Atan2 (-
0.45f * Mathf.Cos (q34 *
Mathf.Deg2Rad) * pwz + (0.65f + 0.45f
* Mathf.Sin (q34 * Mathf.Deg2Rad)) *
 (Mathf.Cos (q12 * Mathf.Deg2Rad) * pwx
+ Mathf.Sin (q12 * Mathf.Deg2Rad) *
pwy),
 (0.65f + 0.45f * Mathf.Sin (q34 *
Mathf.Deg2Rad)) * pwz +
 0.45f * Mathf.Cos (q34 *
Mathf.Deg2Rad) * (Mathf.Cos (q12 *
Mathf.Deg2Rad) * pwx + Mathf.Sin (q12
* Mathf.Deg2Rad) * pwy))) - q34;

 //Calculate q5
 q51 = Mathf.Rad2Deg * Mathf.Acos
 (Mathf.Round ((Mathf.Cos
 (q11*Mathf.Deg2Rad)*Mathf.Sin
 (q21*Mathf.Deg2Rad +
 q31*Mathf.Deg2Rad)*r13 +
 Mathf.Sin
 (q11*Mathf.Deg2Rad)*Mathf.Sin
 (q21*Mathf.Deg2Rad +
 q31*Mathf.Deg2Rad)*r23 +
 Mathf.Cos (q21*Mathf.Deg2Rad +
 q31*Mathf.Deg2Rad)*r33)*1000f)/1000f);
 q52 = -q51;

 q53 = Mathf.Rad2Deg * Mathf.Acos
 (Mathf.Round ((Mathf.Cos
 (q11*Mathf.Deg2Rad)*Mathf.Sin
 (q22*Mathf.Deg2Rad +
 q32*Mathf.Deg2Rad)*r13 +
 Mathf.Sin
 (q11*Mathf.Deg2Rad)*Mathf.Sin
 (q22*Mathf.Deg2Rad +
 q32*Mathf.Deg2Rad)*r23 +
```

```
Mathf.Cos (q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r33)*1000f)/1000f);
q54 = -q53;

q55 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r33)*1000f)/1000f);

q56 = -q55;

q57 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r33)*1000f)/1000f);

q58 = -q57;

q59 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r33)*1000f)/1000f);
q510 = -q59;

q511 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r33)*1000f)/1000f);

q512 = -q511;

q513 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r33)*1000f)/1000f);

q514 = -q513;

q515 = Mathf.Rad2Deg * Mathf.Acos
(Mathf.Round ((Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r23 +
Mathf.Cos (q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r33)*1000f)/1000f);

q516 = -q515;

// Calculate q4
if (Mathf.Sin (q51) == 0f)
{
q41 = q4prev;
q42 = q4prev;
}
else
{
q41 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q11*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q11 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r33));
if (q41 <= 90f && q41 >= 0f)
{
q42 = q41 + 180f;
}
else if (q41 <= 0f && q41 >= -90f)
{
q42 = q41 - 180f;
}
else
{
q42 = q41;
}
}

if (Mathf.Sin (q53) == 0f)
{
q43 = q4prev;
q44 = q4prev;
}
else
{
q43 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q11*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q11 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r13 +
```

```
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r33));
if (q43 <= 90f && q43 >= 0f)
{
q44 = q43 + 180f;
}
else if (q43 <= 0f && q43 >= -90f)
{
q44 = q43 - 180f;
}
else
{
q44 = q43;
}
}

if (Mathf.Sin (q55) == 0f)
{
q45 = q4prev;
q46 = q4prev;
}
else
{
q45 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q11*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q11 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r33));
if (q45 <= 90f && q45 >= 0f)
{
q46 = q45 + 180f;
}
else if (q45 <= 0f && q45 >= -90f)
{
q46 = q45 - 180f;
}
else
{
q46 = q45;
}
}

if (Mathf.Sin (q57) == 0f)
{
q47 = q4prev;
q48 = q4prev;
}
else
{
q47 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q11*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q11 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r13 +
```

```
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r33));
if (q47 <= 90f && q47 >= 0f)
{
q48 = q47 + 180f;
}
else if (q47 <= 0f && q47 >= -90f)
{
q48 = q47 - 180f;
}
else
{
q48 = q47;
}
}

if (Mathf.Sin (q59) == 0f)
{
q49 = q4prev;
q410 = q4prev;
}
else
{
q49 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q12*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q12 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r33));
if (q49 <= 90f && q49 >= 0f)
{
q410 = q49 + 180f;
}
else if (q49 <= 0f && q49 >= -90f)
{
q410 = q49 - 180f;
}
else
{
q410 = q49;
}
}

if (Mathf.Sin (q511) == 0f)
{
q411 = q4prev;
q412 = q4prev;
}
else
{
q411 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q12*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q12 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r13 +
```

```
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r33));
if (q411 <= 90f && q411 >= 0f)
{
q412 = q411 + 180f;
}
else if (q411 <= 0f && q411 >= -90f)
{
q412 = q411 - 180f;
}
else
{
q412 = q411;
}
}

if (Mathf.Sin (q513) == 0f)
{
q413 = q4prev;
q414 = q4prev;
}
else
{
q413 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q12*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q12 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r33));
if (q413 <= 90f && q413 >= 0f)
{
q414 = q413 + 180f;
}
else if (q413 <= 0f && q413 >= -90f)
{
q414 = q413 - 180f;
}
else
{
q414 = q413;
}
}

if (Mathf.Sin (q515) == 0f)
{
q415 = q4prev;
q416 = q4prev;
}
else
{
q415 = Mathf.Rad2Deg * (Mathf.Atan2(-
Mathf.Sin (q12*Mathf.Deg2Rad)*r13 +
Mathf.Cos (q12 * Mathf.Deg2Rad)*r23,
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r13 +
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r23 -
Mathf.Sin (q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r33));
if (q415 <= 90f && q415 >= 0f)
{
q416 = q415 + 180f;
}
else if (q415 <= 0f && q415 >= -90f)
{
q416 = q415 - 180f;
}
else
{
q416 = q415;
}
}

// Calculate q6
q61 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q41*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q41*Mathf.Deg2Rad)*Mathf.Cos
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q41*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q41*Mathf.Deg2Rad)*Mathf.Cos
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q41*Mathf.Deg2Rad)*Mathf.Sin
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q61 >= 0f)
{
q62 = 180f - q61;
q63 = -180f - q61;
}
else
{
q62 = -180f - q61;
q63 = 180f - q61;
}

q64 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q42*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q42*Mathf.Deg2Rad)*Mathf.Cos
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q42*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q42*Mathf.Deg2Rad)*Mathf.Cos
```

```
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q42*Mathf.Deg2Rad)*Mathf.Sin
(q21*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q64 >= 0f)
{
q65 = 180f - q64;
q66 = -180f - q64;
}
else
{
q65 = -180f - q64;
q66 = 180f - q64;
}

q67 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q43*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q43*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q43*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q43*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q43*Mathf.Deg2Rad)*Mathf.Sin
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q67 >= 0f)
{
q68 = 180f - q67;
q69 = -180f - q67;
}
else
{
q68 = -180f - q67;
q69 = 180f - q67;
}

q610 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q44*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q44*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q44*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q44*Mathf.Deg2Rad)*Mathf.Cos
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q44*Mathf.Deg2Rad)*Mathf.Sin
(q22*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q610 >= 0f)
{
q611 = 180f - q610;
q612 = -180f - q610;
}
else
{
q611 = -180f - q610;
q612 = 180f - q610;
}

q613 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q45*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q45*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q45*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q45*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q45*Mathf.Deg2Rad)*Mathf.Sin
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q613 >= 0f)
{
q614 = 180f - q613;
q615 = -180f - q613;
}
else
{
q614 = -180f - q613;
q615 = 180f - q613;
}

q616 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q46*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q46*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q46*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q46*Mathf.Deg2Rad)*Mathf.Cos
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q46*Mathf.Deg2Rad)*Mathf.Sin
(q23*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q616 >= 0f)
{
```

