ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Έλεγχος κινητήρα DC με χρήση ελεγκτή τριών όρων (PID controller)

Παναγιώτης Κατωπόδης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Φούρλας Γεώργιος
Αναπληρωτής Καθηγητής

Λαμία 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Έλεγχος κινητήρα DC με χρήση ελεγκτή τριών όρων (PID controller)

Παναγιώτης Κατωπόδης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Φούρλας Γεώργιος

Αναπληρωτής Καθηγητής

Λαμία 2021

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Controlling of a DC motor with the implementation of a PID controller

Panagiotis Katopodis

FINAL THESIS

ADVISOR

Fourlas Georgios

Associate Professor

Lamia 2021

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../2021

Ο Δηλ.

Παναγιώτης Κατωπόδης

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Το θέμα της παρούσας εργασίας είναι η υλοποίηση ενός μοτέρ στο περιβάλλον CoppeliaSim και την δημιουργία ενός PID ελεγκτή στο περιβάλλον MATLAB για τον έλεγχο του.

Σκοπός της εργασίας είναι η κατανόηση και παραμετροποίηση ενός PID ελεγκτή με σκοπό την ομαλή και προβλέψιμη λειτουργία του μοτέρ που δημιουργήθηκε στο CoppeliaSim ανεξαρτήτως του φορτίου το οποίο θα πρέπει να περιστρέφεται.

Οι ελεγκτές PID είναι ίσως οι πιο διαδεδομένοι τύποι ελεγκτή που χρησιμοποιούνται σε βιομηχανικό και όχι μόνο επίπεδο, όμως η σωστή χρήση τους και η ρύθμισή τους απαιτεί τόσο εμπειρία όσο και βαθιά γνώση του συστήματος το οποίο θα ελέγχουν. Ο ελεγκτής δημιουργήθηκε, στα πλαίσια της παρούσας εργασίας, με τη βοήθεια του λογισμικού MATLAB.

Με τη χρήση του λογισμικού CoppeliaSim μας δίνεται η δυνατότητα να δημιουργήσουμε ένα μοντέλο με τις ιδιότητες που επιθυμούμε και να εκτελέσουμε τις κατάλληλες προσομοιώσεις. Έτσι απαλείφουμε την ανάγκη για προσομοίωση σε πραγματικό περιβάλλον ελαχιστοποιώντας το κόστος και έχοντας την δυνατότητα να προβλέψουμε τυχόν αστοχίες κατά την υλοποίηση του μοντέλου σε πραγματικό περιβάλλον.

ABSTRACT

The subject of this project is the creation of a model motor in CoppeliaSim and the implementation and tuning of a PID controller in MATLAB in order to control it.

The aim of the project is to understand how a PID controller works and being able to configure and calibrate its values in order to work smoothly and predictably when controlling the motor from CoppeliaSim no matter the size of the dummy loads.

Probably the most well-known type of controllers used in the industry but not solely are PID controllers, but to configure such controller correctly involves plenty of knowledge and understanding for the controlling plant and also plenty experience from the user. In this particular project the controller was created using the MATLAB software.

By using the CoppeliaSim software we have the ability to create a model with the specific attributes we want and the ability to execute various simulations. This way we can eliminate the need for real world simulations and testing resulting in more cost efficient ways to deploy the real plant and also being prepared for possible malfunctions and points of failure.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1 Βιβλιογραφική Επισκόπηση

1.1CoppeliaSim

1.1.α. Εισαγωγή CoppeliaSim

Το λογισμικό CoppeliaSim (παλαιότερα γνωστό ως V-Rep) είναι ένα ευρέως διαδεδομένο εργαλείο που χρησιμοποιείται για προσομοιώσεις σε ψηφιακό περιβάλλον. Αρχικά αναπτύχθηκε από την Toshiba και πλέον συντηρείται και αναπτύσσεται από την Coppelia Robotics AG. Η πρώτη του δημόσια κυκλοφορία έγινε στις αρχές του 2010 και έκτοτε χρησιμοποιείτε ευρέος σε έρευνες ρομποτικών συστημάτων καθώς και στον τομέα της εκπαίδευσης. Ο χρήστης μπορεί να εκτελέσει απευθείας scripts γραμμένα σε Lua ή C/C++ καθώς και να χρησιμοποιήσει remote APIs ώστε να εκτελέσει κώδικα γραμμένο σε Python, MATLAB ή ακόμα και Java. Οι σκηνές περιλαμβάνουν μια ιεραρχία από μοντέλα που δημιουργούνται από την συναρμολόγηση αντικειμένων το κάθε ένα με τα δικά του ιδιαίτερα χαρακτηριστικά. Τέλος το CoppeliaSim χρησιμοποιεί μια kinematics engine και physics simulation βιβλιοθήκες για τις προσομοιώσεις του.

1.1.β LUA

Η γλώσσα LUA είναι μια ελαφριά γλώσσα βασισμένη στην γλώσσα C, σχεδιασμένη από μια ομάδα καθηγητών και ερευνητών στο πανεπιστήμιο Pontifical Catholic University στο Ρίο ντε Τζανέιρο. Είναι μια scripting γλώσσα και χρησιμοποιείτε κυρίως σε ενσωματωμένα συστήματα καθώς και σε παιχνίδια όπως το World of Warcraft και Angry Birds. Οι βασικοί λόγοι που για τους οποίους η Lua προτιμάται έναντι άλλων scripting γλωσσών είναι η ταχύτητά της και η ευκολία με την οποία μπορεί να ενσωματωθεί σε κάποιο project. Αξίζει, επίσης, να αναφερθεί ότι μπορεί να τρέξει με τη χρήση ενός απλού C μεταγλωττιστή καθώς και ότι το συνολικό της μέγεθος δεν ξεπερνάει το 1.3 MB.

1.2 MATLAB

1.2.α. Εισαγωγή Matlab

Το λογισμικό MATLAB είναι μια πλατφόρμα σχεδιασμένη για μηχανικούς και ερευνητές δίνοντας τους την δυνατότητα να αναπτύσσουν αλγορίθμους, να δημιουργούν μοντέλα και να αναλύουν δεδομένα. Αρχικά αναπτύχθηκε το 1967 από τον Clever Moler καθηγητή του University of New Mexico με σκοπό να βοηθήσει τους φοιτητές του και είχε την μορφή μιας διαδραστικής αριθμομηχανής. Στις αρχές του 1979 άρχισε η δωρεάν διανομή του σε πανεπιστήμια. Με την ίδρυση της MathWorks, Inc. το 1984 το MATLAB καθιερώθηκε ως εμπορικό προϊόν και έκτοτε έχουν γίνει release πολλές εκδόσεις .

Οι χρήστες έχουν την δυνατότητα να δημιουργήσουν scripts, να χρησιμοποιήσουν ενσωματωμένες αλλά και εξωτερικές βιβλιοθήκες, ανάλογα με τις ανάγκες τους, καθώς και να δημιουργήσουν διαγράμματα και να χρησιμοποιήσουν πληθώρα γραφικών περιβαλλόντων.

Η γλώσσα προγραμματισμού που χρησιμοποιείται είναι proprietary για το λογισμικό και μπορεί να εκτελεστεί απευθείας από το command line ή με μέσο κάποιου text file που περιλαμβάνει τον κώδικα.

1.3 ΕΛΕΓΚΤΕΣ

1.3.α. Εισαγωγή

Ένας ελεγκτής τριών όρων PID (Proportional, Integral, Derivative) έχει ως σκοπό του την διαχείριση της εξόδου ενός συστήματος μέσω ανατροφοδότησης από αισθητήρες και επεξεργασία αυτών των δεδομένων.

Ο ελεγκτής επεξεργάζεται το σφάλμα μεταξύ της επιθυμητής εξόδου και της πραγματικής εξόδου και αναλόγως επεξεργάζεται την είσοδο στο σύστημα ώστε να εξαλείψει τα σφάλματα και να πετύχει την βέλτιστη δυνατή απόκριση .

Εκτός από τους αναλογικούς – ολοκληρωτικούς – διαφορικούς ελεγκτές υπάρχουν και πληθώρα ελεγκτών που χρησιμοποιούν έναν ή δυο από αυτούς τους όρους.

1.3.β. Αναλογικός όρος

Proportional ή αναλογικός ονομάζεται ο όρος που είναι υπεύθυνος για την βελτίωση της συμπεριφοράς του συστήματος και για την διατήρηση της κατά την μεταβατική διάρκεια καθώς και κατά την σταθερή κατάστασή του. Ο αναλογικός όρος δεν έχει την δυνατότητα να εξαλείψει το μόνιμο σφάλμα ούτε να σταθεροποιήσει το σύστημα μετά από κάποια αλλαγή που δεν είχε προβλεφθεί. Για αυτόν τον λόγο σπάνια συναντάται αναλογικός ελεγκτής ενός όρου και συνήθως συνδυάζεται με κάποιον από τους άλλους όρους.

1.3.γ. Ολοκληρωτικός όρος

Integral ή ολοκληρωτικός ονομάζεται ο όρος που είναι υπεύθυνος για την εξάλειψη των σφαλμάτων που προκύπτουν στην σταθερή κατάσταση. Όσο το σφάλμα δεν εξαλείφεται, λόγω της άμεσης εξάρτησης του όρου από τον χρόνο η τιμή του συνεχίζει και αυξάνεται εις βάρος της σταθερότητας και της απόκλισης του συστήματος. Συνήθως συναντάται σε ελεγκτές τύπου PI και PID.

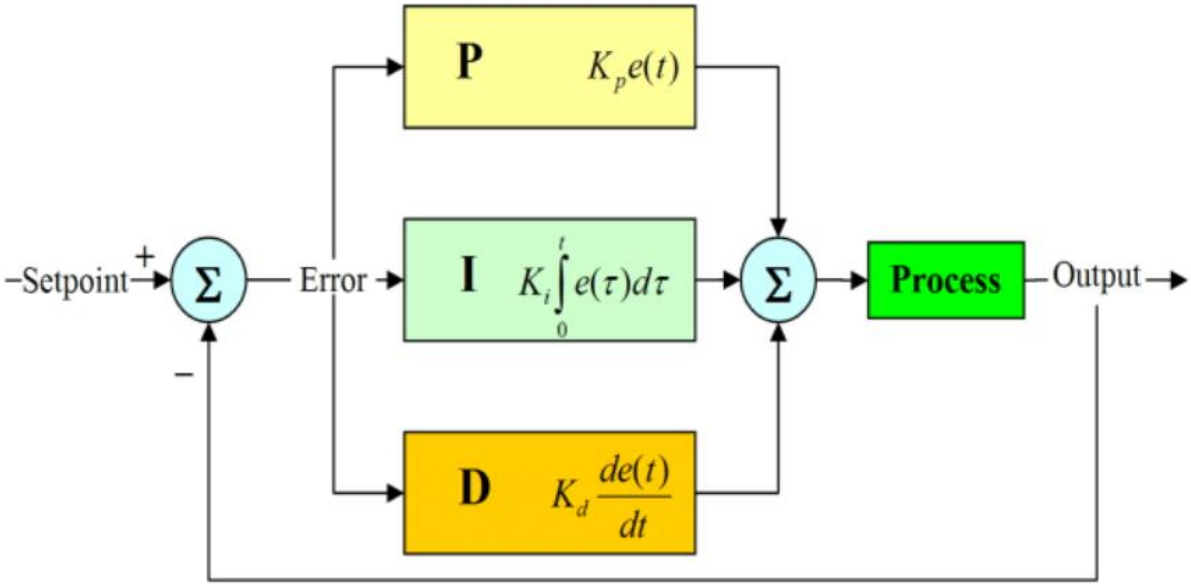
1.3.δ. Διαφορικός όρος

Ο Derivative ή διαφορικός όρος βοηθάει στην απόκριση και την ευστάθεια του συστήματος κάνοντας απόσβεση των αυξομειώσεων που μπορεί να παρατηρούνται και βελτιώνοντας τη συνολική απόδοση του συστήματος. Ο διαφορικός όρος δεν μπορεί ποτέ να υπάρξει μόνος του άρα και συναντάται μόνο σε ελεγκτές δύο και τριών όρων.

1.3.ε. Αναλογικός– ολοκληρωτικός – διαφορικός ελεγκτής

Ένας αναλογικός– ολοκληρωτικός – διαφορικός ελεγκτής συνδυάζει την χρήση και των τριών επιμέρους όρων με σκοπό την κάλυψη όποιων κενών δημιουργούνται από τη χρήση ενός και δυο όρων .

Αρχικά υπολογίζεται το σφάλμα και στη συνεχεία γίνονται οι υπολογισμοί για κάθε έναν όρο ξεχωριστά, στη συνέχεια αθροίζονται τα αποτελέσματα από τους τρεις όρους και δίνονται σαν έξοδος από τον ελεγκτή για να χρησιμοποιηθούν ως είσοδος από το σύστημα.



Εικόνα 1: Block diagram PID controller

1.4 DC ΜΟΤΕΡ

1.4.α. Εισαγωγή

Τα DC μοτέρ εμφανίστηκαν στις αρχές του δέκατου ένατου αιώνα με δημιουργό τους τον επιστήμονα William Sturgeon που δημιούργησε το πρώτο λειτουργικό DC μοτέρ το 1832. Το 1834 ο Ρώσσος μηχανικός Moritz von Jacobi δημιούργησε το πρώτο περιστρεφόμενο μοτέρ δίνοντας έναυσμα σε άλλους μηχανικούς να δημιουργήσουν το πρώτο ηλεκτροκίνητο σκάφος για την μεταφορά επιβατών σε ποτάμια.

Όταν ηλεκτρικό ρεύμα διαρρέει ένα πηνίο δημιούργει μαγνητικό πεδίο. Αλλάζοντας την κατεύθυνση του ρέματος αλλάζει ταυτόχρονα και το μαγνητικό πεδίο. Τα πιο συνήθη DC μοτέρ έχουν έναν ή περισσότερους σταθερούς μαγνήτες στη βάση τους και έναν ή περισσότερους ηλεκτρομαγνήτες στον περιστρεφόμενο άξονα τους. Μεταβάλλοντας την κατεύθυνση του ρεύματος που διαρρέει τους ηλεκτρομαγνήτες επιτυγχάνεται η διαφορά στην πολικότητα μεταξύ του σταθερού μαγνήτη και του ηλεκτρομαγνήτη και έτσι παράγεται κίνηση .

1.4.β. BRUSHED

Ένα brushed DC μοτέρ έχει σταθερούς μαγνήτες στο περίβλημα του και ηλεκτρομαγνήτες στον περιστρεφόμενο άξονα. Τα πλεονεκτήματα του είναι το μικρό αρχικό κόστος κατασκευής, η αξιοπιστία του καθώς και ο εύκολος χειρισμός του. Χρειάζεται όμως συχνή συντήρηση λόγω τον φθειρόμενων μερών του που πρέπει να αντικαθίστανται συχνά για την σωστή λειτουργία του.

1.4.γ. BRUSHLESS

Ένα brushless DC μοτέρ έχει σταθερούς μαγνήτες στον περιστρεφόμενο άξονα και ηλεκτρομαγνήτες στο περίβλημα του. Έτσι δε χρειάζεται να γίνει μεταφορά του ρεύματος μέσω αναλώσιμόν μερών στον άξονα περιστροφής αλλά το ρεύμα διαρρέει απευθείας τους ηλεκτρομαγνήτες στο περίβλημά του. Τα βασικά πλεονεκτήματα ενός τέτοιου μοτέρ είναι η ακρίβεια χειρισμού του και η ευκολία ενσωμάτωσης αισθητήρων, η αξιόπιστη λειτουργία του με ελάχιστη ή και καθόλου συντήρηση και η υψηλή του απόδοση.

1.4.δ. SERVOMOTOR

Τα μοτέρ τύπου servo περιέχουν εκτός από το DC μοτέρ και έναν αισθητήρα θέσης καθώς και έναν ελεγκτή. Έτσι υπάρχει καλύτερος χειρισμός όσον αφορά την γωνιακή ακρίβεια δίνοντας την δυνατότητα να περιστρέφεται ακριβώς όσο χρειάζεται μέχρι να λάβει το επόμενο σήμα.

ΚΕΦΑΛΑΙΟ 2 ΥΛΟΠΟΙΗΣΗ

2.1 COPPELIASIM

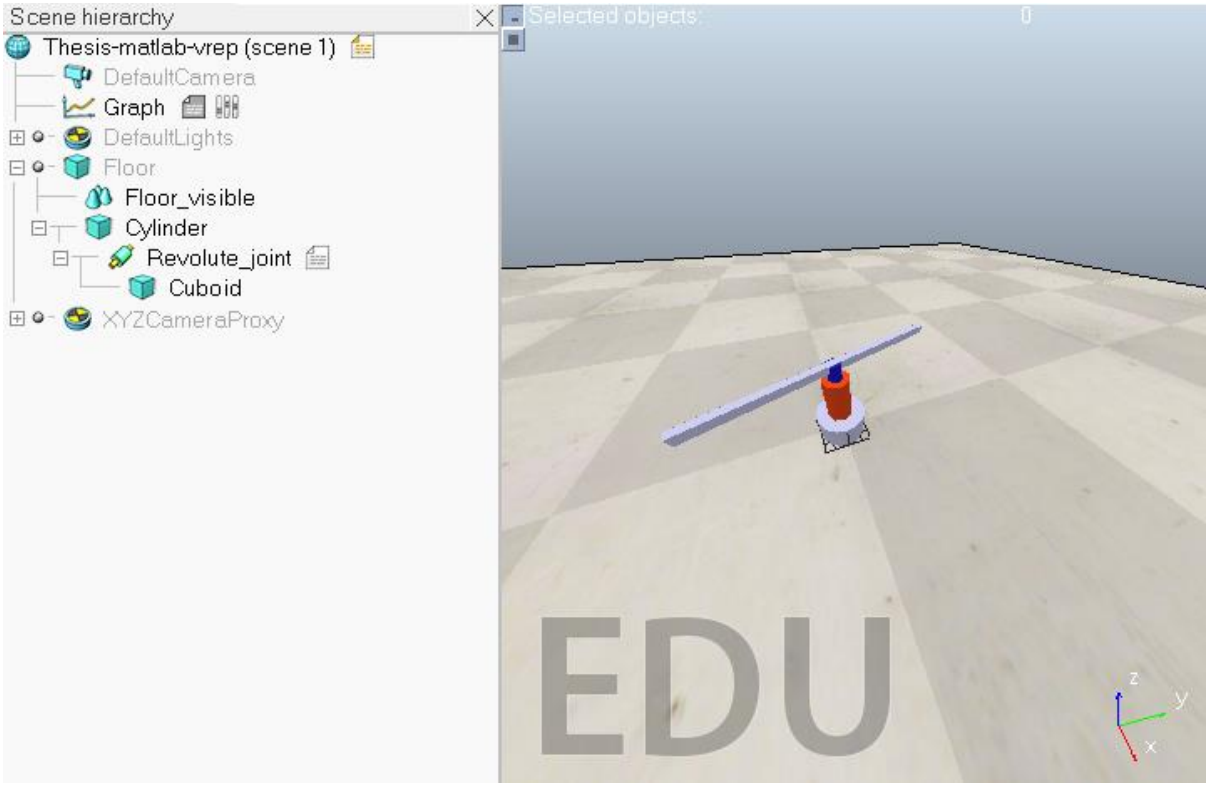
2.1.α. Σχεδίαση μοντέλου στο CoppeliaSim

Σαν βάση για το μοντέλο έχει χρησιμοποιηθεί ένα μη δυναμικό αντικείμενο τύπου Cylinder.