```
q617 = 180f - q616;
q618 = -180f - q616;
}
else
{
q617 = -180f - q616;
q618 = 180f - q616;
}

q619 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q47*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q47*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q47*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q47*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q47*Mathf.Deg2Rad)*Mathf.Sin
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q619 >= 0f)
{
q620 = 180f - q619;
q621 = -180f - q619;
}
else
{
q620 = -180f - q619;
q621 = 180f - q619;
}

q622 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q11*Mathf.Deg2Rad)*Mathf.Co
s (q48*Mathf.Deg2Rad) +
Mathf.Cos
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q48*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q11*Mathf.Deg2Rad)*Mathf.Co
s (q48*Mathf.Deg2Rad) -
Mathf.Sin
(q11*Mathf.Deg2Rad)*Mathf.Sin
(q48*Mathf.Deg2Rad)*Mathf.Cos
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q48*Mathf.Deg2Rad)*Mathf.Sin
(q24*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q622 >= 0f)
{
q623 = 180f - q622;
q624 = -180f - q622;
}
else
{
q623 = -180f - q622;
q624 = 180f - q622;
}

q625 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q49*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q49*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q49*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q49*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q49*Mathf.Deg2Rad)*Mathf.Sin
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q625 >= 0f)
{
q626 = 180f - q625;
q627 = -180f - q625;
}
else
{
q626 = -180f - q625;
q627 = 180f - q625;
}

q628 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q410*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q410*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q410*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q410*Mathf.Deg2Rad)*Mathf.Cos
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q410*Mathf.Deg2Rad)*Mathf.Sin
(q25*Mathf.Deg2Rad +
q31*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q628 >= 0f)
{
q629 = 180f - q628;
q630 = -180f - q628;
}
else
{
q629 = -180f - q628;
q630 = 180f - q628;
}
```

```
q631 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q411*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q411*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q411*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q411*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q411*Mathf.Deg2Rad)*Mathf.Sin
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q631 >= 0f)
{
q632 = 180f - q631;
q633 = -180f - q631;
}
else
{
q632 = -180f - q631;
q633 = 180f - q631;
}

q634 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q412*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q412*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q412*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q412*Mathf.Deg2Rad)*Mathf.Cos
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q412*Mathf.Deg2Rad)*Mathf.Sin
(q26*Mathf.Deg2Rad +
q32*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q634 >= 0f)
{
q635 = 180f - q634;
q636 = -180f - q634;
}
else
{
q635 = -180f - q634;
q636 = 180f - q634;
}

q637 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q413*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q413*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q413*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q413*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q413*Mathf.Deg2Rad)*Mathf.Sin
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q637 >= 0f)
{
q638 = 180f - q637;
q639 = -180f - q637;
}
else
{
q638 = -180f - q637;
q639 = 180f - q637;
}

q640 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q414*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q414*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q414*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q414*Mathf.Deg2Rad)*Mathf.Cos
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q414*Mathf.Deg2Rad)*Mathf.Sin
(q27*Mathf.Deg2Rad +
q33*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q640 >= 0f)
{
q641 = 180f - q640;
q642 = -180f - q640;
}
else
{
q641 = -180f - q640;
q642 = 180f - q640;
}

q643 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q415*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
```

```
(q415*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q415*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q415*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q415*Mathf.Deg2Rad)*Mathf.Sin
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q643 >= 0f)
{
q644 = 180f - q643;
q645 = -180f - q643;
}
else
{
q644 = -180f - q643;
q645 = 180f - q643;
}

q646 = Mathf.Rad2Deg * Mathf.Asin
(Mathf.Round ((-
(Mathf.Sin(q12*Mathf.Deg2Rad)*Mathf.Co
s (q416*Mathf.Deg2Rad) +
Mathf.Cos
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q416*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r11 +

(Mathf.Cos(q12*Mathf.Deg2Rad)*Mathf.Co
s (q416*Mathf.Deg2Rad) -
Mathf.Sin
(q12*Mathf.Deg2Rad)*Mathf.Sin
(q416*Mathf.Deg2Rad)*Mathf.Cos
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad))*r21 +
Mathf.Sin
(q416*Mathf.Deg2Rad)*Mathf.Sin
(q28*Mathf.Deg2Rad +
q34*Mathf.Deg2Rad)*r31)*1000f)/1000f);
if (q646 >= 0f)
{
q647 = 180f - q646;
q648 = -180f - q646;
}
else
{
q647 = -180f - q646;
q648 = 180f - q646;
}
#endregion

//Create solutions Array
float[][] IKSolutionsArray= new
float[][]
{
new float[]{q11,q21,q31,q41,q51,q61},
new float[]{q11,q21,q31,q41,q51,q62},
new float[]{q11,q21,q31,q41,q51,q63},
new float[]{q11,q21,q31,q42,q51,q64},
new float[]{q11,q21,q31,q42,q51,q65},
new float[]{q11,q21,q31,q42,q51,q66},
new float[]{q11,q21,q31,q41,q52,q61},
new float[]{q11,q21,q31,q41,q52,q62},
new float[]{q11,q21,q31,q41,q52,q63},
new float[]{q11,q21,q31,q42,q52,q64},
new float[]{q11,q21,q31,q42,q52,q65},
new float[]{q11,q21,q31,q42,q52,q66},
new float[]{q11,q22,q32,q43,q53,q67},
new float[]{q11,q22,q32,q43,q53,q68},
new float[]{q11,q22,q32,q43,q53,q69},
new float[]{q11,q22,q32,q44,q53,q610},
new float[]{q11,q22,q32,q44,q53,q611},
new float[]{q11,q22,q32,q44,q53,q612},
new float[]{q11,q22,q32,q43,q54,q67},
new float[]{q11,q22,q32,q43,q54,q68},
new float[]{q11,q22,q32,q44,q54,q610},
new float[]{q11,q22,q32,q44,q54,q611},
new float[]{q11,q22,q32,q44,q54,q612},
new float[]{q11,q23,q33,q45,q55,q613},
new float[]{q11,q23,q33,q45,q55,q614},
new float[]{q11,q23,q33,q45,q55,q615},
new float[]{q11,q23,q33,q46,q55,q616},
new float[]{q11,q23,q33,q46,q55,q617},
new float[]{q11,q23,q33,q46,q55,q618},
new float[]{q11,q23,q33,q45,q56,q613},
new float[]{q11,q23,q33,q45,q56,q614},
new float[]{q11,q23,q33,q45,q56,q615},
new float[]{q11,q23,q33,q46,q56,q616},
new float[]{q11,q23,q33,q46,q56,q617},
new float[]{q11,q23,q33,q46,q56,q618},
new float[]{q11,q24,q34,q47,q57,q619},
new float[]{q11,q24,q34,q47,q57,q620},
new float[]{q11,q24,q34,q47,q57,q621},
new float[]{q11,q24,q34,q48,q57,q622},
new float[]{q11,q24,q34,q48,q57,q623},
new float[]{q11,q24,q34,q48,q57,q624},
new float[]{q11,q24,q34,q47,q58,q619},
new float[]{q11,q24,q34,q47,q58,q620},
new float[]{q11,q24,q34,q47,q58,q621},
new float[]{q11,q24,q34,q48,q58,q622},
new float[]{q11,q24,q34,q48,q58,q623},
new float[]{q11,q24,q34,q48,q58,q624},
new float[]{q12,q25,q31,q49,q59,q625},
new float[]{q12,q25,q31,q49,q59,q626},
new float[]{q12,q25,q31,q49,q59,q627},
new
float[]{q12,q25,q31,q410,q59,q628},
new
float[]{q12,q25,q31,q410,q59,q629},
new
float[]{q12,q25,q31,q410,q59,q630},
new
float[]{q12,q25,q31,q49,q510,q625},
new
float[]{q12,q25,q31,q49,q510,q626},
new
float[]{q12,q25,q31,q49,q510,q627},
new
float[]{q12,q25,q31,q410,q510,q628},
new
float[]{q12,q25,q31,q410,q510,q629},
new
float[]{q12,q25,q31,q410,q510,q630},
new
float[]{q12,q26,q32,q411,q511,q631},
new
float[]{q12,q26,q32,q411,q511,q632},
```