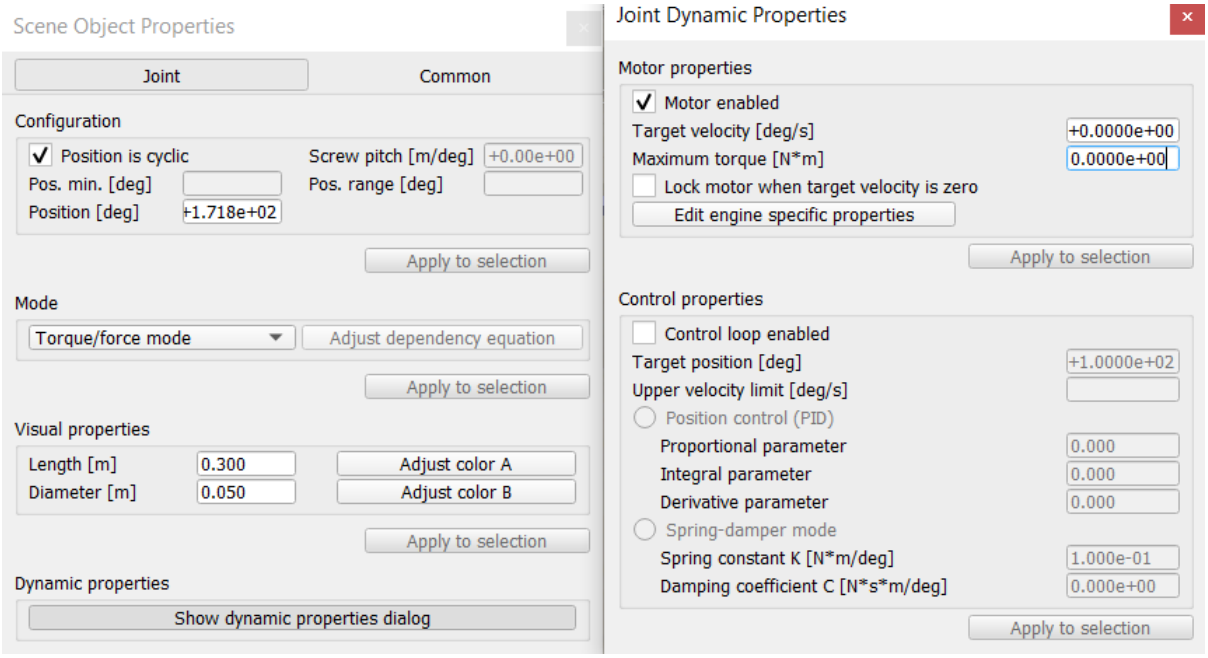
Για να προσομοιώσουμε έναν κινητήρα στο CoppeliaSim αρκεί να χρησιμοποιήσουμε ένα αντικείμενο joint τύπου revolute. Μετά την τοποθέτηση του αντικειμένου στη σκήνη θα πρέπει να ορίσουμε τις ιδιότητες του ώστε να μπορεί να λειτουργήσει ως κινητήρας. Κάνοντας διπλό click πάνω στο αντικείμενο μέσα από την ιεραρχία σκηνής πρέπει στο παράθυρο Scene Object Properties να επιλέξουμε Mode: Torque/force mode. Επίσης υπάρχει η δυνατότητα αλλαγής χρώματος ή του μεγέθους στο οποίο εμφανίζεται το αντικείμενο στη σκηνή μέσω του πεδίου visual properties.

Μέσω της επιλογής Show dynamic properties dialog και μετέπειτα το παράθυρο διαλόγου Joint Dynamic Properties μπορεί να ενεργοποιηθεί η επιλογή Motor enable. Οι τιμές για Target velocity και Maximum torque πρέπει να είναι 0 και θα αλλάξουν κατά την προσομοίωση μέσω του ελεγκτή.

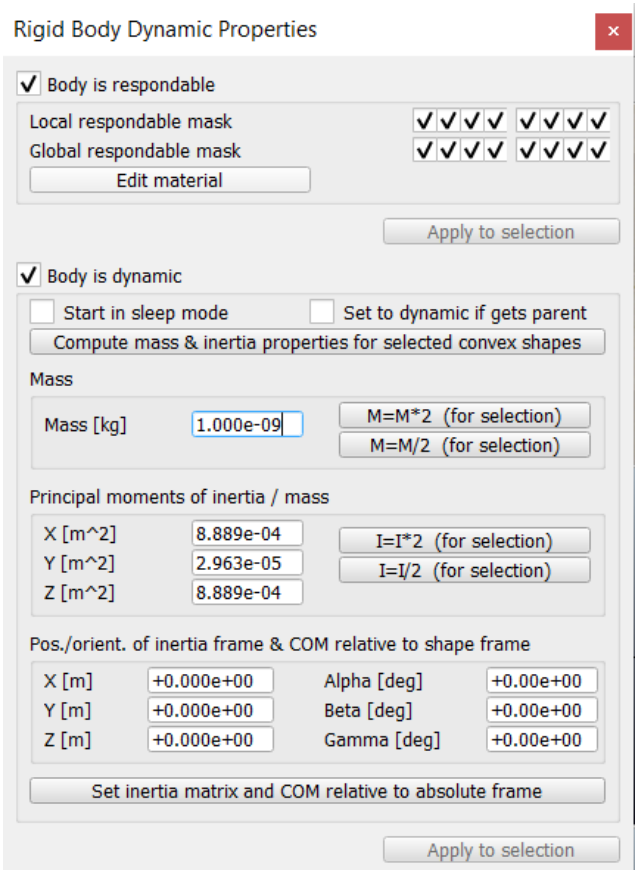
Σαν φορτίο προστίθεται ένα αντικείμενο τύπου Cuboid όπου στις δυναμικές του ιδιότητες έχει επιλεχθεί η επιλογή body is dynamic και η αρχική μάζα του αντικειμένου έχει οριστεί σε 0.



Εικόνα 2: Scene Hierarchy and Motor



Εικόνα 3: Joint properties



Εικόνα 4: Dummy load properties

2.1.β. Προγραμματισμός μοντέλου με τη χρήση γλώσσας LUA

Το CoppeliaSim δίνει την δυνατότητα της άμεσης εκτέλεσης κώδικα Lua από την κονσόλα του ή την εκτέλεση scripts τα οποία συνοδεύουν αντικείμενα στην ιεραρχία σκηνής.

Στην προκειμένη περίπτωση δεν χρειάζεται να τρέξει κάποια εντολή απευθείας από την κονσόλα καθώς ο κώδικας παραμένει σταθερός από την πλευρά του CoppeliaSim και οι αλλαγές στο σύστημα γίνονται μέσο του MATLAB.

Το Regualar Api του CoppeliaSim προσφέρει πληθώρα ετοίμων εντολών για τον έλεγχο τον παραμέτρων του λογισμικού και καθιστά ιδιαίτερα εύκολη την δημιουργία αξιόπιστων scripts.

Κάθε script πρέπει να συνοδεύει κάποιο αντικείμενο ώστε να μπορεί να εκτελεσθεί κατά την προσομοίωση. Σε αυτή την εργασία χρειάστηκαν μόνο non-thereaded τύπου scripts. Threaded scripts χρησιμοποιούνται σε περιπτώσεις που υπάρχει η ανάγκη εκτέλεσης διαφορετικών νημάτων μεταξύ διαφορετικών scripts, τα οποία όμως θα πρέπει να είναι συγχρονισμένα μεταξύ τους και υπάρχει η περίπτωση να χρειάζεται κάποιο να περιμένει.

Κάθε script έχει τέσσερις βασικές συναρτήσεις:

Συνάρτηση sysCall_init(). Πρόκειται για μια συνάρτηση που τρέχει μόνο μια φορά κατά την εκκίνηση του προγράμματος και χρησιμοποιείται για την αρχικοποίηση των τιμών καθώς και τη δημιουργία απαραίτητων μεταβλητών όπως οι object_Hadle.

Συνάρτηση sysCall_actuation(). Τρέχει πολλές φορές κατά την εκτέλεση του προγράμματος και χρησιμοποιείται κυρίως για τη συλλογή δεδομένων από τους αισθητήρες και την αποθήκευσή τους σε μεταβλητές.

Συνάρτηση sysCall_sensing(). Τρέχει πολλές φορές κατά την εκτέλεση του προγράμματος και χρησιμοποιείται για τον αναπροσδιορισμό των στόχων κάθε αντικείμενου που πρέπει να μεταβάλει κάποια του τιμή.

Τέλος η συνάρτηση sysCall_cleanup() χρησιμοποιείται μια φορά με σκοπό να επαναφέρει την σκήνη στην αρχική της κατάσταση καθώς και να επαναφέρει τιμές που έχουν τυχόν αλλάξει κατά την εκτέλεση.

2.1.γ. Ενεργοποίηση remote API για την σύζευξη με MATLAB

Για την επιτυχή επικοινωνία CoppeliaSim – MATLAB μέσω του remote API από την πλευρά του CoppeliaSim το μόνο που χρειάζεται είναι η δημιουργία ενός non-threaded child script οπουδήποτε μέσα στη σκηνή που έχουμε δημιουργήσει και να προστεθεί στην συνάρτηση sysCall_init() η εντολή "simRemoteApi.start(19999)" που σκοπός της είναι η εκκίνηση του remote API από την πλευρά του server(CoppeliaSim), στην προκειμένη περίπτωση στην πόρτα 19999.

Στη ενότητα 3.2.β. θα δούμε ολοκληρωμένη τη σύζευξη τον δυο λογισμικών καθώς και τους επιμέρους τύπους επικοινωνίας μεταξύ τους.

2.1.δ. Δημιουργία custom συναρτήσεων

Στο CoppeliaSim υπάρχουν remote API συναρτήσεις αποκλειστικά για χρήση με το MATLAB για βασικές λειτουργίες. Θα ήταν όμως σχεδόν αδύνατο κάθε συνάρτηση που υπάρχει στο regular API να είναι "μεταφρασμένη" και για το API του MATLAB. Για αυτόν τον λόγο υπάρχει η συνάρτηση simxCallScriptFunction() που σκοπός της είναι η κλήση προσαρμοσμένων συναρτήσεων για όποια λειτουργία δεν υπάρχει άμεσα διαθέσιμη μέσω του MATLAB. Ο τρόπος κλήσης της συνάρτησης simxCallScriptFunction() θα περιγραφεί εκτενέστερα στην ενότητα 2.2.δ .

Τέτοιες συναρτήσεις δηλώνονται κάτω από την sysCall_init() και για την κλήση τους απαραίτητη προϋπόθεση είναι να τρέχει η προσομοίωση.

```
function getJointVelocity(inInts,inFloats,inStrings,inBuffer)
    local v=sim.getJointVelocity(joint)
    return {},{v},{},''
end
```

Στο παραπάνω παράδειγμα έχουμε δημιουργήσει μια συνάρτηση με την ονομασία getJointVelocity που επιστρέφει μετά την κλίση της 4 τιμές. Καλώντας τη συνάρτηση sim.getJointelocity για το αντικείμενο joint, στην τοπική μεταβλητή "v" τοποθετείται η ταχύτητα περιστροφής του μοτέρ. Τέλος επιστρέφουμε η τιμή της μεταβλητής "v" επιστρέφεται στο δεύτερο κατά σειρά όρισμα που είναι τύπου float.

2.1.ε. code overview

```
1  function sysCall_init()
2      simRemoteApi.start(19999)
3      joint = sim.getObjectHandle('Revolute_joint')
4      VelocityGraph=sim.getObjectHandle('Graph')
5      Velocity=sim.addGraphStream(VelocityGraph,'Velocity','deg/s',2,{1,0,0},0)
6  end
7
8  function getJointVelocity(inInts,inFloats,inStrings,inBuffer)
9      local v=sim.getJointVelocity(joint)
10     return {},{v},{},''
11 end
12
13 function sysCall_actuation()
14     speed=sim.getJointVelocity(joint)
15     print(speed , 'deg/s')
16     sim.setGraphStreamValue(VelocityGraph,Velocity,speed)
17 end
18
19 function sysCall_sensing()
20 end
21
22 function sysCall_cleanup()
23 end
24
25
26
```

Στην συναρτιση sysCall_init()στην γραμμή:

2. Γίνεται εκκίνηση του remote API στην πόρτα 19999.
3. Ανατίθεται στην μεταβλητή joint η τιμή του object handle για το μοτέρ.
4. Ανατίθεται στην μεταβλητή VelocityGraph η τιμή του object handle για το Γράφημα.
5. Δημιουργείται ένα data stream για το γράφημα και ανατίθεται το return value του στην μεταβλητή Velocity .

Στη συνέχεια δημιουργείται η συνάρτηση getJointVelocity() οπου:

8. Ορίζονται οι τέσσερις είσοδοι της συνάρτησης

   (inInts,inFloats,inStrings,inBuffer)

9. Ορίζεται μια τοπική μεταβλητή "v" και της ανατίθεται το αποτέλεσμα της συνάρτησης sim.getJointVelocity() που επιστρέφει την ταχύτητα περιστροφής του μοτέρ.
10. Επιστρέφονται οι τέσσερις τιμές που έχει σαν έξοδο η συνάρτηση και στην δεύτερη κατά σειρά που είναι τύπου float ανατίθεται η τιμή της μεταβλητής "v".

    Τέλος στην συνάρτηση sysCall_actuation() στην γραμμή:

14. Ορίζεται μια μεταβλητή "speed" και τις ανατίθεται το αποτέλεσμα της συνάρτησης sim.getJointVelocity() που επιστρέφει την ταχύτητα περιστροφής του μοτέρ.
15. Τυπώνεται στην κονσόλα η μεταβλητή "speed".
16. Περνιέται η μεταβλητή "speed" στο διάγραμμα.

2.2 MATLAB

2.2.α. Δημιουργία του PID ελεγκτή

Για τον ελεγκτή έχουν δημιουργηθεί συνολικά τρεις συναρτήσεις η κάθε μια με την δική της λειτουργία.

Αρχικά έχει δημιουργηθεί η συνάρτηση init_pid() που λειτουργεί ως constructor και δημιουργεί και αρχικοποιεί τον ελεγκτή. Επίσης δημιουργήθηκαν δύο μεταβλητές που θα αποθηκεύουν την τιμή του προηγούμενου σφάλματος, καθώς και το σφάλμα του ολοκληρωτικού όρου.

```
function pid = init_pid(KP, KI, KD)
    pid.Kp = KP;
    pid.Ki = KI;
    pid.Kd = KD;
    pid.previousError = 0.0;
    pid.integralError = 0.0;
end
```

Επιπλέον, η συνάρτηση pid_output() δέχεται ως ορίσματα τον ελεγκτή που δημιουργήθηκε με την init_pid(), την επιθυμητή ταχύτητα περιστροφής, την πραγματική ταχύτητα περιστροφής και μια τιμή που ελέγχει την συχνότητα λειτουργίας του ελεγκτή.

```
function output = pid_output(pid, TargetVelocity, actualVelocity, dt)
    error = TargetVelocity - actualVelocity;
    if (pid.Ki > 0.0)
        pid.integralError = pid.integralError + error*dt;
    end
    derivativeError = (error - pid.previousError)/dt;
    pid.previousError = error;
    output = pid.Kp*error + output_adapting(pid.Ki*pid.integralError,1000) + pid.Kd*derivativeError;
    output = output_adapting(output,50000);   %  Max_force Nm
end
```

Το σφάλμα υπολογίζεται και στη συνέχεια ελέγχεται αν ο ολοκληρωτικός όρος I, που έχει ως σκοπό να εξαλείψει το μόνιμο σφάλμα, είναι μεγαλύτερος από το 0. Σε αυτήν την περίπτωση, αυξάνεται το σφάλμα του ολοκληρωτικού όρου κατά error*dt. Στη συνέχεια υπολογίζεται το σφάλμα του διαφορικού όρου το οποίο ορίζεται ως (error - pid.previousError)/dt και αντικαθίσταται η τιμή του προηγούμενου σφάλματος με την τιμή του καινούριου.

Η έξοδος του ελεγκτή ορίζεται ως το άθροισμα του κάθε όρου πολλαπλασιασμένο με το αντίστοιχο σφάλμα που τους αντιστοιχεί . Εδώ παρατηρούμε μια ιδιαιτερότητα στον υπολογισμό της τιμής για τον ολοκληρωτικό όρο, ο οποίος σε περίπτωση που το αποτέλεσμα δεν επιτευχθεί με την πάροδο του χρόνου, το σφάλμα του θα γίνεται όλο και μεγαλύτερο οπότε και θα πρέπει να το περιορίσουμε με τη συνάρτηση output_adapting() που θα περιγραφεί παρακάτω. Τέλος, κάνοντας πάλι χρήση της συνάρτησης output_adapting() η τελική έξοδος του ελεγκτή προσαρμόζεται ώστε να μην ξεπερνάει τη μέγιστη δύναμη που μπορεί να παράξει το μοτέρ.

Η συνάρτηση output_adapting() έχει δύο ορίσματα. Το πρώτο όρισμα είναι η τιμή που θέλουμε να προσαρμόσουμε και το όριο με βάση το οποίο θα προσαρμοστεί. Στην περίπτωση που η αρχική τιμή είναι μεγαλύτερη από το όριο η τιμή της αναπροσαρμόζεται στην τιμή του ορίου, ενώ σε περίπτωση που η τιμή της είναι μικρότερη από την αντίθετη τιμή του ορίου παίρνει την αντίθετη τιμή του ορίου και τέλος σ περίπτωση που η τιμή βρίσκεται στα ορισμένα όρια, παραμένει ίδια.

```
function output = output_adapting(x, limit)
    if (x > limit)
        output = limit;
    elseif (x < -limit)
        output = -limit;
    else
        output = x;
    end
end
```

2.2.β. Σύζευξη με CopelliaSim

Για να επιτευχθεί η σύζευξη του client(MATLAB) με το CoppeliaSim σε περιβάλλον Windows χρειάζεται αρχικά να αντιγραφεί το αρχείο remoteApi.dll από το μονοπάτι

"/V-REP_installation_directory/programming/remoteAPIBinding/lib/lib"

και τα αρχεία remoteApiProto.m και remApi.m από το μονοπάτι "/V-REP_installation_directory/programming/remoteAPIBinding/matlab", στον ίδιο κατάλογο με το ".m", αρχείο που περιέχει τον κώδικα MATLAB.

Με την εντολή "vrep = remApi('remoteApi');" δημιουργούμε το αντικείμενο vrep και φορτώνουμε την βιβλιοθήκη remoteApi .

Στη συνεχεία κάνοντας κλήση της εντολής simxStart ορίζουμε την διεύθυνση IP όπου βρίσκεται ο server μας, την πόρτα μεσώ της οποίας θέλουμε να συνδεθούμε, 2 boolean τιμές που καθορίζουν αν η συνάρτηση θα περιμένει να επιτευχθεί σύζευξη ή όχι και αν σε περίπτωση απώλειας σύζευξης θα επιδιωχθεί ξανά σύζευξη ή όχι , τον μέγιστο χρόνο αναμονής και, τέλος, τη συχνότητα με την οποία θα στέλνονται τα πακέτα. Σε περίπτωση μη επιτυχούς σύζευξης με τον server η συνάρτηση μας επιστρέφει την τιμή -1, ενώ σε οποιασδήποτε άλλη περίπτωση η σύζευξη είναι επιτυχής και επιστέφεται το ClientID που αντιστοιχεί στον client.