```
new
float[]{q12,q26,q32,q411,q511,q633},
new
float[]{q12,q26,q32,q412,q511,q634},
new
float[]{q12,q26,q32,q412,q511,q635},
new
float[]{q12,q26,q32,q412,q511,q636},
new
float[]{q12,q26,q32,q411,q512,q631},
new
float[]{q12,q26,q32,q411,q512,q632},
new
float[]{q12,q26,q32,q411,q512,q633},
new
float[]{q12,q26,q32,q412,q512,q634},
new
float[]{q12,q26,q32,q412,q512,q635},
new
float[]{q12,q26,q32,q412,q512,q636},
new
float[]{q12,q27,q33,q413,q513,q637},
new
float[]{q12,q27,q33,q413,q513,q638},
new
float[]{q12,q27,q33,q413,q513,q639},
new
float[]{q12,q27,q33,q414,q513,q640},
new
float[]{q12,q27,q33,q414,q513,q641},
new
float[]{q12,q27,q33,q414,q513,q642},
new
float[]{q12,q27,q33,q413,q514,q637},
new
float[]{q12,q27,q33,q413,q514,q638},
new
float[]{q12,q27,q33,q413,q514,q639},
new
float[]{q12,q27,q33,q414,q514,q640},
new
float[]{q12,q27,q33,q414,q514,q641},
new
float[]{q12,q27,q33,q414,q514,q642},
new
float[]{q12,q28,q34,q415,q515,q643},
new
float[]{q12,q28,q34,q415,q515,q644},
new
float[]{q12,q28,q34,q415,q515,q645},
new
float[]{q12,q28,q34,q416,q515,q646},
new
float[]{q12,q28,q34,q416,q515,q647},
new
float[]{q12,q28,q34,q416,q515,q648},
new
float[]{q12,q28,q34,q415,q516,q643},
new
float[]{q12,q28,q34,q415,q516,q644},
new
float[]{q12,q28,q34,q415,q516,q645},
new
float[]{q12,q28,q34,q416,q516,q646},
new
float[]{q12,q28,q34,q416,q516,q647},
new
float[]{q12,q28,q34,q416,q516,q648},
};

//Calculate the sum of joint angles of
each solution and choose the
configuration with the smallest
deviation from the previous position
sum = Mathf.Infinity;
minSum = sum;
float[] jointSolutionDeg = new
float[6];
candidateJointSolutions.Clear ();

//Iterate 95 times for all solutions
for (int
i=0;i<IKSolutionsArray.GetLength(0);i+
+){
if
(Q1InsideLimits(IKSolutionsArray[i][0]
) &&
Q2InsideLimits(IKSolutionsArray[i][1])
&&
Q3InsideLimits(IKSolutionsArray[i][2])
&&
Q4InsideLimits(IKSolutionsArray[i][3])
&&
Q5InsideLimits(IKSolutionsArray[i][4])
&&
Q6InsideLimits(IKSolutionsArray[i][5])
){
//Sum_i{(Qprev-Qi)^2} - The sum of
squared joint values is the primary
metric that is used in picking one of
the eight solutions
//This works well if at least one of
the solutions has all of its joints
values within the joint limits
sum1 = Mathf.Pow(q1prev -
IKSolutionsArray[i][0],2f) + Mathf.Pow
(q2prev - IKSolutionsArray[i][1],2f) +
Mathf.Pow (q3prev -
IKSolutionsArray[i][2],2f) + Mathf.Pow
(q4prev - IKSolutionsArray[i][3],2f) +
Mathf.Pow (q5prev -
IKSolutionsArray[i][4],2f) +
Mathf.Pow(q6prev -
IKSolutionsArray[i][5],2f);
sum2 =
(Mathf.Pow(IKSolutionsArray[i][0]/(160
f - (- 160f)),2) +
Mathf.Pow(IKSolutionsArray[i][1]-(-
90f)/(47.5f-(-227.5f)),2) +
Mathf.Pow(IKSolutionsArray[i][2]-
90f/(232.5f-(-52.5f)),2) +
Mathf.Pow(IKSolutionsArray[i][3]/(270f
-(-270f)),2) +
Mathf.Pow(IKSolutionsArray[i][4]-
7.5f/(120f-(-105f)),2) +
Mathf.Pow(IKSolutionsArray[i][5]/(270f
-(-270f)),2))/6;
sum = 0.6f*sum1 + 0.4f*sum2;

//keep solutions
candidateJointSolutions.Add
(IKSolutionsArray [i]);

if (sum <= minSum) {
minSum = sum;
rotZ1 = IKSolutionsArray [i][0];
```

```
 rotZ2 = IKSolutionsArray [i][1];
 rotZ3 = IKSolutionsArray [i][2];
 rotZ4 = IKSolutionsArray [i][3];
 rotZ5 = IKSolutionsArray [i][4];
 rotZ6 = IKSolutionsArray [i][5];
 jointSolutionDeg = new float[]{
rotZ1, rotZ2, rotZ3, rotZ4, rotZ5,
rotZ6 };
 }
 }
 else if (minSum==Mathf.Infinity) {
 jointSolutionDeg = new
float[]{q1prev,q2prev,q3prev,q4prev,q5
prev,q6prev};
 candidateJointSolutions.Add (new
float[]{q1prev,q2prev,q3prev,q4prev,q5
prev,q6prev});
 }
 }

 //Sort the best #  solutions according
to criteria if the IK is called for
trajectory planning
 if (mode == "PLAN") {

 int solutionsToKeep = 20;
 float tempsum;
 for (int i=0;i<solutionsToKeep;i++){
 tempsum = Mathf.Infinity;
 for (int j = i; j<
candidateJointSolutions.Count;j++){
 sum2 =
(Mathf.Pow(candidateJointSolutions[j][
0]/(160f - (- 160f)),2) +
Mathf.Pow(candidateJointSolutions[j][1
]-(-90f)/(47.5f-(-227.5f)),2) +
Mathf.Pow(candidateJointSolutions[j][2
]-90f/(232.5f-(-52.5f)),2) +
Mathf.Pow(candidateJointSolutions[j][3
]/(270f-(-270f)),2) +
Mathf.Pow(candidateJointSolutions[j][4
]-7.5f/(120f-(-105f)),2) +
Mathf.Pow(candidateJointSolutions[j][5
]/(270f-(-270f)),2))/6;
 sum = sum2;
 if (sum < tempsum) {
 tempsum = sum;
 candidateJointSolutions.Insert (i,
candidateJointSolutions [j]);
 candidateJointSolutions.RemoveAt (j +
1);
 }
 else if (sum == tempsum) {
 candidateJointSolutions.RemoveAt (j);
 j--;
 }
 }
 }
 //Remove rest solutions
 if (candidateJointSolutions.Count >=
solutionsToKeep) {
 candidateJointSolutions.RemoveRange
(solutionsToKeep,
candidateJointSolutions.Count -
solutionsToKeep);
 }
 }
```

```
 if (Input.GetKey (KeyCode.Space)) {
 CalculateDoReady();
 }

 if (Mathf.Abs (q3prev - rotZ3) < 85f)
 {
 //Save new position as previous
 q1prev = rotZ1;
 q2prev = rotZ2;
 q3prev = rotZ3;
 q4prev = rotZ4;
 q5prev = rotZ5;
 q6prev = rotZ6;
 }

 else {
 rotZ1 = q1prev;
 rotZ2 = q2prev;
 rotZ3 = q3prev;
 rotZ4 = q4prev;
 rotZ5 = q5prev;
 rotZ6 = q6prev;
 }

 if (minSum == Mathf.Infinity) {
 Debug.Log ("outofDexterousSpaceFlag =
True");
 outofDexterousSpaceFlag = true;
 }
 else {
 outofDexterousSpaceFlag = false;
 }

 return candidateJointSolutions;
 }

 public  void CalculateDoReady()
 {
 rotZ1 = initRotZ1;
 rotZ2 = initRotZ2;
 rotZ3 = initRotZ3;
 rotZ4 = initRotZ4;
 rotZ5 = initRotZ5;
 rotZ6 = initRotZ6;

 j1Rot = j1Def;
 j2Rot = j2Def;
 j3Rot = j3Def;
 j4Rot = j4Def;
 j5Rot = j5Def;
 j6Rot = j6Def;

 q1prev = rotZ1;
 q2prev = rotZ2;
 q3prev = rotZ3;
 q4prev = rotZ4;
 q5prev = rotZ5;
 q6prev = rotZ6;