2.2.γ. Ενεργοποίηση Synchronous mode

Μετά την εξασφάλιση πετυχημένης επικοινωνίας μεταξύ server και client κανονικά θα έπρεπε να μπορεί να τρέξει ο ελεγκτής απευθείας από την πλευρά του client Λόγω όμως του ότι τα δεδομένα που αποστέλλονται από το MATLAB ουσιαστικά ελέγχουν την προσομοίωση προκύπτει το πρόβλημα της ασύγχρονης εκτέλεσης των δύο. Έτσι παρότι μπορεί ο server να λειτουργεί σωστά και οι υπολογισμοί του client να είναι επίσης σωστοί, το συνολικό αποτέλεσμα ενδέχεται να παρουσιάζει σφάλματα εξαιτίας του ότι ο καθένας τρέχει με τον δικό του ρυθμό.

Αυτό το πρόβλημα λύνεται με την ενεργοποίηση του Synchronous mode, το οποίο εγγυάται τη συγχρονισμένη λειτουργία client και server.

Οι βασικές εντολές που χρειαζόμαστε για τον συγχρονισμό είναι δύο και είναι οι simxSynchronous και simxSynchronousTrigger.

Η simxSynchronous παίρνει σαν όρισμα το ClientID που δημιουργήθηκε από την simxStart και μια μεταβλητή τύπου Boolean για την ενεργοποίηση της διαδικασίας.

Η simxSynchronousTrigger στέλνει ένα σήμα στον server για να του επιστέψει την συνέχεια στο επόμενο βήμα της προσομοίωσης.

Το block κώδικα που περικλείεται από αυτές τις δύο εντολές κατά την εκτέλεσή του έχει τον πλήρη έλεγχο της προσομοίωσης και μπορεί να εκτελείται για όση ώρα χρειάζεται κάθε φορά.

2.2.δ. simxCallScriptFunction

Αυτή η συνάρτηση έχει σαν σκοπό της να καλύψει οποιαδήποτε "κενά" υπάρχουν μεταξύ του regular και του remote API. Μας δίνει την δυνατότητα κατά την διάρκεια της προσομοίωσης να καλέσουμε scripts τα οποίο έχουμε δημιουργήσει στο CoppeliaSim.

Η συνάρτηση έχει εννέα ορίσματα:

1. Το clientID που δημιουργήθηκε στην αρχή της σύζευξης από την simxStart.
2. Το όνομα του αντικειμένου στον κώδικα του οποίου υπάρχει η συνάρτηση που θέλουμε να καλέσουμε.
3. Το είδος του script που βρίσκεται η συνάρτηση.
4. Το όνομα της συνάρτησης που θέλουμε να καλέσουμε.

5-7. Τρεις μεταβλητές τύπου Int, Float και String που θα δοθούν στην συνάρτηση σαν είσοδος. Οι μεταβλητές αυτές μπορούν να είναι και πίνακες.

8. Τον input buffer που στέλνουμε στην συνάρτηση ο οποίος μπορεί να είναι και πίνακας.

9. Το operationMode για την συνάρτηση, το οποίο συστήνεται να είναι τύπου opmode_blocking.

Επιστρέφονται 5 μεταβλητές:

Ένα returnCode το οποίο ενημερώνει για την επιτυχή ή μη εκτέλεσή της μέσω ενός συνόλου flags. Στην περίπτωση επιτυχούς εκτέλεσης επιστρέφεται η τιμή 0. Τρεις μεταβλητές τύπου Int, Float και String που μπορούν να είναι και πίνακες και επιστέφουν τις εξόδους από την συνάρτηση που κλήθηκε και τέλος τον buffer που επιστρέφεται μετά την εκτέλεση.

2.2.ε. code overview

```
clear all
clc
disp('Program started');
vrep = remApi('remoteApi');
vrep.simxFinish(-1);
clientID = vrep.simxStart('127.0.0.1',19999,true,true,5000,5);

Optimal_speed = 10;
Max_speed = 50;
dt = 0.05;

if (clientID > -1)
    disp('Connected to remote API server');
    pid = init_pid(0, 0, 0);

   [returnCode,Motor]=vrep.simxGetObjectHandle(clientID,'Revolute_joint',vrep.simx_opmode_blocking);

    while (clientID ~= -1)
        vrep.simxSynchronous(clientID,true);
        [ret, retInts, retFloats, retStrings, retBuff]=vrep.simxCallScriptFunction(clientID,'Revolute_joint',
        vrep.sim_scripttype_childscript,'getJointVelocity',[],[],[],[],vrep.simx_opmode_blocking);
        output = pid_output(pid, Optimal_speed, retFloats(1), dt)

        if (retFloats(1) <= Optimal_speed)

            vrep.simxSetJointMaxForce(clientID,Motor,output,vrep.simx_opmode_streaming);
            vrep.simxSetJointTargetVelocity(clientID,Motor,Max_speed,vrep.simx_opmode_streaming);
        else
            vrep.simxSetJointMaxForce(clientID,Motor,abs(output)/10,vrep.simx_opmode_streaming);
            vrep.simxSetJointTargetVelocity(clientID,Motor,1,vrep.simx_opmode_streaming);
        end

        vrep.simxSynchronousTrigger(clientID);
    end

else
    disp('Failed connecting to remote API server');
end
    vrep.delete();
    disp('Program ended');
```

Επίσης γίνετε χρήση τον συναρτήσεων από την ενότητα 5.2.1 .

Γραμμή:

1. Καθαρισμός του buffer
2. καθαρισμός command window
3. Εμφάνιση μηνύματος
4. Δημιουργία αντικειμένου και φόρτωση βιβλιοθήκης
5. Τερματισμός οποιασδήποτε ανοιχτής προσομοίωσης για την αποφυγή τυχόν σφαλμάτων
6. Δημιουργία clientId και εκκίνηση επικοινωνίας

8-9-10. Ορισμός αρχικών τιμών για τις μεταβλητές Optimal_speed, Max_speed και dt.

12. Έλεγχος αν έχει ξεκινήσει η επικοινωνία.
13. Εκτύπωση μηνύματος επιτυχούς σύνδεσης.

14. Αρχικοποίηση του ελεγκτή

16. Κλήση της εντολής simxGetObjectHandle του remote API και αποθήκευση του objectHandle για το μοτέρ στην μεταβλητή Motor

18-23. Βρόγχος ελέγχου του συστήματος

18. Όσο υπάρχει επικοινωνία άρα το ClientID είναι διαφορετικό του -1

19. Ενεργοποίηση του synchronoys mode

20-21. Κλήση της εντολής getJointVelocity που βρίσκεται στο script του μοτέρ στο CoppeiaSim μέσω της simxCallScriptFunction

22 Ανάθεση στην μεταβλητή output της τιμής εξόδου του ελεγκτή που υπολογίζεται από την συνάρτηση pid_output(pid, Optimal_speed, retFloats(1), dt) όπου : pid είναι ο ελεγκτής που δημιουργήθηκε στη σειρά 14 , Optimal_speed και dt οι τιμές που αρχικοποιήθηκαν στις γραμμές 8 και 10 και retFloats(1) η τιμή της τρέχουσας ταχύτητας που προέκυψε από την κλήση της εντολής getJointVelocity μέσω της simxCallScriptFunction.

24. Αν η ταχύτητα είναι ίση ή μικρότερη από την ιδανική

26. Ορισμός της ροπής του μοτέρ σύμφωνα με την έξοδο του ελεγκτή

27. Δυνατότητα επίτευξης μέγιστης ταχύτητας στο μοτέρ

28. Αλλιώς

29 Ορισμός της ροπής του μοτέρ στην απόλυτη τιμή του ενός δεκάτου της εξόδου του ελεγκτή

30. Ορισμός της μέγιστης ταχύτητας του μοτέρ ως 1 deg/s

33. Αποστολή σήματος στο CoppeliaSim για την συνέχιση της προσομοίωσης

37. Εκτύπωση μηνύματος αποτυχίας σε περίπτωση αυτής

39-40. Τερματισμός επικοινωνίας και εκτύπωση του αντιστοίχου μηνύματος

ΚΕΦΑΛΑΙΟ 3 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΛΕΓΚΤΗ

ΧΕΙΡΟΚΙΝΗΤΑ

Μιας και ο σκοπός αυτής της εργασίας είναι η κατανόηση τον pid ελεγκτών θα φτάσουμε όσο το δυνατόν πιο κοντά στη βέλτιστη λύση ρυθμίζοντας τον ελεγκτή χειροκίνητα και συγκρίνοντας τα δεδομένα που θα συλλεχθούν.

Αρχικά θα πρέπει να οριστούν οι απαραίτητες παράμετροι για την προσομοίωση που θα ακολουθήσει.

Αυτές είναι:

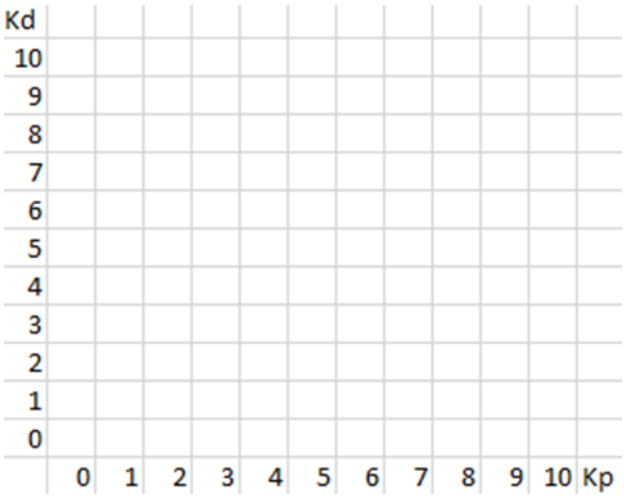
1. Η μέγιστη ροή που μπορεί να παράξει το μοτέρ την οποία και ορίζουμε στα 70Nm .
2. Η μέγιστη ταχύτητα περιστροφής του μοτέρ την οποία ορίζουμε ως 50Deg/s.
3. Η επιθυμητή ταχύτητα περιστροφής 10Deg/s.
4. Το βάρος του dymmy load το οποίο είναι 4000kg.
5.