 pwx = 0f;
 pwy = 0f;
 pwz = 1.1f;
 ax = 0f;
 ay = 0f;
 az = 0f;
 }
```

```
// Calculates target position of the
robot in Quaternion
public Quaternion[]
DegreestoQuaternion(float[]
jointInDegrees)
{
Quaternion[] jointSolutionQua = new
Quaternion[6];
IKCalculatedJointRotations.Clear ();
//new angle is old + difference of the
new from the initial with inverted
sign to match Unity's coordinate
system
j1Rot = Quaternion.Euler (rotX1,
rotYAll, initRotZ1 -
(jointInDegrees[0]-initRotZ1));
j2Rot = Quaternion.Euler (rotX2,
rotYAll, initRotZ2 -
(jointInDegrees[1]-initRotZ2));
j3Rot = Quaternion.Euler (rotX3,
rotYAll, initRotZ3 -
(jointInDegrees[2]-initRotZ3));
j4Rot = Quaternion.Euler (rotX4,
rotYAll, initRotZ4 -
(jointInDegrees[3]-initRotZ4));
j5Rot = Quaternion.Euler (rotX5,
rotYAll, initRotZ5 -
(jointInDegrees[4]-initRotZ5));
j6Rot = Quaternion.Euler (rotX6,
rotYAll, initRotZ6 -
(jointInDegrees[5]-initRotZ6));
jointSolutionQua = new
Quaternion[6]{j1Rot,j2Rot,j3Rot,j4Rot,
j5Rot,j6Rot};
return jointSolutionQua;
}

public bool Q1InsideLimits( float q1)
{
float minValue = -160f;
float maxValue = 160f;
if ((q1 >= minValue) && (q1 <=
maxValue)) {
return true;
} else {
return false;
}
}

public bool Q2InsideLimits( float q2)
{
float minValue = -227.5f;
float maxValue = 47.5f;
if ((q2 >= minValue) && (q2 <=
maxValue)) {
return true;
} else {
return false;
}
}

public bool Q3InsideLimits( float q3)
{
float minValue = -52.5f;
float maxValue = 232.5f;
if ((q3 >= minValue) && (q3 <=
maxValue)) {
return true;
} else {
return false;
}
}

public bool Q4InsideLimits( float q4)
{
float minValue = -270f;
float maxValue = 270f;
if ((q4 >= minValue) && (q4 <=
maxValue)) {
return true;
} else {
return false;
}
}

public bool Q5InsideLimits( float q5)
{
float minValue = -105f;
float maxValue = 120f;
if ((q5 >= minValue) && (q5 <=
maxValue)) {
return true;
} else {
return false;
}
}

public bool Q6InsideLimits( float q6)
{
float minValue = -270f;
float maxValue = 270f;
if ((q6 >= minValue) && (q6 <=
maxValue)) {
return true;
} else {
return false;
}
}

// Forward kinematic controller
public List<Vector4>
FKcontroller(float[] jointValues, int
desiredJointPos = 0){

float q1, q2, q3, q4, q5, q6;
List<Vector4> FKsolution = new
List<Vector4> ();
float alpha = 0.45f;
float d = 0.65f;
float d2 = 0.085f;
q1 = jointValues[0] * Mathf.Deg2Rad;
q2 = jointValues[1] * Mathf.Deg2Rad;
q3 = jointValues[2] * Mathf.Deg2Rad;
q4 = jointValues[3] * Mathf.Deg2Rad;
q5 = jointValues[4] * Mathf.Deg2Rad;
q6 = jointValues[5] * Mathf.Deg2Rad;

Matrix4x4 T01 = Matrix4x4.identity;
Matrix4x4 T12 = Matrix4x4.identity;
Matrix4x4 T23 = Matrix4x4.identity;
Matrix4x4 T34 = Matrix4x4.identity;
Matrix4x4 T45 = Matrix4x4.identity;
Matrix4x4 T56 = Matrix4x4.identity;
Matrix4x4 T67 = Matrix4x4.identity;
```

```
//Create Tranformation Matrix form
Joint 0 --> 1
// [c1,-s1,0,0]
// [s1, c1,0,0]
// [ 0,  0,1,0]
// [ 0,  0,0,1]
T01.m00 = Mathf.Cos(q1);
T01.m01 = -Mathf.Sin(q1);
T01.m10 = Mathf.Sin(q1);
T01.m11 = Mathf.Cos(q1);
//T01.m00 = Mathf.Cos(q2-
90f*Mathf.Deg2Rad) ;
//T01.m01 = -Mathf.Sin (q2-
90f*Mathf.Deg2Rad) ;
//T01.m11 = 0f;
//T01.m12 = 1f;
//T01.m20 = -Mathf.Sin (q2-
90f*Mathf.Deg2Rad) ;
//T01.m21 = -Mathf.Cos (q2-
90f*Mathf.Deg2Rad) ;
//T01.m22 = 0f;


//Create Tranformation Matrix form
Joint 1 --> 2
// [c2,-s2,0,0]
// [ 0,  0,1,0]
// [-s2, -c2,0,0]
// [ 0,  0,0,1]
T12.m00 = Mathf.Cos(q2);
T12.m01 = -Mathf.Sin (q2);
T12.m11 = 0f;
T12.m12 = 1f;
T12.m20 = -Mathf.Sin (q2);
T12.m21 = -Mathf.Cos (q2);
T12.m22 = 0f;

//T12.m00 = Mathf.Cos(q2) ;
//T12.m02 = Mathf.Sin (q2) ;
//T12.m20 = -Mathf.Sin (q2) ;
//T12.m22 =  Mathf.Cos(q2) ;

//Create Tranformation Matrix form
Joint 2 --> 3
// [c3,-s3,0,alpha]
// [s3, c3,0,0]
// [ 0,  0,1,0]
// [ 0,  0,0,1]
T23.m00 = Mathf.Cos(q3);
T23.m01 = -Mathf.Sin(q3);
T23.m03 = alpha;
T23.m10 = Mathf.Sin(q3);
T23.m11 = Mathf.Cos(q3);
//T23.m00 = Mathf.Cos(q3);
//T23.m01 = -Mathf.Sin(q3);
//T23.m11 = 0f;
//T23.m12 = 1f;
//T23.m20 = -Mathf.Sin(q3);
//T23.m21 = -Mathf.Cos(q3);
//T23.m22 = 0f;
//T23.m03 = alpha;

//Create Tranformation Matrix form
Joint 3 --> 4
// [c4,-s4,0,0]
// [ 0,  0,-1,-d]
// [s4,c4, 0,0]
// [ 0,  0,0,1]
T34.m00 = Mathf.Cos(q4);
T34.m01 = -Mathf.Sin (q4);
T34.m11 = 0f;
T34.m12 = -1f;
T34.m13 = -d;
T34.m20 = Mathf.Sin (q4);
T34.m21 = Mathf.Cos (q4);
T34.m22 = 0f;

//Create Tranformation Matrix form
Joint 4 --> 5
// [c5,-s5,0,0]
// [ 0,  0,1,0]
// [-s5,-c5,0,0]
// [ 0,  0,0,1]
T45.m00 = Mathf.Cos(q5);
T45.m01 = -Mathf.Sin (q5);
T45.m11 = 0f;
T45.m12 = 1f;
T45.m20 = -Mathf.Sin (q5);
T45.m21 = -Mathf.Cos (q5);
T45.m22 = 0f;

//Create Tranformation Matrix form
Joint 5 --> 6
// [c6,-s6,0,0]
// [ 0,  0,-1,0]
// [s6, c6,0,0]
// [ 0,  0,0,1]
T56.m00 = Mathf.Cos(q6);
T56.m01 = -Mathf.Sin (q6);
T56.m11 = 0f;
T56.m12 = -1f;
T56.m20 = Mathf.Sin (q6);
T56.m21 = Mathf.Cos (q6);
T56.m22 = 0f;

//Create Tranformation Matrix form
Joint 6 --> 7 ( Wrist to Tool Flange)
// [1,0,0,0]
// [0,0,-1,0]
// [0,1,0,d2]
// [0,0,0,1]
T67.m11 = 0f;
T67.m12 = -1f;
T67.m23 = d2;
T67.m21 = 1f;
T67.m22 = 0f;

Matrix4x4 T02 = T01 * T12;
Matrix4x4 T03 = T02 * T23;
Matrix4x4 T04 = T03 * T34;
Matrix4x4 T05 = T04 * T45;
Matrix4x4 T06 = T05 * T56;
Matrix4x4 T07 = T06 * T67;

switch (desiredJointPos) {
//by default returns tool flange
position
case 0:
FKsolution.Add (T07.GetColumn (3));
//Get the translation matrix
FKsolution.Add (T07.GetColumn (2));
//Get the Rz matrix
break;
case 1:
FKsolution.Add (T01.GetColumn (3));
FKsolution.Add (T01.GetColumn (2));
```