3.1.α. Παράδειγμα 1

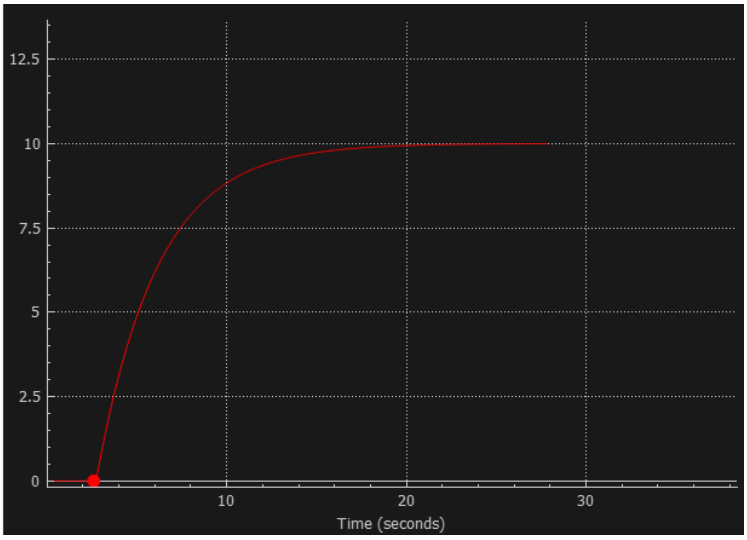
Ας ονομάσουμε αυτό το παράδειγμα «Παράδειγμα 1».

Η διαδικασία που θα ακολουθήσουμε είναι η εξής.

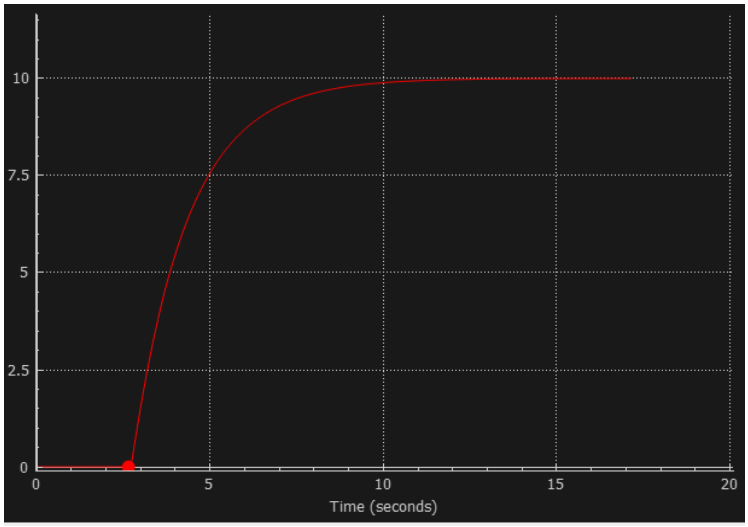
Αρχικά θα δημιουργήσουμε έναν δυοδιάστατο διόραμα χ, ψ για τον αναλογικό και τον διαφορικό όρο αντίστοιχα.



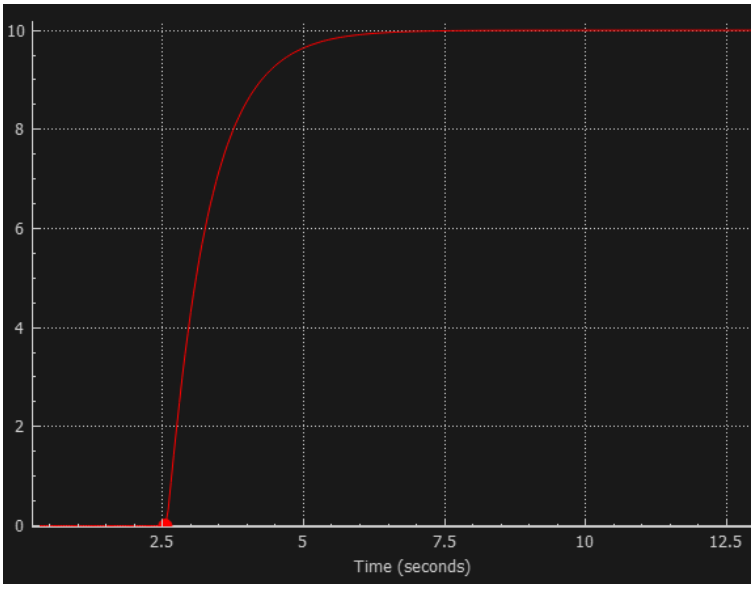
Ξεκινώντας από τον αναλογικό όρο θα προσπαθήσουμε να κάνουμε την απόκριση του συστήματος μας όσο το δυνατόν ταχύτερη δίνοντας στον ελεγκτή τις τιμές από 1 ως και 9 καταγράφοντας κάθε φορά τον αριθμό της προσπάθειάς μας.



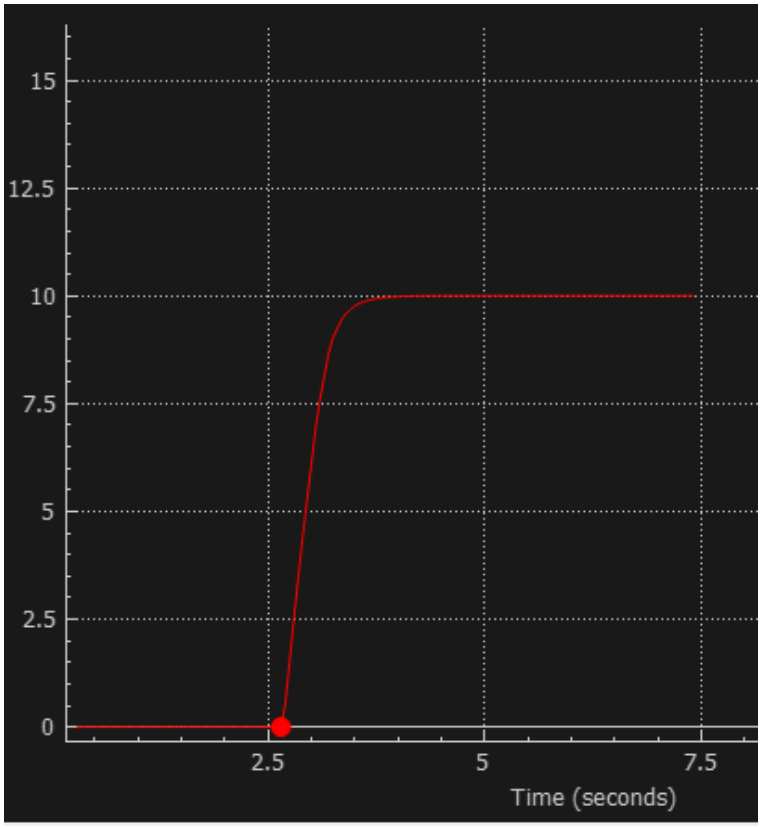
pid(1,0,0)



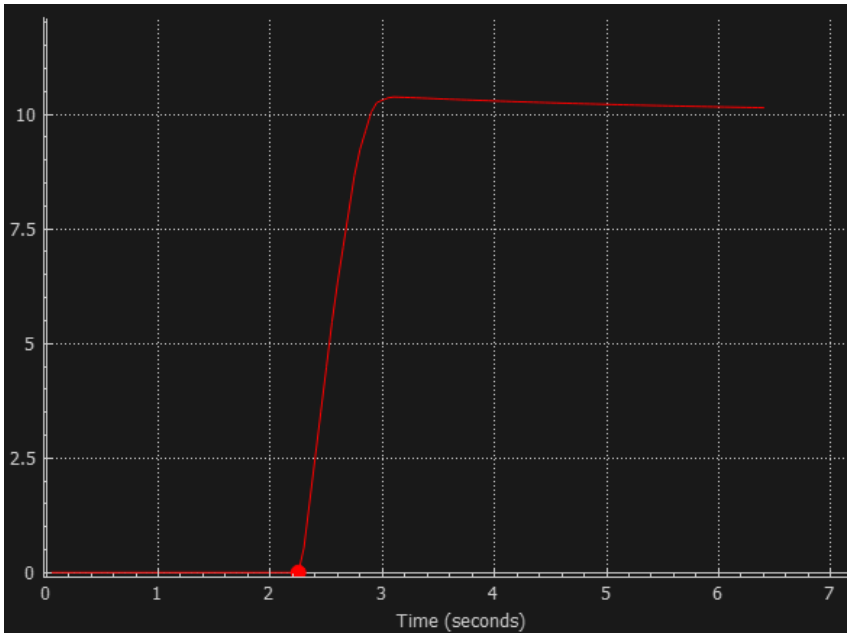
pid(2,0,0)



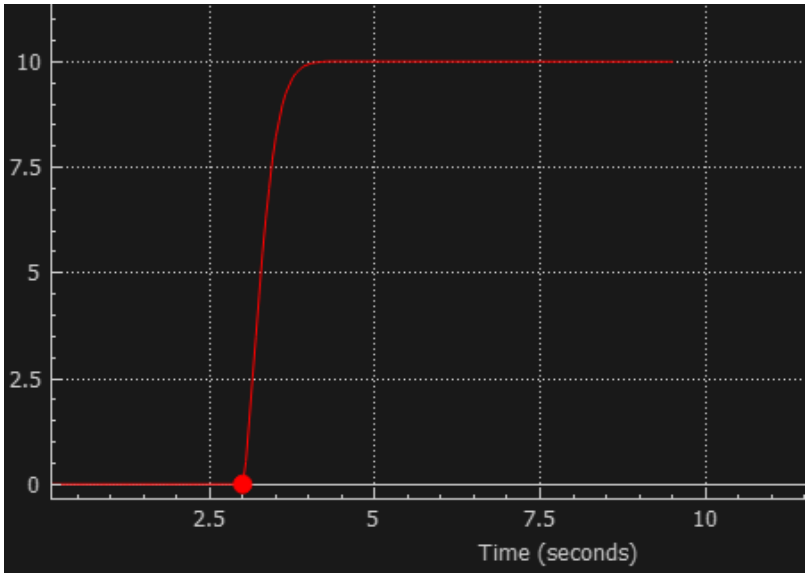
pid(4,0,0)



pid(8,0,0)



pid(10,0,0)

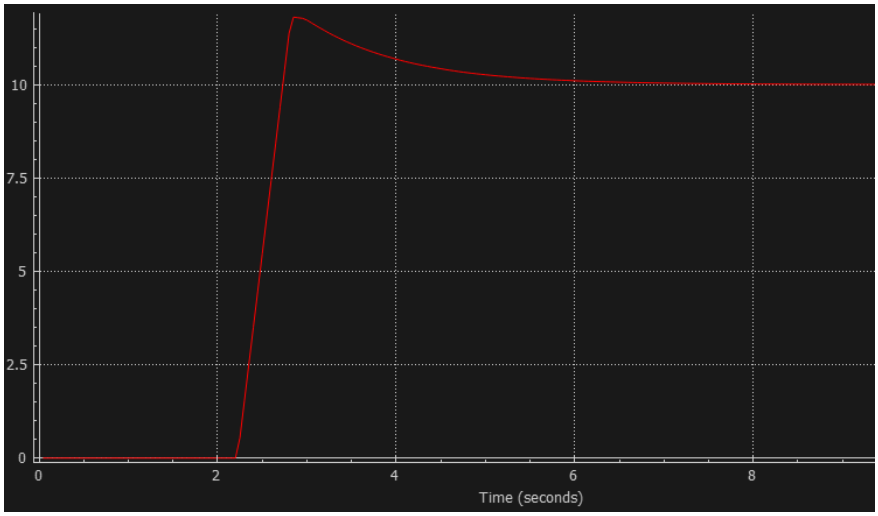


pid(9,0,0)

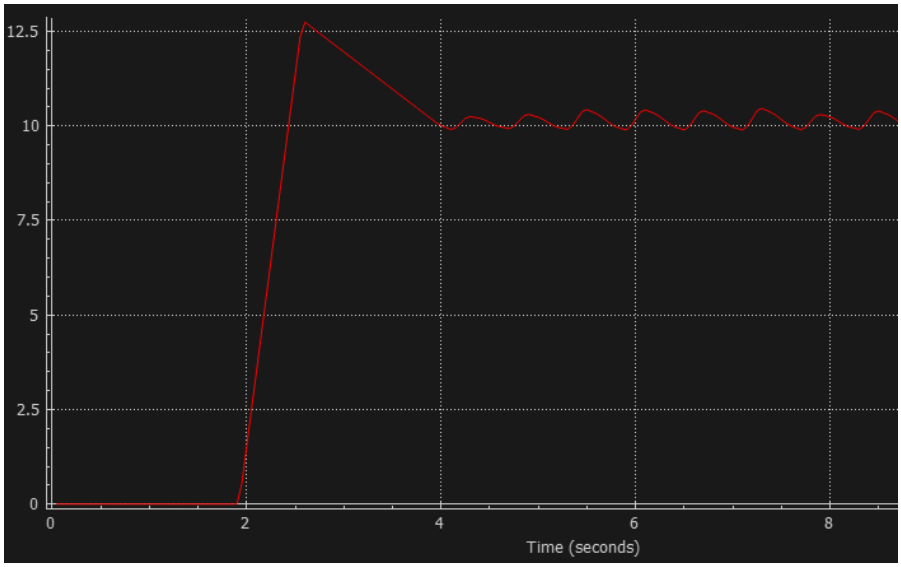
Κάθε φορά που πραγματοποιείται μια δοκιμή με διαφορετικές τιμές για τον ελεγκτή, περνάνε στην αντίστοιχη θέση του πίνακα τον αριθμό της προσπάθειας.

Kd												
10												
9												
8												
7												
6												
5												
4												
3												
2												
1												
0		1	2		3				4	6	5	
	0	1	2	3	4	5	6	7	8	9	10	Kp

Σύμφωνα με τα αποτελέσματα η ιδανική τιμή του αναλογικού όρου είναι η τιμή 8 ή 9. Διαλέγουμε την τιμή 9 λόγω καλύτερης απόκρισης και αρχίζουμε την ίδια διαδικασία αλλάζοντας την τιμή του διαφορικού όρου.



pid(9,0,1)



pid(9,0,10)

Από την δοκιμή των τιμών 1 και 10 για τον διαφορικό όρο παρατηρούμε ότι το σύστημα έχει χειρότερη συμπεριφορά, ξεπερνώντας κατά πολύ την επιθυμητή του ταχύτητα και χρειάζεται συνολικά περισσότερο χρόνο για να φτάσει σε αυτή. Επίσης παρατηρούμε ότι για την τιμή 1 το σύστημα με την πάροδο του χρόνου επιτυγχάνει τον στόχο του και σταθεροποιείται ενώ για την τιμή 10 οι διακυμάνσεις του είναι εμφανείς.

Kd												
10										8		
9												
8												
7												
6												
5												
4												
3												
2												
1											7	
0		1	2		3				4	6	5	
	0	1	2	3	4	5	6	7	8	9	10	Kp

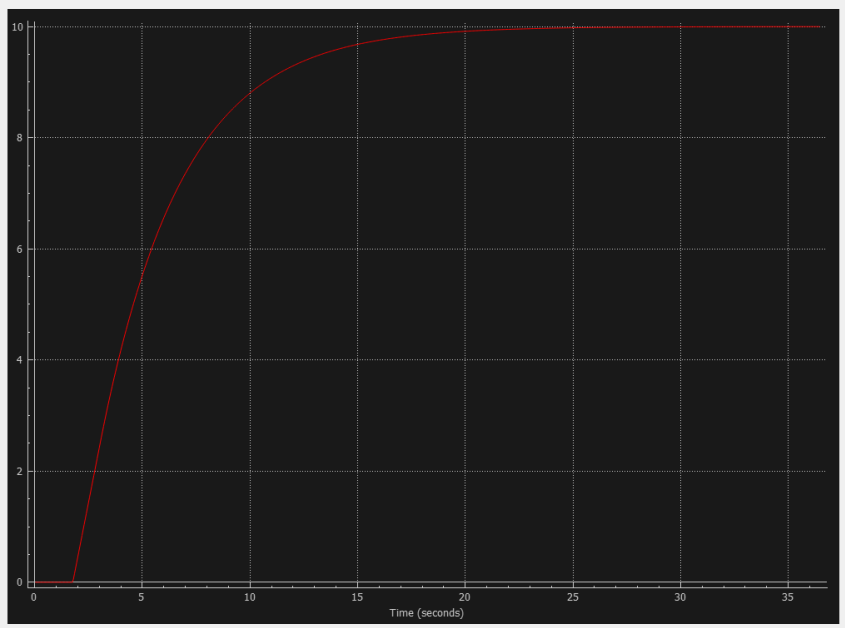
Συμπερασματικά το σύστημα φαίνεται να έχει την καλύτερη απόκριση για τις τιμές P=9 και D=0 στις 8 προσπάθειες που έχουν γίνει. Μένει να γίνουν οι αντίστοιχες δοκιμές και για τον ολοκληρωτικό όρο, κάτι το οποίο δεν χειμάζεται στην ουσία αν παρατηρήσουμε τα αποτελέσματα για τον ελεγκτή (9,0,0), κατά τα οποία δεν υπάρχει steady state error και άρα και η χρήση του δεν είναι αναγκαία.

Συμπεραίνουμε, λοιπόν, ότι το μοντέλο του παραδείγματος 1 θα μπορούσε να ελεγχθεί επιτυχώς και με έναν απλό αναλογικό ελεγκτή με απόκριση 1,4sec και σφάλμα μικρότερο του 1%.

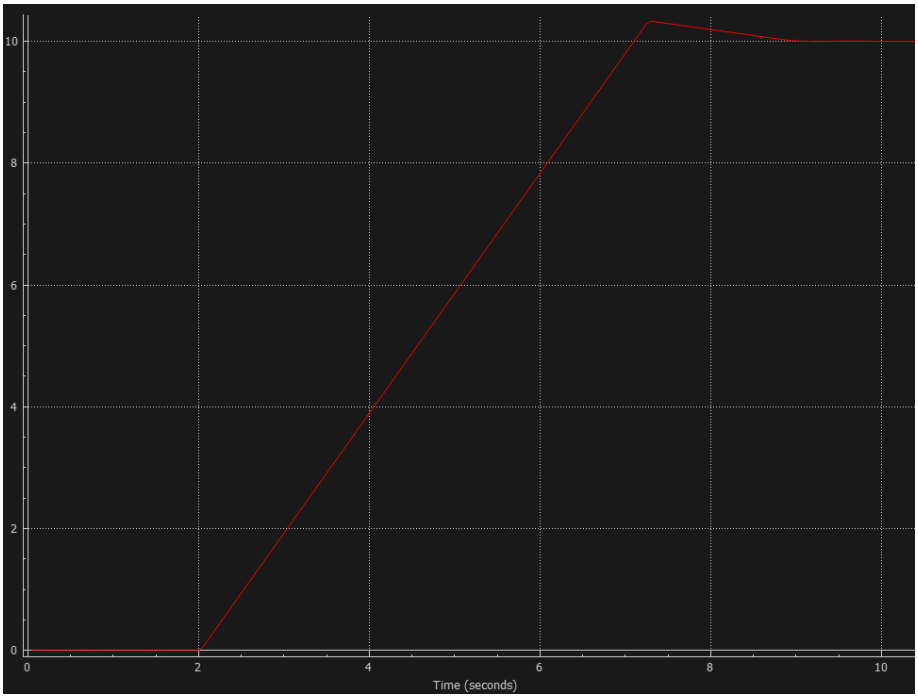
3.1.β. Παράδειγμα 2

Στο επόμενο παράδειγμα η τιμή για το βάρος του dymmy load έχει αλλάξει από 4000kg σε 40000kg.