```
break;
case 2:
FKsolution.Add (T02.GetColumn (3));
FKsolution.Add (T02.GetColumn (2));
break;
case 3:
FKsolution.Add (T03.GetColumn (3));
FKsolution.Add (T03.GetColumn (2));
break;
case 4:
FKsolution.Add (T04.GetColumn (3));
FKsolution.Add (T04.GetColumn (2));
break;
case 5:
FKsolution.Add (T05.GetColumn (3));
FKsolution.Add (T05.GetColumn (2));
break;
case 6:
FKsolution.Add (T07.GetColumn (3));
FKsolution.Add (T07.GetColumn (2));
break;
case 7:
break;
default:
Debug.Log (" Joint " + desiredJointPos
+ " does not exist.");
break;
}
return FKsolution;
}
}
```

RRT Controller.cs

```
using UnityEngine;
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using Random= UnityEngine.Random;
using System.Linq;

public class RRTController
:MonoBehaviour {

private SphereCollider WorkSphere;
private SphereCollider CursorSphere;
static int totalIterations;
public Param RRTCreationParameters;
private Vector3 RandomPos;
private obstacle Obstacles; //Instance
for tool Sphere collider


public Vector3 GoalPoint;
private GameObject OKNode;
private GameObject NOKNode;
public bool enableDebugVisuals;
List<GameObject> instantiatedObjs =
new List<GameObject>();

// Use this for initialization
void Start () {
OKNode = GameObject.Find ("OKNode");
NOKNode = GameObject.Find ("NOKNode");
WorkSphere = GameObject.Find
("Object_108").GetComponent<SphereColl
ider> ();
CursorSphere = GameObject.Find
("SpherePerUser").GetComponent<SphereC
ollider> ();
}

public struct UNode
{
public Vector3 Point;
public int PrevPoint;

public UNode(Vector3 p1, int p2){
this.Point = p1;
this.PrevPoint = p2;
}
}

public class Rrt
{
public List<UNode> MyNodes;
public List<UNode> MyPath;

public Rrt() {
MyNodes= new List<UNode>();
MyPath= new List<UNode>();
}

public void AddTreeNode (UNode n) {
MyNodes.Add(n);
}

public void AddPathNode (UNode n) {
MyPath.Add(n);
}

}

private class obstacle {
private SphereCollider CursorObst;

private obstacle(){
this.CursorObst=GameObject.Find
("SpherePerUser").GetComponent<SphereC
ollider> ();
}
}

[System.Serializable]
public struct Param{
public int maxiter;
public float res;
public float thresh;
public int smoothiter;
}

//Generate random points inside Shpere
public  Vector3
SphereVector3Randomizer(Vector3 C,
float R){
Vector3 SphCenter = C;
float SphRadius = R ;
float thi = Random.Range (0f,
(float)Math.PI*2);
float t = Random.value;
float phi = Mathf.Acos (2 * t - 1);
//transform spherical coordinates in
(x, y, z)
float r = SphRadius * Mathf.Pow
(Random.value, 1f / 3f);
float x = SphCenter.x + (r * Mathf.Cos
(thi) * Mathf.Sin (phi));
float y = SphCenter.y + (r * Mathf.Sin
(thi) * Mathf.Sin (phi));
float z = SphCenter.z + (r * Mathf.Cos
(phi));
Vector3 point = new Vector3(x, y, z);
return point;
}

//Remove points not in the working
space of the robot
public Vector3
WorkingSpaceFilter(Vector3 point,
Vector3 C,float R){
Vector3 SphCenter = C;
float SphRadius = R;
Vector3 PointICenterReference = point
- SphCenter;
float PointFromCenterDistance =
Mathf.Sqrt (Mathf.Pow
(PointICenterReference.x, 2f) +
Mathf.Pow (PointICenterReference.y,
2f) + Mathf.Pow
(PointICenterReference.z, 2f));
Vector3 projectedpoint = SphRadius /
PointFromCenterDistance *
PointICenterReference + SphCenter;
return projectedpoint;
}
```

```
public List<Vector3>
FindPath(SphereCollider WorkSphere,
SphereCollider Obstacles, Param pars,
Vector3 p_start, Vector3 p_goal){
Vector3 RandomPos;
GameObject NOKclone;
GameObject OKclone;
Vector3 ProjectedPoint;
List<UNode> finalPath;
List<Vector3> finalSmoothedPath;
float dist;
Rrt rrt = new Rrt ();
UNode StartNode = new UNode (p_start,
-1);
rrt.AddTreeNode (StartNode);
//Destroy all instantiated objects
that are used for visual debugging in
order to run a new RRT instance
if (enableDebugVisuals) {
//Destroy Game Objects
if (instantiatedObjs.Count>0){
foreach (GameObject i in
instantiatedObjs) {
GameObject.Destroy (i);
}
}
//Clear List
instantiatedObjs=new
List<GameObject>();
}
int iterations = 1;
while (iterations <= pars.maxiter) {
//Generate random Vector3 inside the
sphere volume - working space
RandomPos = SphereVector3Randomizer
(WorkSphere.bounds.center,
WorkSphere.radius *
WorkSphere.transform.parent.localScale
.x);
ProjectedPoint = WorkingSpaceFilter
(RandomPos,
WorkSphere.bounds.center,WorkSphere.ra
dius *
WorkSphere.transform.parent.localScale
.x);

if (Physics.CheckSphere (RandomPos,
pars.res, 9)) {
if (enableDebugVisuals) {
Debug.Log ("TempSphere collision
detected - Creating NOK node");
NOKclone =
(GameObject)GameObject.Instantiate
((GameObject)NOKNode, RandomPos,
Quaternion.identity);
instantiatedObjs.Add (NOKclone);
}
iterations++;
if (iterations > pars.maxiter) {
Debug.Log ("No solution Found!!");
}
continue;
}
//check out of working space
if (!Physics.CheckSphere
(ProjectedPoint, pars.res, 9)) {
if (Physics.Linecast (RandomPos,
ProjectedPoint, 9)) {
if (enableDebugVisuals) {
OKclone =
(GameObject)GameObject.Instantiate
((GameObject)OKNode, RandomPos,
Quaternion.identity);
instantiatedObjs.Add (OKclone);
}
}
else {
if (enableDebugVisuals) {
Debug.Log ("TempSphere out of Working
Space");
NOKclone =
(GameObject)GameObject.Instantiate
((GameObject)NOKNode, RandomPos,
Quaternion.identity);
NOKclone.GetComponent<MeshRenderer>
().material.color = Color.black;
instantiatedObjs.Add (NOKclone);
NOKclone =
(GameObject)GameObject.Instantiate
((GameObject)NOKNode, ProjectedPoint,
Quaternion.identity);
NOKclone.GetComponent<MeshRenderer>
().material.color = Color.blue;
instantiatedObjs.Add (NOKclone);
LineRenderer NOKline = new GameObject
("Line").AddComponent<LineRenderer>
();
NOKline.SetWidth (0.01f, 0.01f);
NOKline.material.color = Color.white;
NOKline.SetColors (Color.white,
Color.white);
NOKline.SetPosition (0, RandomPos);
NOKline.SetPosition (1,
ProjectedPoint);
instantiatedObjs.Add
(NOKline.gameObject);
}
iterations++;
continue;
}
}
if (enableDebugVisuals) {
OKclone =
(GameObject)GameObject.Instantiate
((GameObject)OKNode, RandomPos,
Quaternion.identity);
instantiatedObjs.Add (OKclone);
}

int imin = 0;
int i;
float mindist = Mathf.Infinity;
for (i = 0; i < rrt.MyNodes.Count;
i++) {
dist = Vector3.Distance (rrt.MyNodes
[i].Point, RandomPos);
if ((i == 0) || (dist < mindist)) {
mindist = dist;
imin = i;
}
}

// check line between the current
random pos and closest parent node
```

```
if (Physics.Linecast (rrt.MyNodes
[imin].Point, RandomPos/*, out hit*/,
9)) {
if (enableDebugVisuals) {
LineRenderer NOKline = new GameObject
("Line").AddComponent<LineRenderer>
();
NOKline.SetWidth (0.01f, 0.01f);
NOKline.material.color = Color.red;
NOKline.SetColors (Color.red,
Color.red);
NOKline.SetPosition (0, rrt.MyNodes
[imin].Point);
NOKline.SetPosition (1, RandomPos);
instantiatedObjs.Add
(NOKline.gameObject);
}
iterations++;
continue;
}
if (enableDebugVisuals) {
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.01f, 0.01f);
OKline.material.color = Color.green;
OKline.SetColors (Color.green,
Color.green);
OKline.SetPosition (0, rrt.MyNodes
[imin].Point);
OKline.SetPosition (1, RandomPos);
instantiatedObjs.Add
(OKline.gameObject);
}
UNode RandomNode = new UNode
(RandomPos, imin);
rrt.AddTreeNode (RandomNode);
dist = Vector3.Distance (RandomPos,
p_goal);
if (dist < pars.thresh) {
if (Physics.CheckSphere (p_goal,
pars.res, 9)) {
if (enableDebugVisuals) {
NOKclone =
(GameObject)GameObject.Instantiate
((GameObject)NOKNode, p_goal,
Quaternion.identity);
instantiatedObjs.Add (NOKclone);
}
Debug.Log ("Obstacle in Goal Position
- No solution Possible!....Stopping
RRT controller...");
iterations = pars.maxiter + 1;
break;
}
if (Physics.Linecast (p_goal,
RandomPos, 9)) { // check final point
if (enableDebugVisuals) {
LineRenderer NOKline = new GameObject
("Line").AddComponent<LineRenderer>
();
NOKline.SetWidth (0.01f, 0.01f);
NOKline.material.color = Color.red;
NOKline.SetColors (Color.red,
Color.red);
NOKline.SetPosition (0, RandomPos);
NOKline.SetPosition (1, p_goal);
instantiatedObjs.Add
(NOKline.gameObject);
}
} else {
if (enableDebugVisuals) {
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.01f, 0.01f);
OKline.material.color = Color.green;
OKline.SetColors (Color.green,
Color.green);
OKline.SetPosition (0, RandomPos);
OKline.SetPosition (1, p_goal);
instantiatedObjs.Add
(OKline.gameObject);
}

UNode GoalNode = new UNode (p_goal,
(rrt.MyNodes.Count-1));
rrt.AddTreeNode (GoalNode);
if (enableDebugVisuals) {
OKclone =
(GameObject)GameObject.Instantiate
((GameObject)OKNode, p_goal,
Quaternion.identity);
OKclone.GetComponent<MeshRenderer>
().material.color = Color.blue;
instantiatedObjs.Add (OKclone);
}
Debug.Log ("Target Found!");
iterations = pars.maxiter + 1;
break;
}
}
totalIterations = iterations;
iterations++;
} //while End

//Contstruct Path
int k = rrt.MyNodes.Count-1; //List
index is 0-based so k represent the
number of node with 1st node (end
effector position) being node # 0
rrt.AddPathNode(rrt.MyNodes [k]);
//Add Goal Node
while (true){
k=rrt.MyNodes [k].PrevPoint;
if (k==0)
{
rrt.AddPathNode(rrt.MyNodes [k]);
//Add Starting Node
finalPath = rrt.MyPath;
finalPath.Reverse ();
break;
}
rrt.AddPathNode(rrt.MyNodes [k]);
//Add Nodes in between
}

//Highlight Path
if (enableDebugVisuals) {
for (int j = 0; j < (rrt.MyPath.Count
- 1); j++) {
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.04f, 0.04f);
```

```
OKline.material.color = Color.blue;
OKline.SetColors (Color.blue,
Color.blue);
OKline.SetPosition (0, rrt.MyPath
[j].Point);
OKline.SetPosition (1, rrt.MyPath [j +
1].Point);
instantiatedObjs.Add
(OKline.gameObject);
}
 }
finalSmoothedPath = PathSmoother
(finalPath,pars.smoothiter);
return finalSmoothedPath;
}

public List<Vector3>
PathSmoother(List<UNode>
UnsmoothedPath, int smoothIterations){
List<Vector3> SmoothedPath = new
List<Vector3>();
List<Vector3> MyPath = new
List<Vector3>();

for (int k=0; k <
UnsmoothedPath.Count; k++) {
MyPath.Add(UnsmoothedPath[k].Point);
}
int SizeoflengthArray = MyPath.Count;
//Number of lengths is Path.count
(#Nodes) with L[0]=0
float[] PathLengths = new
float[SizeoflengthArray];
Array.Clear (PathLengths, 0,
SizeoflengthArray); //Zero out the
array
for (int k = 1; k < SizeoflengthArray;
k++) {
PathLengths[k]=
Vector3.Distance(MyPath[k-1],
MyPath[k]) + PathLengths[k-1];
}
float InitialLength =
PathLengths.Last();
int iter = 1;
while (iter <= smoothIterations){
float reducer1 =
Random.Range(0.0f,1.0f)*
PathLengths.Last();
float reducer2 =
Random.Range(0.0f,1.0f)*
PathLengths.Last();
if (reducer2 < reducer1){
float tempr = reducer1;
reducer1 = reducer2;
reducer2 = tempr;
}
int k=0;
int i=0;
int j=0;
for  (k = 1; k < SizeoflengthArray;
k++){
if (reducer1 < PathLengths[k]){
i = k - 1;
break;
}
}
for  (k = (i + 1); k <
SizeoflengthArray; k++){
if (reducer2 < PathLengths[k]){
j = k - 1;
break;
}
}
if (j <= i){
iter++;
continue;
}
float t1 = ((reducer1 -
PathLengths[i]) / (PathLengths[i+1] -
PathLengths[i]));
Vector3 gamma1 = (1 - t1) * MyPath[i]
+ t1 * MyPath[i+1];
float t2 = ((reducer2 -
PathLengths[j]) / (PathLengths[j+1] -
PathLengths[j]));
Vector3 gamma2 = (1 - t2) * MyPath[j]
+ t2 * MyPath[j+1];
//check for Collision
//Using spherecast to allow a small
distance (radius) between obstacle and
robot
RaycastHit hit;
if (Physics.SphereCast (gamma1,0.06f,
(gamma2 - gamma1).normalized,out
hit,Vector3.Distance(gamma1,gamma2)
,9)) {
if (enableDebugVisuals) {
LineRenderer NOKline = new GameObject
("Line").AddComponent<LineRenderer>
();
NOKline.SetWidth (0.01f, 0.01f);
NOKline.material.color = Color.red;
NOKline.SetColors (Color.red,
Color.red);
NOKline.SetPosition (0, gamma1);
NOKline.SetPosition (1, gamma2);
instantiatedObjs.Add
(NOKline.gameObject);
}
iter++;
continue;
}

//Clean the previous Smoothed table
SmoothedPath.Clear();
//Create new Smooth Path table
for (k = 0; k <= i; k++) {
SmoothedPath.Add (MyPath [k]);
}

SmoothedPath.Add (gamma1);
SmoothedPath.Add (gamma2);

for (k = j+1; k < SizeoflengthArray;
k++) {
SmoothedPath.Add (MyPath [k]);
}

MyPath = SmoothedPath.ToList(); //Need
copy List not reference it
//recalculate needed variables for the
iterations
```

```
SizeoflengthArray = MyPath.Count;
//Number of lengths is Path.count
(#Nodes) with L[0]=0
Array.Resize(ref
PathLengths,SizeoflengthArray);
Array.Clear (PathLengths, 0,
SizeoflengthArray); //Zero out the
array
for (k = 1; k < SizeoflengthArray;
k++) {
PathLengths[k]=
Vector3.Distance(MyPath[k-1],
MyPath[k]) + PathLengths[k-1];
}
iter++;
}

//Highlight Final Smoothed Path
if (enableDebugVisuals) {
for (int d = 0; d <
(SmoothedPath.Count - 1); d++) {
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.04f, 0.04f);
OKline.material.color = Color.yellow;
OKline.SetColors (Color.yellow,
Color.yellow);
OKline.SetPosition (0, SmoothedPath
[d]);
OKline.SetPosition (1, SmoothedPath [d
+ 1]);
instantiatedObjs.Add
(OKline.gameObject);
}
}
return SmoothedPath;
}

//To call from RobotMover defining
only the goal point
public List<Vector3> Pathplan( Vector3
Start, Vector3 Goal){
return FindPath (WorkSphere,
CursorSphere, RRTCreationParameters,
Start,Goal);
}
}
```

TrajectortPlanner.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

public class LinearTrajectoryPlanner
{

//Staubli RX90 offline instance
public RobotController robot = new
RobotController();
public bool activateTrajectPlan =
true;
public List<PathPoint>
interpolatedPoints;

public class TrajectorySegment
{
public List<PathPoint>
candidateSolutions;
public PathPoint winnerSolution;
public TrajectorySegment (){
this.candidateSolutions = new
List<PathPoint>();
this.winnerSolution = new PathPoint();
}
}

public class TrajectoryPath
{
public List<TrajectorySegment>
SegmentID;
public TrajectoryPath(){
this.SegmentID = new
List<TrajectorySegment> (0);
}
}

//Create the trajectory points using
the Bounded Deviation method
public List<PathPoint>
BoundedDeviationTrajectory(Vector3
startPoint, Vector3 endPoint){

float tolerance = 0.005f;
interpolatedPoints = new
List<PathPoint>();
//Add first Point
interpolatedPoints.Add (new PathPoint
());

interpolatedPoints[0].cartesian.transl
ationVector = startPoint;
interpolatedPoints[0].joint =new
JointConfig( robot.IKcontroller
(startPoint, new Vector3
(0f,0f,0f))[0]);

//Add last point
interpolatedPoints.Add (new PathPoint
());

interpolatedPoints[1].cartesian.transl
ationVector = endPoint;
interpolatedPoints[1].joint = new
JointConfig( robot.IKcontroller
(endPoint, new Vector3
(0f,0f,0f))[0]);

bool isLine = false;
while (isLine == false) {
for (int i = 1; i <
interpolatedPoints.Count; i++) {
float[] midPointJoint = new float[] {
(interpolatedPoints [i-1].joint.q1 +
interpolatedPoints [i].joint.q1) / 2,
(interpolatedPoints [i-1].joint.q2 +
interpolatedPoints [i].joint.q2) / 2,
(interpolatedPoints [i-1].joint.q3 +
interpolatedPoints [i].joint.q3) / 2,
(interpolatedPoints [i-1].joint.q4 +
interpolatedPoints [i].joint.q4) / 2,
(interpolatedPoints [i-1].joint.q5 +
interpolatedPoints [i].joint.q5) / 2,
(interpolatedPoints [i-1].joint.q6 +
interpolatedPoints [i].joint.q6) / 2
};
Vector3 midPointCartesian =
robot.FKcontroller (midPointJoint)[0];
Vector3 actualMidPointCartesian =
(interpolatedPoints [i-
1].cartesian.translationVector +
interpolatedPoints
[i].cartesian.translationVector) / 2;
if
(!(HelperFunctions.Vector3AlmostEqual
(midPointCartesian,
actualMidPointCartesian, tolerance)))
{
//Add knot point
interpolatedPoints.Insert (i, new
PathPoint ());
interpolatedPoints
[i].cartesian.translationVector =
actualMidPointCartesian;
interpolatedPoints [i].joint = new
JointConfig (robot.IKcontroller
(actualMidPointCartesian, new Vector3
(0f, 0f, 0f))[0]);
isLine = false;
i--; // decrease i to check again the
latest newly created segments
}
else {
isLine = true;
}
}
}

//Draw points
List<Vector3> toDraw = new
List<Vector3> ();
for (int j = 0; j <
interpolatedPoints.Count; j++) {

toDraw.Add(interpolatedPoints[j].carte
sian.translationVector);
}
PathDraw (toDraw);
return interpolatedPoints;
}
```

```
//Remove the Draw...just for temp
debug its here
private List<GameObject>
instantiatedObjs=new List<GameObject>
();
void PathDraw(List<Vector3> Path){

if (instantiatedObjs.Count > 0) {
foreach (GameObject i in
instantiatedObjs) {
GameObject.Destroy (i);
}
instantiatedObjs = new
List<GameObject> ();
}

for (int d = 1; d < (Path.Count - 1);
d++) {
GameObject sphere =
GameObject.CreatePrimitive
(PrimitiveType.Sphere);
sphere.transform.position =
HelperFunctions.robottoUnityCoordinate
SystemChange (Path [d]);
sphere.transform.localScale = new
Vector3 (0.04f, 0.04f, 0.04f);
sphere.GetComponent<Renderer>
().material.color = Color.blue;
sphere.layer=9;
instantiatedObjs.Add (sphere);
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
OKline.SetWidth (0.02f, 0.02f);
OKline.material.color = Color.blue;
OKline.SetColors (Color.blue,
Color.blue);
OKline.SetPosition (0,
HelperFunctions.robottoUnityCoordinate
SystemChange (Path [d]));
OKline.SetPosition (1,
HelperFunctions.robottoUnityCoordinate
SystemChange (Path [d + 1]));
instantiatedObjs.Add
(OKline.gameObject);
}
}

bool inCollision = true;

//Function to Plan the Trajectory
public List<List<Vector3>>
TrajectPlan(RobotPath wPath){

List<TrajectoryPath>
totalPathSolutionsContainer = new
List<TrajectoryPath> ();
RobotPath modPath = wPath; //Reference
the robot Path from RobotMover and
modify it for the Trajectory planning
PathSegment tempSegment = new
PathSegment ();
float lerpStep = 0.2f; //the division
of the straight line segment to check
float tempStep = lerpStep;
Vector3 intermediatePoint;
List<PathPoint> waypoints=new
List<PathPoint>();
//List of all poses of the robot on
the specified trajectory
List<List<Vector3>> robotPoses = new
List<List<Vector3>> ();
//list of the coordinates of all
joints of the robot in a specific
cartesian end effector position
according to the choosen IK solution.
List<Vector3> robotPose = new
List<Vector3> ();
Debug.Log ("Trajectory Planning
activated");
for (int j = 1; j <
wPath.SegmentID.Count; j++) {
tempSegment = wPath.SegmentID[j];
//In case Segment has only 1 point
then create a temporary list and add
the goal point so the trajectPlan can
calculate the interpolated points
waypoints.Clear();
waypoints = new List<PathPoint>
(tempSegment.segmentPoint); // Copying
the list - Not referencing!

if (waypoints.Count == 1) {
waypoints.Add (tempSegment.endPoint);
}

for (int i = 1; i < waypoints.Count;
i++) {
while (tempStep <= 1f) {
intermediatePoint = Vector3.Lerp
(waypoints [i -
1].cartesian.translationVector,
waypoints
[i].cartesian.translationVector,
tempStep);
Vector3 intermediatePointR =
Vector3.Lerp (waypoints [i -
1].cartesian.rotationVector, waypoints
[i].cartesian.rotationVector,
tempStep);
float[] currentJointCoordinates =
robot.IKcontroller (intermediatePoint,
intermediatePointR) [0];
List<float[]>
candidateSolutionsForEachPoint =
robot.IKcontroller (intermediatePoint,
intermediatePointR,"PLAN");
//convert List<Float> to
List<PathPoint>
List<PathPoint> allsolutions = new
List<PathPoint>();
JointConfig tempJoint;
PathPoint tempPoint = new PathPoint();
for (int d = 0; d <
candidateSolutionsForEachPoint.Count;
d++) {
allsolutions.Add (tempPoint);
tempJoint = new JointConfig
(candidateSolutionsForEachPoint [d]);
allsolutions [d].joint = tempJoint;
allsolutions
[d].cartesian.translationVector =
intermediatePoint;
allsolutions
[d].cartesian.rotationVector =
intermediatePointR;
```

```
}

//itterate and remove all solutions
not close to the target point.
int t = 0;
for (int k = 0; k <
candidateSolutionsForEachPoint.Count;
k++) {
float[] tempSolution =
candidateSolutionsForEachPoint [k];

//Check if solution results in the
desired cartesian coordinates and
delete the ones that don't
Vector3 validPosition =
robot.FKcontroller (tempSolution) [0];
if
(HelperFunctions.Vector3AlmostEqual
(intermediatePoint, validPosition,
0.001f) == false) {
allsolutions.RemoveAt (k-t);
t++;
}
}

//Modify Final Path to execute
if (tempStep == 1) { // just update
existing point
modPath.SegmentID
[j].segmentPoint.Last ().joint.qarray
= candidateSolutionsForEachPoint [0];
} else {
PathPoint pointToInsert = new
PathPoint ();

pointToInsert.cartesian.translationVec
tor = intermediatePoint;
pointToInsert.joint.qarray =
candidateSolutionsForEachPoint [0];
if (modPath.SegmentID
[j].segmentPoint.Count == 1) {
modPath.SegmentID
[j].segmentPoint.Add (pointToInsert);
} else {
int positionToInsert =
modPath.SegmentID
[j].segmentPoint.Count - 1;
modPath.SegmentID
[j].segmentPoint.Insert
(positionToInsert, pointToInsert);
}
}

if (allsolutions.Count == 0) {
Debug.Log ("Robot cannot reach
desired position with the specified
joint setup in point: " +
intermediatePoint.x + " / " +
intermediatePoint.y + " / " +
intermediatePoint.z);
}
else  {
//iterate in all solutions for a non-
colliding solution
for (int l = 0; l <
allsolutions.Count; l++) {
//convert to float to use as input in
the FK controller
float[] tempJ =
allsolutions[l].joint.qarray;
robotPose = SimulateRobotPose
(tempJ);
if (RobotPoseObstacleDetector
(robotPose)) {
Debug.Log ("Selecting next Pose
solution to avoid Link collison");
continue;
}
else {
robotPoses.Add (robotPose);
}
}
if (robotPoses.Count == 0) {
Debug.Log ("Trajectory Planner did
not find any free collision
solution");
}
}
tempStep = tempStep + lerpStep;
}
tempStep = lerpStep;
}
}
return robotPoses;
}

//Check if robot can follow the path
public List<List<Vector3>> segm = new
List<List<Vector3>> ();

public bool
TrajectoryObstacleDetector(RobotPath
wholePath){

bool newTrajectoryPath = false;
if ((activateTrajectPlan) ||
(inCollision)) {
if (instantiatedObjs.Count > 0) {
foreach (GameObject i in
instantiatedObjs) {
GameObject.Destroy (i);
}
instantiatedObjs = new
List<GameObject> ();
}
segm = new List<List<Vector3>> ();
segm = TrajectPlan (wholePath);
activateTrajectPlan = false;
newTrajectoryPath = true;
}
inCollision = true;

for (int k = 0; k < segm.Count; k++) {
for (int j = 1; j < segm [k].Count;
j++) {
//ignore robot in the raycast
if (Physics.Linecast
(HelperFunctions.robottoUnityCoordinat
eSystemChange (segm [k] [j - 1]),
HelperFunctions.robottoUnityCoordinate
SystemChange (segm [k] [j]), ~((1 <<
10) | (1 << 9)))) {
if (newTrajectoryPath) {
LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
```

```
 OKline.SetWidth (0.02f, 0.02f);
 OKline.material.color = Color.red;
 OKline.SetColors (Color.red,
Color.red);
 OKline.SetPosition (0,
HelperFunctions.robottoUnityCoordinate
SystemChange (segm [k] [j - 1]));
 OKline.SetPosition (1,
HelperFunctions.robottoUnityCoordinate
SystemChange (segm [k] [j]));
 instantiatedObjs.Add
(OKline.gameObject);
 }
 Debug.Log ("Collision in Robot Link :
" + j);
 inCollision = true;
 } else {
 if (newTrajectoryPath) {
 LineRenderer OKline = new GameObject
("Line").AddComponent<LineRenderer>
();
 OKline.SetWidth (0.02f, 0.02f);
 OKline.material.color = Color.green;
 OKline.SetColors (Color.green,
Color.green);
 OKline.SetPosition (0,
HelperFunctions.robottoUnityCoordinate
SystemChange (segm [k] [j - 1]));
 OKline.SetPosition (1,
HelperFunctions.robottoUnityCoordinate
SystemChange (segm [k] [j]));
 instantiatedObjs.Add
(OKline.gameObject);
 }
 inCollision = false;
 }
 }
 }
newTrajectoryPath = false;
return inCollision;
}

public bool
RobotPoseObstacleDetector(List<Vector3
> robotPose){

bool linkCollision = true;
for (int j = 1; j < robotPose.Count;
j++) {
//ignore robot in the raycast
if (Physics.Linecast
(HelperFunctions.robottoUnityCoordinat
eSystemChange (robotPose [j - 1]),
HelperFunctions.robottoUnityCoordinate
SystemChange (robotPose[j]), ~((1 <<
10) | (1 << 9)))) {
 Debug.Log ("Collision in Robot Link :
" + j);
 linkCollision = true;
 return linkCollision;
 }
 else {
 linkCollision = false;
 }
 }
return linkCollision;
}
```

```
public List<Vector3> SimulateRobotPose
(float[] jointValues){
List<Vector3> jointPos = new
List<Vector3> ();
Vector3 joint1Pos = robot.FKcontroller
(jointValues, 1)[0];
Vector3 joint3Pos = robot.FKcontroller
(jointValues, 3)[0];
Vector3 joint4Pos = robot.FKcontroller
(jointValues, 4)[0];
Vector3 flangePos = robot.FKcontroller
(jointValues)[0];
jointPos.Add (joint1Pos);
jointPos.Add (joint3Pos);
jointPos.Add (joint4Pos);
jointPos.Add (flangePos);
return jointPos;
}

}
```

HelperFunctions.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

//structure for Cartesian points
[System.Serializable]
public struct Cartesian6DConfig
{
public float x;
public float y;
public float z;
public float a;
public float b;
public float c;
public Vector3 translationVector;
public Vector3 rotationVector;
public Cartesian6DConfig(float aX, float aY, float aZ, float aA, float aB, float aC)
{
this.x = aX;
this.y = aY;
this.z = aZ;
this.a = aA;
this.b = aB;
this.c = aC;
this.translationVector = new Vector3 (aX, aY, aZ);
this.rotationVector = new Vector3 (aA, aB, aC);
}
}

//Structure for Joint space configuration
[System.Serializable]
public struct JointConfig
{
public float q1;
public float q2;
public float q3;
public float q4;
public float q5;
public float q6;
public float[] qarray;
public JointConfig(float[] aq)
{
this.q1 = aq[0];
this.q2 = aq[1];
this.q3 = aq[2];
this.q4 = aq[3];
this.q5 = aq[4];
this.q6 = aq[5];
qarray= new float[6];
qarray = aq;
}
}

[System.Serializable]
public class PathPoint
{
public Cartesian6DConfig cartesian;
public JointConfig joint;
}

[System.Serializable]
public class PathSegment
{
public List<PathPoint> segmentPoint;
public PathPoint startPoint;
public PathPoint endPoint;
public PathSegment (){
this.segmentPoint = new List<PathPoint>();
this.startPoint = new PathPoint();
this.endPoint = new PathPoint();
}
}

[System.Serializable]
public class RobotPath
{

public List<PathSegment> SegmentID;
//user inputs a List of the needed Path points and the constructor of the class creates the RobotPath structure that contains
//all needed info even for the intermediate points that will be created from the trajectory planner or RRT path planner
public RobotPath(List<Cartesian6DConfig> userPath){
SegmentID = new List<PathSegment> (0);
for (int i=0; i<userPath.Count;i++){
//Add a Segment (contains many points)
SegmentID.Add (new PathSegment());
//Add a Segment Point
SegmentID [i].segmentPoint.Add (new PathPoint ());
//For each segment, points in index 0 are the user defined points that have to be executed no matter what. Segments start from 1 and each
//segment contains all the points before it. So segment 1 contains points between Starting point 0(included) and point 1(not included), segment 2 between
//points 1 and 2, ...., last Segment between n-1 and n point.
if (i == 0) {
//SegmentID 0 stores only the first Position or the robot current position
SegmentID [i].segmentPoint [0].cartesian = userPath [i];
SegmentID [i].startPoint.cartesian = userPath [i];
SegmentID [i].endPoint.cartesian = userPath [i];
}
else {
//Assign the point value
SegmentID [i].segmentPoint [0].cartesian = userPath [i-1];
//Assign the values for the Segments members
SegmentID [i].startPoint.cartesian = userPath [i-1];
```

```
SegmentID [i].endPoint.cartesian =
userPath [i];
}
}
Debug.Log ("Segments are " +
(this.SegmentID.Count-1));
}
}

public class HelperFunctions{
public static bool
Vector3AlmostEqual(Vector3 a, Vector3
b, float PercentageDifferenceAllowed){
bool isAlmostEqual = false;
if( (a - b).sqrMagnitude <= (a *
PercentageDifferenceAllowed).sqrMagnit
ude ) {
//Debug.Log( "They are almost equal.
Less than " +
PercentageDifferenceAllowed*100+ "%
different" );
isAlmostEqual = true;
}
return isAlmostEqual;
}

public bool FloatAlmostEqual(float a,
float b, float epsilon) {
return (Math.Abs(a-b)< epsilon);
}

public bool QuaternionAlmostEqual
(Quaternion a,Quaternion b){
//the "==" Quaternion operator by
default checks if angle difference is
bellow error (epsilon)
return (a==b);
}
}
```