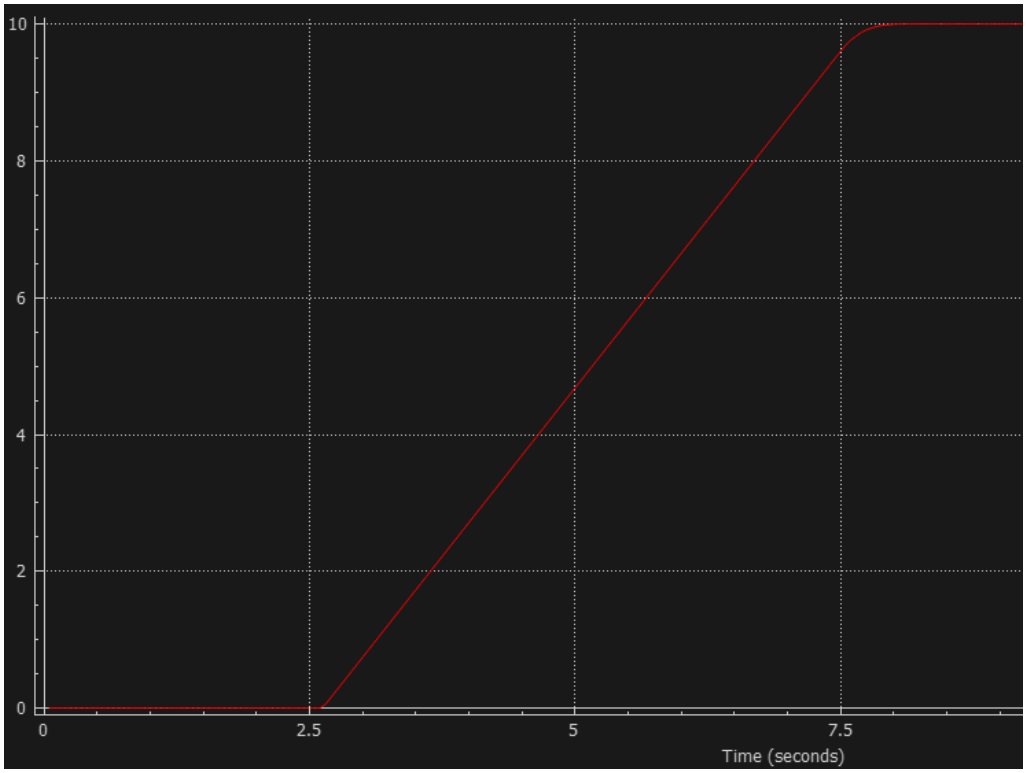
Αρχικά δοκιμάζοντας τις ιδανικές τιμές για το παράδειγμα 1 φαίνεται πως το σύστημα χρειάζεται περίπου 20 δευτερόλεπτα για να σταθεροποιηθεί .



pid(9,0,0)



pid(1000,0,0)



pid (100,0,0)

Άρα και στο παράδειγμα 2 ο έλεγχος θα μπορούσε να γίνει αποκλειστικά με έναν αναλογικό ελεγκτή με τιμή 100 και η τιμή θα σταθεροποιούνταν μετά από 5,4 δευτερόλεπτα.

ΑΥΤΟΜΑΤΑ

Για την αυτόματη ρύθμιση ενός ελεγκτή υπάρχουν αρκετές μέθοδοι η κάθε μια με τα προτερήματα και τα μειονεκτήματα της. Κάθε μια ακολουθεί τους δικούς τις κανόνες για να πετύχει το καλύτερο δυνατό αποτέλεσμα. Κάποιες από τις πιο γνώστες είναι οι Ziegler-Nichols, Chien, Hrones and Reswick, Cohen-Coon, Kappa-tau και Lambda.

3.2.α. Ziegler-Nichols.

Είναι μια αρκετά διαδεδομένη μέθοδος με επιθετικά αποτελέσματα που πολλές φορβές επιτρέπει στο αποτέλεσμα να ξεπεράσει την επιθυμητή τιμή ακόμα και 25% κατά τη διαδικασία της βελτιστοποίησης.

Τα βασικά της πλεονεκτήματα είναι η ευκολία στην χρήση και τα αρκετά καλά αποτελέσματα της σε απλά loops καθώς και η δυνατότητα να λειτουργήσει χωρίς να υπάρχει βαθιά γνώση τον διεργασιών που εκτελούνται.

Βασικό της μειονέκτημα είναι η υπερβολικά μεγάλη απόκλιση που μπορεί να έχει από το επιθυμητό αποτέλεσμα, η οποία φτάνει το 25% όπως και ότι έχει σχεδιαστεί για να καταπολεμά επιθετικές αλλαγές στο αποτέλεσμα και όχι για να παρακολουθεί και να διατηρεί μια σταθερή τιμή.

3.2.β. Cohen-Coon

Είναι βασισμένη στην μέθοδο Ziegler-Nichols χρησιμοποιεί όμως περισσότερα δεδομένα από το σύστημα που χειρίζεται και έτσι έχει καλύτερη συνολική απόδοση . Τα πλεονεκτήματα και μειονεκτήματα της είναι ίδια με αυτά της Ziegler-Nichols με μόνη διαφορά την ότι αποκρίνεται καλύτερα με συστήματα με μεγάλες αναμονές.

Cohen Coon	K_c	T_I	T_D
P	$\frac{1}{a}\left(1+\frac{0.35\tau}{1-\tau}\right)$		
PI	$\frac{0.9}{a}\left(1+\frac{0.92\tau}{1-\tau}\right)$	$\frac{3.3-3.0\tau}{1+1.2\tau}L$	
PD	$\frac{1.24}{a}\left(1+\frac{0.13\tau}{1-\tau}\right)$		$\frac{0.27-0.36\tau}{1-0.87\tau}L$
PID	$\frac{1.35}{a}\left(1+\frac{0.18\tau}{1-\tau}\right)$	$\frac{2.5-2.0\tau}{1-0.39\tau}L$	$\frac{0.37-0.37\tau}{1-0.81\tau}L$

$\tau = L/(L+T)$; $a = K_{process}L/T$

Cohen-Coon method

3.2.γ. Kappa-Tau

Ουσιαστικά είναι η εξέλιξη της Ziegler-Nichols σχεδιασμένη να απαλείψει προβλήματα όπως η μεγάλη απόκλιση από την επιθυμητή τιμή και η μη αποδοτική λειτουργία σε συστήματα με μεγάλες αναμονές.

Βασικά πλεονεκτήματα είναι η εξάλειψη του overshoot και βασικό μειονέκτημα η αδυναμία να ορίσει απαιτήσεις για απόδοση σε κλειστούς βρόγχους .

$f(\tau)$	Slow controller			Fast controller		
↓	a_0	a_1	a_2	a_0	a_1	a_2
aK_C	0.29	-2.7	3.7	0.78	-4.1	5.7
T_I/L	8.9	-6.6	3.0	8.9	-6.6	3.0
α	0.81	0.73	1.9	0.44	0.78	-0.45

$$f(\tau) = a_0 \exp(a_1 \tau + a_2 \tau^2)$$

Kappa-Tau method

3.2.δ. Lambda

Τα βασικά πλεονεκτήματα της μεθόδου αυτής είναι ο ορισμός μια σταθεράς χρόνου που ελέγχει τον χρόνο απόκρισης του ελεγκτή, η εξάλειψη του overshoot, η ευελιξία που έχει στην διαχείριση αλλαγών στο σύστημα και η σωστή λειτουργία του ελεγκτή ακόμα και σε συστήματα με μεγάλους χρόνους αναμονής.

Τα κύρια μειονεκτήματα της μεθόδου είναι η αδυναμία της να ελέγξει τον ολοκληρωτικό όρο, η αδυναμία να ορίσει απαιτήσεις για απόδοση σε κλειστούς βρόγχους και η αργή ταχύτητα με την οποία εξαλείφει όποιες διαταραχές δημιουργηθούν στο σύστημα (ιδιαίτερα εμφανές πρόβλημα σε συστήματα με αργή απόκριση).

Lambda	K_c	T_I
PI	$\frac{T}{K_p(\lambda + L)}$	T

Typically $\lambda = 3 \cdot \max(L,T)$ (very stable loop)

Reduce λ to get a faster response

Lambda method

ΒΙΒΛΙΟΓΡΑΦΙΑ

https://www.lua.org/about.html

https://en.wikipedia.org/wiki/CoppeliaSim

https://www.puc-rio.br/english/

https://www.coppeliarobotics.com/

https://www.mathworks.com/discovery/what-is-matlab.html

https://en.wikipedia.org/wiki/MATLAB

https://www.researchgate.net/publication/316709017_Design_of_HMI_Based_on_PID_Control_of_Temperature

https://en.wikipedia.org/wiki/DC_motor#Electromagnetic_motors

https://en.wikipedia.org/wiki/Brushless_DC_electric_motor

https://en.wikipedia.org/wiki/Servomotor#Motors

https://www.parvalux.com/the-history-of-dc-motors/

https://en.wikipedia.org/wiki/PID_controller#Integral

https://www.incatools.com/pid-tuning/pid-tuning-methods/

https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=ControlPID

https://openclass.teiwm.gr/modules/document/file.php/EE107/7_%CE%95%CE%9B%CE%95%CE%93%CE%9A%CE%A4%CE%95%CE%A3%20PID.pdf

http://gun.teipir.gr/DSAELAB/Ergastiriakes/pidtutorial.pdf

https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm