ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
"ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΕΛΕΓΧΟΥ"

Έλεγχος Θερμοκρασίας με ανεμιστήρα χρησιμοποιώντας την πλατφόρμα μικροελεγκτών Arduino

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Ι. Νικηφοράκης

Επιβλέπων: Νικόλαος Καραμπετάκης
Αν. Καθηγητής Α.Π.Θ.

Θεσσαλονίκη, Ιανουάριος 2013

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
"ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΕΛΕΓΧΟΥ"

Έλεγχος Θερμοκρασίας με ανεμιστήρα χρησιμοποιώντας την πλατφόρμα μικροελεγκτών Arduino

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Ι. Νικηφοράκης

Επιβλέπων: Νικόλαος Καραμπετάκης
Αν. Καθηγητής Α.Π.Θ.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

.....	.....	.....
Ν. Καραμπετάκης	Ε. Αντωνίου	Σ. Βολογιαννίδης
Αν. Καθηγητής Α.Π.Θ.	Επ. Καθηγητή Α.Τ.Ε.Ι.Θ.	Καθηγητής Τ.Ε.Ι. Σερρών.

Θεσσαλονίκη, Ιανουάριος 2013

……..
Μάριος Νικηφοράκης
Πτυχιούχος Μηχ. Αυτοματισμών Α.Τ.Ε.Ι.Θ.

ΠΕΡΙΛΗΨΗ

Ο σκοπός της εργασίας είναι η ανάλυση των ελεγκτών PID (Proportional, Integral, Derivative) με χρήση της πλακέτας Arduino. Η απεικόνιση των δεδομένων όπως και η ανάλυση και μοντελοποίηση των άγνωστων συστημάτων γίνεται μέσω προγραμμάτων που υλοποιήθηκαν και εργαλείων του Matlab.

Η συνεχής ανάγκη για καλύτερη ακρίβεια στην ρύθμιση κάποιον μεγεθών απαιτεί όλο και πιο σύγχρονες και καλύτερες μεθόδους ελέγχου. Ένας αρκετά ικανοποιητικός τύπος ελεγκτή, ο οποίος βρίσκει αρκετές χρήσεις σε βιομηχανικές περισσότερο αλλά και σε άλλες εφαρμογές, είναι ο αναλογικός, ολοκληρωτικός και διαφορικός ελεγκτής (PID). Ο ελεγκτής αυτός εξάγει αποτέλεσμα για τον έλεγχο του συστήματος όχι μόνο από το σφάλμα (διαφορά επιθυμητής τιμής με ανάδραση) αλλά επίσης από την κλίση του σφάλματος (διαφορικό κομμάτι) και από το εμβαδό του σφάλματος στον χρόνο (ολοκληρωτικό κομμάτι).

Στην παρούσα εργασία γίνεται η κατασκευή ελεγκτή θερμοκρασίας και ελεγκτή στροφών ανεμιστήρα. Ο πρώτος ελεγκτής είναι συνδεδεμένος σε σειρά με κλειστό βρόχο που περιέχει τον δεύτερο ελεγκτή. Έτσι έγινε μια κατασκευή η οποία περιέχει έναν αισθητήρα θερμοκρασίας για την ανάδραση θερμοκρασίας, έναν ανεμιστήρα για την ψύξη του σύστηματος που θέλουμε να ελέγξουμε και μια λυχνία η οποία θερμαίνει σταθερά το περιβάλλον (για αλλαγή συνθηκών περιβάλλοντος). Η ανάδραση των στροφών του κινητήρα γίνεται μέσω αισθητήρα που περιέχεται στον ανεμιστήρα. Επίσης κατασκευάστηκε πρόγραμμα γραφικού περιβάλλοντος (GUI) σε Matlab για την real – time απεικόνιση των εισόδων και εξόδων των ελεγκτών, όσο και για την ρύθμιση των παραμέτρων του ελέγχου όπως για τα setpoints και τα κέρδη των ελεγκτών. Ο έλεγχος εκτελείται στην πλακέτα του Arduino ενώ η επικοινωνία της πλακέτας με τον υπολογιστή γίνεται μέσω USB και κατάλληλων προγραμμάτων.

Το έγγραφο αρχικά περιέχει στα δύο πρώτα εισαγωγικά κεφάλαια μια βιβλιογραφική αναφορά για το Arduino και για το Matlab αντίστοιχα. Έτσι σε αυτά τα κεφάλαια υπάρχουν ιστορικές περιγραφές για τα αντικείμενα, ανάλυση των χαρακτηριστικών τους, όπως και τεχνικά ζητήματα που αφορούν την διαχείριση αυτών των εργαλείων.

Στο τρίτο κεφάλαιο αρχικά αναλύεται ο στόχος της εργασίας, δίνονται τα διαγράμματα βαθμίδων του συστήματος και στην συνέχεια γίνεται η περιγραφή της

κατασκευής που υλοποιήθηκε για να γίνει η εκτέλεση των πειραμάτων και η απεικόνιση των αποτελεσμάτων σε πραγματικό χρόνο. Έτσι αρχικά αναλύεται η κατασκευή ως προς το hardware που υλοποιήθηκε τόσο κυκλωματικά, όσο και κατασκευαστικά (χωροταξικά). Στην συνέχεια αναλύονται τα κομμάτια κώδικα που υλοποιήθηκαν (software) που είναι τα εξής: κώδικες σε Arduino για σειριακή επικοινωνία και έλεγχο και κώδικες σε Matlab για την σειριακή επικοινωνία, το γραφικό περιβάλλον του χρήστη και την εμφάνιση των αποτελεσμάτων που αποθηκεύτηκαν.

Το τέταρτο και πέμπτο κεφάλαιο αποτελούν την προσπάθεια μοντελοποίησης του συστήματος ελέγχου ανεμιστήρα και θερμοκρασίας αντίστοιχα. Επίσης γίνεται σύγκριση του θεωρητικού μοντέλου με το πραγματικό καθώς και tuning του ελεγκτή με χειροκίνητο τρόπο αλλά και αυτόματο μέσω του θεωρητικού μοντέλου που υπολογίστηκε.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Arduino, matlab, PID ελεγκτής, μοντελοποίηση, απεικόνιση δεδομένων, ελεγκτής θερμοκρασίας, ελεγκτής στροφών

ABSTRACT

The purpose of this paper is to analyze the PID controllers (Proportional, Integral, Derivative) using Arduino. The representation of data as well as analysis and modeling of unknown systems is implemented through programs and tools of Matlab.

The continuing need for better precision in system analysis requires ever more modern and better control methods. A fairly satisfactory type of controller, who finds many uses in industrial and other applications, is the proportional, integral and derivative controller (PID). The PID controller outputs the result to control the system not only by the error (difference setpoint with feedback) but also from the slope of the error (differential part) and the error area over time (integral part).

In this paper described the construction of a thermostat and a fan speed controller. The first controller is connected in series with a closed loop containing the second controller. So it was made an electronic device that contains a temperature sensor for temperature feedback, one fan for the cooling of the controlled system and a lamp for the heating of the system. Feedback speed control of the fan implement throught a tacho sensor containing in the fan. Also built a graphical user interface program (GUI) in Matlab for real - time display of incoming and outgoing data of the controllers, and to configure the setpoints and gains of the controllers. The PID controllers are running in the Arduino board and the board's communication with the PC is via USB and appropriate programs.

The document initially contains the first two introductory chapters a literature reference for the Arduino and Matlab respectively. So these chapters are historical descriptions of the objects, analyze their characteristics, as well as technical issues related to the management of these tools.

In the beginning of third chapter discussed the scope of this work by giving the block diagrams of the system, then, a description of the pid device for the execution of experiments and real time data acquisition. So first analyzes the devices hardware implemented as electronic circuit, and construction layout. Afterwards, the software code described, which are: codes in Arduino for serial communication and control and codes in Matlab for serial communication, the graphical user interface and a code to display the stored results.

The fourth and fifth chapter will attempt modeling fan control system and temperature control system respectively. Also compare the theoretical model with the real and tuning the controller in manual mode and automatic through the theoretical model that calculated.

KEY WORDS

Arduino, matlab, PID control, modeling, real time data acquisition, temperature controller, speed controller.

ΠΡΟΛΟΓΟΣ

Το παρόν έγγραφο αποτελεί την αναφορά της πτυχιακής εργασίας που υλοποιήθηκε στα πλαίσια του μεταπτυχιακού προγράμματος "ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΕΛΕΓΧΟΥ" κατά το ακαδημαϊκό έτος 2011-2012.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου. Αναπληρωτή Καθηγητή του τμήματος Μαθηματικών, κ. Νικόλαο Καραμπετάκη για την πολύτιμη βοήθεια του καθώς και για τις πολύτιμες γνώσεις με τις οποίες με εφοδίασε καθ'όλη τη διάρκεια του μεταπτυχιακού προγράμματος.

Επίσης θα ήθελα να ευχαριστήσω τον Καθηγητή του ΤΕΙ Σερρών Σταύρο Βολογιαννίδη για την βοήθεια και καθοδήγηση του για την σωστή ανάπτυξη και διεκπεραίωση της διπλωματικής εργασίας και τον επ. καθηγητή Α.Π.Θ. Ε. Αντωνίου για τον χρόνο που αφιέρωσε στη μελέτη και αξιολόγηση της εργασίας.

Τέλος, θέλω να ευχαριστήσω όλους όσους με στήριξαν και με βοήθησαν για την υλοποίηση αυτής της εργασίας.

Αφιερώνεται
στην Αργυρώ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1
Arduino

1.1 Εισαγωγή

Όπως το περιγράφει ο δημιουργός του, το Arduino είναι μία ανοικτού κώδικα (open-source) πλατφόρμα «πρωτοτυποποίησης» ηλεκτρονικών κυκλωμάτων βασισμένη σε ευέλικτο και εύκολο στη χρήση υλικό (hardware) και λογισμικό (software) που προορίζεται για οποιονδήποτε έχει λίγη προγραμματιστική εμπειρία, στοιχειώδεις γνώσεις ηλεκτρονικών και ενδιαφέρεται να δημιουργήσει διαδραστικά αντικείμενα ή περιβάλλοντα.

Με τον όρο πλατφόρμα ανοιχτού κώδικα, εννοούμε ότι το arduino έχει σχεδιαστεί με τις αρχές του ανοικτού κώδικα (Open Source Hardware), δηλαδή τα ηλεκτρονικά σχέδια είναι διαθέσιμα για οποιονδήποτε θέλει να κατασκευάσει ή ακόμα και να βελτιώσει την σχεδίαση του υλικού ή να βασιστεί για την δημιουργία κάποιου άλλου υλικού. Το λογισμικό των μικροελεγκτών, έχει διαθέσιμο τον πηγαίο κώδικα για να μπορεί επίσης να βελτιωθεί ή να επαναχρησιμοποιηθεί και εκείνο.

Το Arduino αποτελείται από δύο κύρια μέρη, την πλακέτα Arduino το οποίο είναι το κομμάτι του hardware πάνω στο οποίο εργάζεται ο κατασκευαστής όταν πραγματοποιεί μία κατασκευή ενώ το δεύτερο τμήμα είναι το Arduino IDE, το κομμάτι του λογισμικού που τρέχει στον υπολογιστή. Το IDE χρησιμοποιείται για να δημιουργηθεί ένα sketch (ένα μικρό πρόγραμμα στον υπολογιστή) που φορτώνεται στον μικροελεγκτή της πλακέτα Arduino. Το sketch λέει στην πλακέτα arduino τι πρέπει να κάνει. Ο χρήστης μπορεί να προγραμματίσει το arduino να δέχεται δεδομένα από μια πληθώρα αισθητηρίων (θερμοκρασίας, υγρασίας, φωτεινότητας, κίνησης, επιταχυνσιόμετρο και χιλιάδες άλλους αισθητήρες) στις θύρες εισόδου και στην συνέχεια να επεξεργάζεται αυτά τα δεδομένα και να επιδράει σε ότι το περιβάλλει χρησιμοποιώντας τις θύρες εξόδου, αναβοσβήνοντας φώτα ή αντιστάσεις θέρμανσης, ελέγχοντας κινητήρες, συσκευές παραγωγής ηχητικών σημάτων και άλλες συσκευές εξόδου.

1.1.1 Ιστορική αναδρομή

Το 2005 στην Ivrea της Ιταλίας κατασκευάζεται μία πρότυπη ηλεκτρονική πλατφόρμα ανοιχτού κώδικα στην αρχή για εκπαιδευτικούς σκοπούς, του τεχνικού πανεπιστημίου της πόλης, σε μια προσπάθεια να ελαχιστοποιήσουν το κόστος του εκπαιδευτικού υλικού που έπρεπε να αγοράσουν οι φοιτητές, και στην συνέχεια δόθηκε σε κυκλοφορία στο ευρύ κοινό. Το όνομα της συσκευής αυτής έχει τις ρίζες της από τον Arduin of Ivrea, έναν βασιλιά της Ιταλίας του ενάτου αιώνα όπου κατοικούσε στην ίδια πόλη. Η συσκευή ονομάστηκε "Arduino" που αντιστοιχούσε σε ένα ιταλικό ανδρικό όνομα και σήμαινε "ισχυρός φίλος". Η ομάδα που κατασκεύασε το Arduino αποτελείται από τους Massimo Banzi, David Cuartielles, Tom Igoe, David Mellis και Gianluca Martino.

Το arduino αναπτύχθηκε σύμφωνα με την πλατφόρμα Wiring, μία πτυχιακή εργασία του Hernando Barragan από το Interaction Design Institute Ivrea. Είχε ως στόχο να είναι μία ηλεκτρονική εκδοχή της Processing που θα χρησιμοποιούσε ένα περιβάλλον προγραμματισμού δικό της αλλά θα έμοιαζε σχεδιαστικά και συντακτικά με αυτό της Processing. Η Processing είναι μια γλώσσα προγραμματισμού ανοικτού κώδικα που βασίζεται στην Java.

1.2 Μικροελεγκτής ATmega 328

Ο συγκεκριμένος μικροελεγκτής είναι της εταιρείας ATMEL και είναι βασισμένος στην αρχιτεκτονική AVR. Η αρχιτεκτονική AVR βασίζεται σε μία τροποποίηση της αρχιτεκτονικής Harvard των 8 bit, RISC (Reduced Instruction Set Computing). Με βάση την αρχιτεκτονική Harvard το πρόγραμμα και τα δεδομένα είναι αποθηκευμένα σε διακριτά φυσικά συστήματα μνήμης τα οποία παρουσιάζονται σε διαφορετικές φυσικές διευθύνσεις μνήμης. Συνολικά διαθέτει 131 εντολές μήκους 16 ή 32 bit και 32 καταχωρητές των 8 bit.

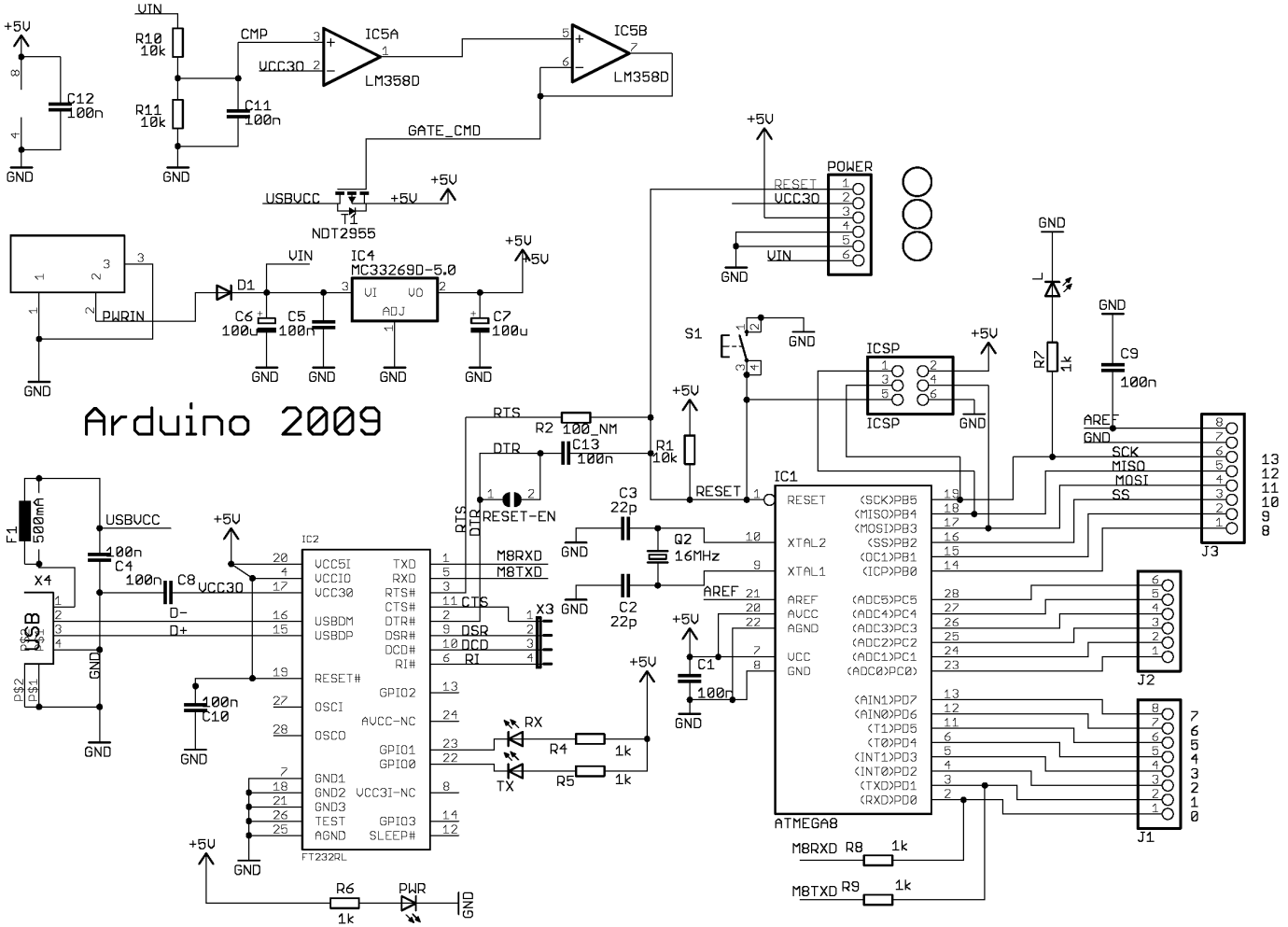
Το συγκεκριμένο μοντέλο μικροελεγκτή διαθέτει 32Kbytes μνήμης Flash (10000 κύκλων εγγραφής/διαγραφής), 2KBytes εσωτερικής μνήμης SRAM και 1Kbyte μνήμης EEPROM (100000 κύκλων εγγραφής/διαγραφής). Στη μνήμη Flash αποθηκεύεται το πρόγραμμα, στην SRAM δημιουργούνται και αποθηκεύονται οι τιμές των μεταβλητών κατά τη διάρκεια λειτουργίας του μικροελεγκτή ενώ η μνήμη EEPROM χρησιμοποιείται για την μόνιμη αποθήκευση δεδομένων καθώς είναι μη πτητική.

Μπορεί να χρονιστεί σε συχνότητα ρολογιού μέχρι τα 20 MHz ενώ λειτουργεί σε τάση τροφοδοσίας 4.5-5.5 V. Να σημειωθεί ότι όπως βρίσκεται ενσωματωμένος στην πλατφόρμα Arduino είναι χρονισμένος στη συχνότητα των 16 MHz μέσω εξωτερικού κρυσταλλικού ταλαντωτή. Διατίθεται σε συσκευασία Plastic Dual Inline Package (PDIP) των 28 ακροδεκτών και λειτουργεί σε θερμοκρασιακό εύρος (-40°C έως 85°C).

Διαθέτει 14 ψηφιακές εισόδους/εξόδους (0-5 V). Ακόμη διαθέτει 6 αναλογικές εισόδους οι οποίες επιλέγονται με πολυπλέκτη ο οποίος τις οδηγεί σε ένα μετατροπέα ADC διαδοχικής προσέγγισης με μέγιστη ανάλυση 10 bit, μέγιστο ρυθμό δειγματοληψίας 15kSPS και μέσο χρόνο μετατροπής 180 μs. Οι αναλογικές είσοδοι δέχονται τάσεις 0-5 V ενώ δίνεται η δυνατότητα να ορισθεί εξωτερικά το επίπεδο τάσης αναφοράς σε οποιαδήποτε τιμή μεταξύ 0 και 5 V με αποτέλεσμα να αυξάνεται η διακριτική ικανότητα.

1.3 Περιγραφή κυκλώματος – Μοντέλα

Η δημοφιλέστερη έκδοση του arduino είναι η Duemilanove/UNO (εικόνα 1) που βασίζεται στο ολοκληρωμένο ATmega328 και συμπληρωματικά εξαρτήματα για την διευκόλυνση του χρήστη στον προγραμματισμό και την ενσωμάτωση του σε άλλα κυκλώματα. Όλες οι πλακέτες περιλαμβάνουν ένα γραμμικό ρυθμιστή τάσης 5V και έναν κρυσταλλικό ταλαντωτή 16MHz. Ο μικροελεγκτής είναι από κατασκευής προγραμματισμένος με ένα bootloader, έτσι ώστε να μην χρειάζεται εξωτερικός προγραμματιστής.



Εικόνα 1: Arduino Duemilanove

1.4 Τρόποι προγραμματισμού

Ο μικροελεγκτής της πλακέτας είναι προγραμματισμένος έτσι ώστε να χρησιμοποιεί την γλώσσα προγραμματισμού του arduino που βασίζεται στην γλώσσα wiring (πρόκειται για τη C++ με κάποιες αλλαγές και απλοποιήσεις). Επίσης προγραμματίζεται μέσω υπολογιστή με την βοήθεια του arduino IDE (Integrated Development Environment – Ολοκληρωμένο Περιβάλλον ανάπτυξης). Το IDE είναι ένα ειδικό πρόγραμμα βασισμένο σε java που εκτελείται στον υπολογιστή και επιτρέπει να γραφούν τα sketches για την πλακέτα arduino σε μία απλή γλώσσα που διαμορφώθηκε μέσα από την γλώσσα Processing. Πατώντας ένα κουμπί, το πρόγραμμα φορτώνει το sketch στον μικροελεγκτή του arduino. Η χρήση της γλώσσας προγραμματισμού java για την δημιουργία του περιβάλλοντος προγραμματισμού δίνει την δυνατότητα στο IDE του arduino να μπορεί να εκτελεστεί σε όλα τα λειτουργικά συστήματα (όπως Windows, Unix, Linux, BSD, MacOS) το ίδιο, χωρίς να χρειάζεται μεταγλώττιση ή να αλλάξει ο πηγαίος κώδικας.

Οι εφαρμογές που γράφονται στο arduino μπορούν να λειτουργούν αυτόνομα ή μπορούν να επικοινωνούν με τον υπολογιστή μέσω προγραμμάτων όπως Flash, Processing, MaxMSP, Matlab – Simulink, Labview κ.α.

1.5 Wiring

Η Wiring είναι μια ανοιχτή πλατφόρμα ηλεκτρονικών πρωτοτύπων πηγή της οποίας αποτελεί μια γλώσσα προγραμματισμού, ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE), και έναν ενσωματωμένο μικροελεγκτή. Αναπτύχθηκε το 2003 από Hernando Barragán.

Ο Barragán ξεκίνησε το έργο του στο Interaction Design Institute Ivrea. Το πρόγραμμα αυτή τη στιγμή αναπτύσσεται στη Σχολή Αρχιτεκτονικής και Σχεδιασμού στο Universidad de Los Andes στη Μπογκοτά, Κολομβία.

Η Wiring βασίζεται στην Processing, ένα ανοιχτό έργο που ξεκίνησε από τους Casey Reas και Benjamin Fry που στο παρελθόν υπήρξαν μέλη του MIT Media Lab.

Το περιβάλλον ανάπτυξης (IDE) της Wiring είναι μια cross-platform εφαρμογή γραμμένη σε Java, η οποία προέρχεται από το περιβάλλον που είχε γίνει για τη γλώσσα προγραμματισμού Processing. Περιλαμβάνει ένα πρόγραμμα επεξεργασίας κώδικα με χαρακτηριστικά όπως είναι η επισήμανση σύνταξης και αυτόματη στοίχιση του κώδικα.

Το IDE της Wiring έρχεται με μια βιβλιοθήκη C / C + + που ονομάζεται "wiring", η οποία κάνει την χρήση εισόδου / εξόδου πολύ πιο εύκολη. Τα προγράμματα σε Wiring είναι γραμμένα σε C / C + +, αν και οι χρήστες το μόνο που χρειάζεται να ορίσουν είναι δύο functions (μεθόδους) για να υλοποιήσουν ένα εκτελέσιμο πρόγραμμα:

setup ( ) - μια μέθοδος που εκτελείται μία φορά κατά την έναρξη ενός προγράμματος, το οποίο μπορεί να χρησιμοποιηθεί για να καθορίσει τις αρχικές ρυθμίσεις περιβάλλοντος.

loop ( ) - μια μέθοδος που καλείται επανειλημμένα μέχρι η πλακέτα να απενεργοποιηθεί.

Ένα τυπικό πρώτο πρόγραμμα για έναν μικροελεγκτή είναι απλά να αναβοσβήνει ένα LED (δίοδος εκπομπής φωτός). Στο περιβάλλον Wiring, ο χρήστης μπορεί να γράψει ένα πρόγραμμα σαν αυτό:

```
int ledPin = WLED;                // όνομα LED

void setup () {
   pinMode(ledPin, OUTPUT);       // ορισμός του pin 48 ως έξοδος
}

void loop () {
   digitalWrite(ledPin, HIGH);    // άναμμα LED
   delay (1000);                  // αναμονή 1000 milliseconds
   digitalWrite(ledPin, LOW);     // σβήσιμο LED
   delay (1000);                  // αναμονή 1 sec
}
```

Όταν ο χρήστης κάνει κλικ στο "Upload to Wiring hardware" κουμπί στο IDE, ένα αντίγραφο του κώδικα είναι γραμμένο σε ένα προσωρινό αρχείο περιλαμβάνοντας μια

κεφαλίδα στην κορυφή και μια πολύ απλή συνάρτηση main () στο κάτω μέρος, για να το κάνει ένα έγκυρο C + + πρόγραμμα.

Το γραφικό περιβάλλον ανάπτυξης της Wiring χρησιμοποιεί το GNU toolchain και AVR libc για την μεταγλώτιση των προγραμμάτων, και χρησιμοποιεί avrdude για την φόρτωση του προγράμματος στην πλακέτα.

Για την επικοινωνία της πλακέτας με τον υπολογιστή υπάρχουν συγκεκριμένες εντολές σειριακής επικοινωνίας. Χρησιμοποιούνται για την επικοινωνία μεταξύ της πλακέτας Arduino και έναν υπολογιστή ή σε άλλες συσκευές. Όλες οι πλακέτες Arduino έχουν τουλάχιστον μία σειριακή θύρα (επίσης γνωστή ως ένα UART ή USART): Serial. Η θύρα αυτή επικοινωνεί μέσω των ψηφιακών pins 0 (RX) και 1 (TX), καθώς και με τον υπολογιστή μέσω USB. Έτσι, εάν χρησιμοποιούνται αυτές οι λειτουργίες, δεν μπορούν να χρησιμοποιηθούν τα 0 και 1 pins για ψηφιακή είσοδο ή έξοδο.

Κάποιες από τις εντολές οι οποίες είναι χρήσιμες για την σειριακή επικοινωνία είναι οι εξής:

serial.begin(speed) : Εκκινεί μια σειριακή επικοινωνία με ταχύτητα που ορίζεται στην παράμετρο speed.

serial.end( ) : Λήγει μια σειριακή επικοινωνία.

serial.read( ) : Γίνεται η ανάγνωση ενός στοιχείου από τον buffer της σειριακής.

serial.write(val) : Γίνεται η καταγραφή της τιμής val στον buffer της σειριακής για αποστολή.

serial.available( ) : Επιστρέφει το πλήθος των bytes που είναι αποθηκευμένα προς ανάγνωση στον buffer της σειριακής.

1.6 PID και η βιβλιοθήκη του Arduino

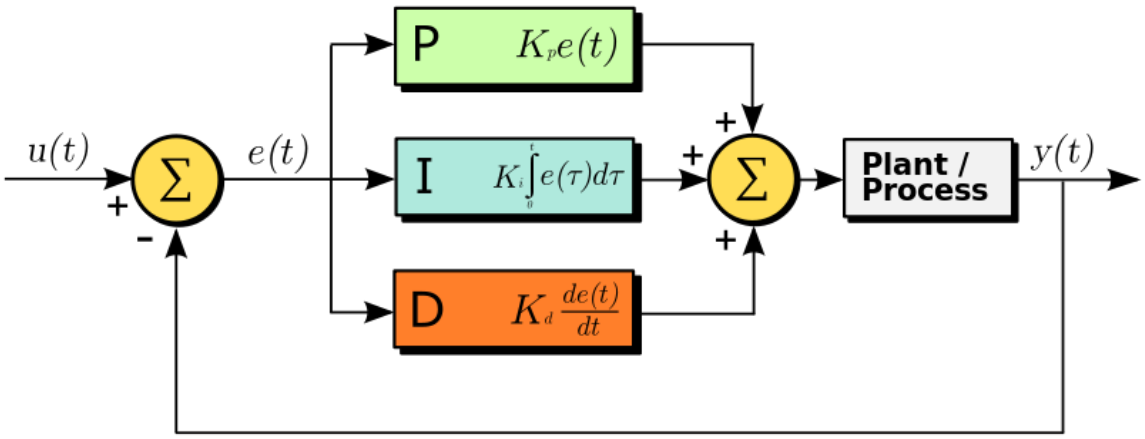
Ένας PID ελεγκτής υπολογίζει ένα «σφάλμα» ως την διαφορά μεταξύ μιας μετρημένης τιμής και ενός επιθυμητού σημείου. Ο ελεγκτής προσπαθεί να ελαχιστοποιήσει το σφάλμα ρύθμισης.

Έτσι, για τον ελεγκτή PID χρειάζεται η μέτρηση (είσοδος - input), η επιθυμητή τιμή (setpoint) και η μεταβλητή προσαρμογής (έξοδος - output). Ο ελεγκτής PID τότε

ρυθμίζει την μεταβλητή προσαρμογής προσπαθώντας να κάνει την είσοδο να ισούται με την επιθυμητή τιμή.

Η ιδιαιτερότητα του ελεγκτή PID σε σχέση με τους υπόλοιπους ελεγκτές είναι πως υπολογίζει την διαφορά – σφάλμα , την παράγωγο του σφάλματος και το ολοκλήρωμα αυτού και εξάγει αποτέλεσμα για την τιμή που θα πρέπει να πάρει η έξοδος μέσω του αθροίσματος με βάρη των παραπάνω όπως φαίνεται στην παρακάτω σχέση που αποτελεί την συνάρτηση εξόδου του ελεγκτή:

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \qquad (1.1)$$



Σχήμα 1.1 – Ελεγκτής PID

Το Arduino περιλαμβάνει βιβλιοθήκη PID η οποία είναι εύκολη στην χρήση. Πολύς χρόνος δαπανήθηκε από τους σχεδιαστές της βιβλιοθήκης, βετιστοποιώντας τον αλγόριθμο σε αυτήν τη βιβλιοθήκη κάνοντάς τον συκρίσιμο με αυτούς που χρησιμοποιούνται στη βιομηχανία.

Ανάλυση εντολών της βιβλιοθήκης.

PID (&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction) : Δημιουργεί έναν ελεγκτή PID με τις συγκεκριμένες μεταβλητές ως είσοδο, έξοδο και εισάγοντας τις τιμές Kp, Ki και Kd ως τις τιμές του αναλογικού, ολοκληρωτικού και διαφορικού κέρδους αντίστοιχα. Η παράμετρος Direction μπορεί να πάρει τις τιμές 'DIRECT' και 'REVERT' αν το μέγεθος εξόδου αυξάνεται ανάλογα από το μέγεθος εισόδου ή αντιστρόφως ανάλογα αντίστοιχα.

Compute() : Υπολογίζει την τιμή της εξόδου του PID. Στην πράξη περιέχει όλο τον αλγόριθμο του PID ελεγκτή. Ως είσοδο και έξοδο λαμβάνονται οι μεταβλητές που είχαν ορισθεί κατά την δημιουργία του ελεγκτή.

SetMode (mode): Καθορίζει αν ο ελεγκτής θα είναι αυτόματος ή όχι.

SetOutputLimits(min, max) : Καθορίζει τα όρια της εξόδου του ελεγκτή. Η προκαθορισμένη τιμή είναι από 0 ως 255.

SetTunings(Kp, Ki, Kd) : Καθορίζει τις τιμές των κερδών: αναλογικού, ολοκληρωτικού και διαφορικού αντίστοιχα.

SetSampleTime(SampleTime) : Καθορίζει την τιμές της περιόδου της δειγματοληψίας για τον ελεγκτή PID, σε ms. Η προκαθορισμένη τιμή είναι 200ms.

SetControllerDirection(Direction) : Αλλάζει την φορά του ελέγχου. Η φορά του ελέγχου αναλύεται στην εντολή PID().

Display Functions : Η βιβλιοθήκη αυτή περιέχει και εντολές για την ανάγνωση των παραμέτρων. Αυτές είναι οι εξής: GetKp, GetKi, GetKd, GetMode, GetDirection.

ΚΕΦΑΛΑΙΟ 2
Αναγνώριση και Έλεγχος Συστημάτων με την βοήθεια του Matlab

2.1 Εισαγωγή

Για την επικοινωνία του υπολογιστή με την πλατφόρμα Arduino επιλέχθηκε το πακέτο λογισμικού MATLAB, παρ' ότι υπήρχε ήδη έτοιμο ένα γραφικό περιβάλλον για επικοινωνία του υπολογιστή με την PID βιβλιοθήκη του arduino γραμμένο σε processing με την ονομασία PID Front-End.

Η επιλογή του MATLAB έγινε επειδή είναι το βασικό εργαλείο που έπρεπε να χρησιμοποιήσουμε για την αναγνώριση της συνάρτησης μεταφοράς του συστήματος μας και την εύρεση των συντελεστών Kp, Ki και Kd του PID ελεγκτή.

2.2 Το MATLAB ως εργαλείο ελέγχου

Το MATLAB είναι μια γλώσσα προγραμματισμού που έχει σχεδιαστεί ειδικά για το χειρισμό πινάκων. Λόγω της πληθώρας αλγορίθμων που περιέχει και της ευχρηστίας του, το MATLAB είναι ένα εργαλείο που επιλέγεται από πολλούς μηχανικούς ελέγχου για το σχεδιασμό και την προσομοίωση συστημάτων ελέγχου. Το MATLAB έχει μια σειρά από εργαλειοθήκες που ονομάζονται "Toolbox". Σχεδόν όλες οι λειτουργίες που περιγράφονται πιο κάτω βρίσκονται στην εργαλειοθήκη ελέγχου (Control Toolbox).

Απομόνωση Εισόδου-εξόδου

Σε ένα σύστημα MIMO (Multiple Inputs, Multiple Outputs), τυπικά μπορεί να είναι σημαντικό να απομονωθεί ένα μοναδικό ζεύγος εισόδου-εξόδου, για ανάλυση. Κάθε είσοδος αντιστοιχεί σε μία μόνο σειρά στην μήτρα B, και κάθε έξοδος αντιστοιχεί σε μία μόνο στήλη στην C μήτρα. Για παράδειγμα, για να απομονώσει την 2η είσοδο και την 3η έξοδο, μπορούμε να δημιουργήσουμε ένα σύστημα:

```
sys = ss(A, B(:,2), C(3,:), D);
```

Βηματική απόκριση

Κατ 'αρχάς, ας ρίξουμε μια ματιά στην κλασική προσέγγιση, με το ακόλουθο σύστημα:

$$G(s) = \frac{5s + 10}{s^2 + 4s + 5} \tag{2.1}$$

Αυτό το σύστημα μπορεί αποτελεσματικά να μοντελοποιηθεί ως δύο διανύσματα των συντελεστών, NUM (Αριθμητής) και DEN (Παρονομαστής):

NUM = [5, 10]
DEN = [1, 4, 5]

Τώρα, μπορούμε να χρησιμοποιήσουμε την εντολή step του MATLAB για να παράγει την βηματική απόκριση σε αυτό το σύστημα:

```
step(NUM, DEN, t);
```

Όπου το t είναι ένα διάνυσμα χρόνου. Εάν δεν υπάρχουν αποτελέσματα για την αριστερή πλευρά που παρέχονται από εσάς, η συνάρτηση step θα παράγει αυτόματα μία γραφική παράσταση της απόκρισης βήματος. Αν, όμως χρησιμοποιήσουμε την ακόλουθη μορφή:

```
[y, x, t] = step(NUM, DEN, t);
```

Στη συνέχεια, το MATLAB δεν θα παράγει αυτόματα ένα διάγραμμα, αλλά θα πρέπει να κατασκευαστεί από εμάς.

Τώρα, ας ρίξουμε μια ματιά σε μια πιο σύγχρονη προσέγγιση στα συστήματα αυτομάτου ελέγχου μέσω περιγραφών στο χώρο των καταστάσεων. Αν έχουμε τις

μήτρες A, B, C και D, μπορούμε να συνδέσουμε αυτές στη συνάρτηση step, όπως φαίνεται:

```
step(A, B, C, D);
```

ή, μπορεί προαιρετικά να περιλαμβάνει ένα διάνυσμα για το χρόνο, t:

```
step(A, B, C, D, t);
```

Επίσης μπορούμε να καλέσουμε την συνάρτηση όχι για να παράγουμε ένα γράφημα, αλλά για να πάρουμε τα δεδομένα της απόκρισης μέσω της εντολής :

```
[y, x, t] = step(NUM, DEN, t);
```

Όπου y είναι το πλάτος της εξόδου της βηματικής απόκρισης, ενώ το χ είναι η εσωτερική κατάσταση του συστήματος από τις εξισώσεις κατάστασης:

$$\begin{aligned} x' &= Ax + Bu \\ y &= Cx + Du \end{aligned} \qquad (2.2)$$

Κλασική ↔ Σύγχρονη

Το MATLAB περιλαμβάνει χαρακτηριστικά που μπορεί να χρησιμοποιηθούν για να γίνει η αυτόματη μετατροπή στην αναπαράσταση χώρου κατάστασης από την συνάρτηση μεταφοράς. Αυτή η συνάρτηση, tf2ss, χρησιμοποιείται ως εξής:

```
[A, B, C, D] = tf2ss(NUM, DEN);
```

Όπου NUM και DEN είναι τα διανύσματα συντελεστή του αριθμητή και του παρονομαστή της συνάρτησης μεταφοράς, αντίστοιχα.

Στο ίδιο πνεύμα, μπορούμε να μετατρέψουμε αντίστροφα από μια συνάρτηση μεταφοράς στον χώρο καταστάσεων χρησιμοποιώντας τη λειτουργία ss2tf, ως εξής:

```
[NUM, DEN] = ss2tf(A, B, C, D);
```

Ή, αν έχουμε περισσότερες από μία εισόδους σε ένα διάνυσμα u, μπορούμε να το γράψουμε ως εξής:

```
[NUM, DEN] = ss2tf(A, B, C, D, u);
```

Γεωμετρικός τόπος ριζών

Το MATLAB παρέχει ένα χρήσιμο, αυτόματο εργαλείο για τη δημιουργία του γραφήματος γεωμετρικού τόπου ριζών από μια συνάρτηση μεταφοράς, την εντολή rlocus. Έτσι έχουμε τις ακόλουθη χρήση της συνάρτησης:

```
rlocus(num, den);
```

ή:

```
rlocus(A, B, C, D);
```

Όπου οι μεταβλητές num και den αποτελούν τον αριθμητή και παρονομαστή της συνάρτησης μεταφοράς αντίστοιχα, ενώ οι A,B,C και D είναι οι πίνακες στον χώρο των καταστάσεων.

Διάγραμμα Bode

Το MATLAB προσφέρει επίσης μια σειρά από εργαλεία για την εξέταση των χαρακτηριστικών απόκριση συχνότητας ενός συστήματος, τόσο με τη χρήση διαγραμμάτων Bode, αλλά και χρησιμοποιώντας διαγράμματα Nyquist. Για την κατασκευή ενός διαγράμματος Bode από μια συνάρτηση μεταφοράς, χρησιμοποιούμε την ακόλουθη εντολή:

```
[mag, phase, omega] = bode(NUM, DEN, omega);
```

ή:

```
[mag, phase, omega] = bode(A, B, C, D, u, omega);
```

Όπου "omega" είναι το διάνυσμα συχνοτήτων όπου αναλύονται τα σημεία απόκρισης πλάτους και φάσεως.

Σύνδεση συστημάτων και ανάδραση

Το MATLAB προσφέρει εύκολο τρόπο για την σύνδεση των συστημάτων μεταξύ τους. Για να συνδέσουμε δύο συστήματα G1 και G2 μεταξύ τους, απλά εκτελούμε την παρακάτω εντολή, καθώς παράγεται το συνολικό σύστημα G:

```
G = G1 * G2 ;
```

Όπου G, G1 και G2 είναι συναρτήσεις μεταφοράς τύπου Matlab.

Για να συνδέσουμε ένα σύστημα F1 ως ανάδραση με το σύστημα G1 τότε εκτελούμε την εντολή feedback με την οποία παράγεται το συνολικό σύστημα G. Ο τρόπος κλήσης της συνάρτησης είναι αυτός (για αρνητική ανάδραση) :

```
G = feedback( G1 , F1 , -1 ) ;
```

Προσομοίωση γραμμικών συστημάτων

Η συνάρτηση **lsim** προσομοιώνει την απόκριση των συνεχών ή διακριτών γραμμικών συστημάτων για αυθαίρετες εισόδους. Όταν καλείται χωρίς αριστερό όρισμα (αποτέλεσμα), γίνεται εμφάνιση του διαγράμματος στην οθόνη.

Η εξής κλήση:

```
lsim( sys , u , t ) ;
```

παράγει μια καμπύλη της απόκρισης του δυναμικού συστήματος στο χρόνο t, για είσοδο u. Το διάνυσμα t καθορίζει τα δείγματα του χρόνου για την προσομοίωση (σε μονάδες χρόνος του συστήματος, που καθορίζεται στην ιδιότητα του TIMEUNIT sys). Για παράδειγμα:

```
t = 0 : dt : Tfinal;
```

Η μήτρα u πρέπει να έχει τόσες σειρές όσο και τα δείγματα του χρόνου (διάρκεια (t)) και τόσες στήλες όσες οι είσοδοι του συστήματος. Κάθε γραμμή u (i, :) καθορίζει την τιμή εισόδων στο δείγμα χρόνου t (i). Στην συγκεκριμένη εργασία όπου έχουμε συστήματα SISO (Single Input, Single Output) τα ορίσματα u και t αποτελούν και τα δύο διανύσματα.

2.3 Γραφικά περιβάλλοντα διεπαφής (GUI)

Το Matlab προσφέρει στο χρήστη τη δυνατότητα να κατασκευάσει δικές του γραφικές διεπαφές (GUI – Graphical User Interface). Η χρησιμότητα της λειτουργίας αυτής είναι μεγάλη, επειδή τα προγράμματα – εφαρμογές που περιέχουν γραφικό περιβάλλον γίνονται πιο φιλικά προς τον τελικό χρήστη.

Το Matlab προσφέρει μια ικανοποιητική εργαλειοθήκη, η οποία διευκολύνει πολύ τη δημιουργία ενός GUI. Αυτή η εργαλειοθήκη που ονομάζεται GUIDE (Graphical User Interface Design Environment), περιέχει πληθώρα εργαλείων ελέγχου όπως κουμπιά,

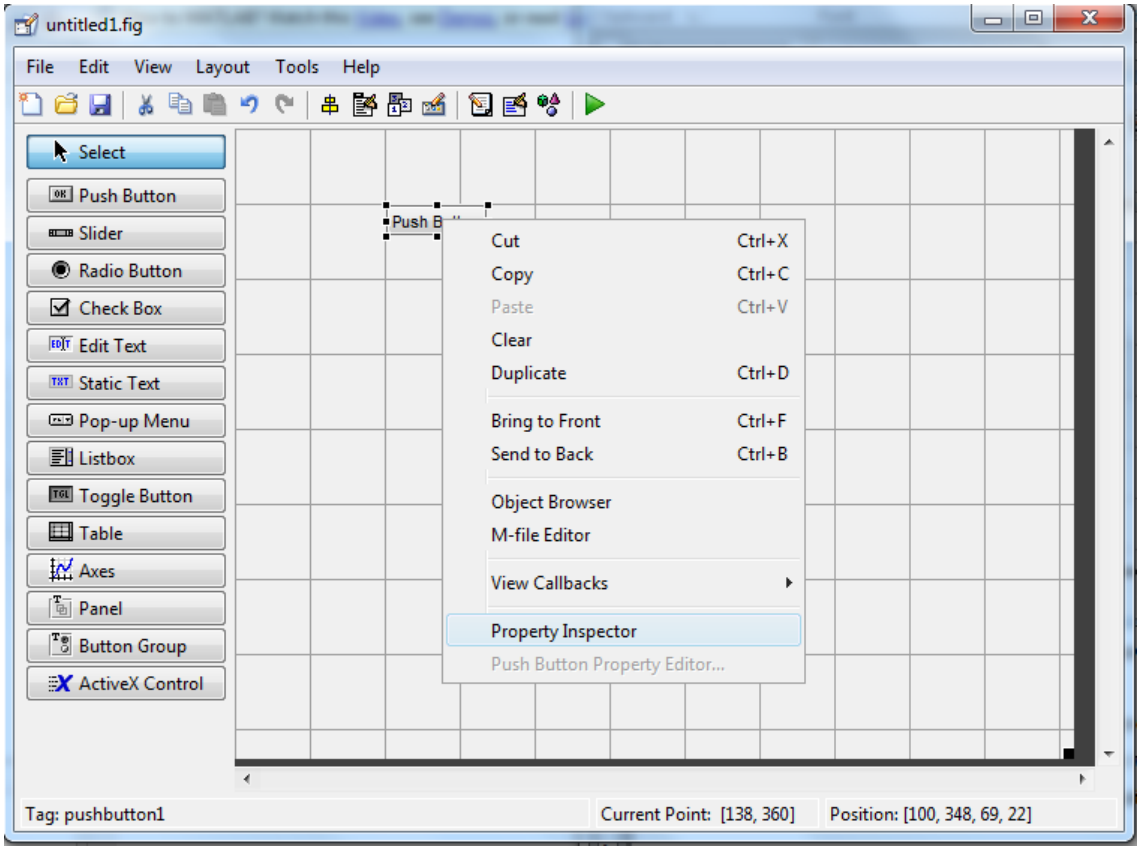
πλαίσια κ.α. Κατα την προγραμματισμό ενός GUI δημιουργούνται δύο αρχεία, ένα Fig-file και ένα M-file.

Το Fig-file ουσιαστικά είναι το παράθυρο-figure, όπου το Matlab αποθηκεύει τα στοιχεία ελέγχου και την ακριβή θέση τους. Εδώ ο προγραμματιστής σχεδιάζει την εμφάνιση του παραθύρου.

Στο M-file ο προγραμματιστής πρέπει να γράψει τον κώδικα που θα ενσωματωθεί στα στοιχεία ελέγχου (π.χ. κουμπιά, φόρμες εισαγωγής δεδομένων, γραφικές παραστάσεις, κ.α.), ώστε αυτά να επιτελούν τις επιθυμητές λειτουργίες.

Πρέπει να σημειωθεί ότι το κάθε αρχείο *.fig πρέπει να συνοδεύεται από το αντίστοιχο αρχείο *.m (με το ίδιο όνομα) για να είναι λειτουργικό.

Η εκκίνηση του GUI γίνεται με την εντολή guide, η αλλιώς ακλουθώντας την διαδρομή Start > MATLAB > GUIDE (GUI Builder) μέσα από το περιβάλλον του προγράμματος.



Σχήμα 2.1 – Προγραμματιστικό Περιβάλλον GUI

Το αρχικό παράθυρο κατασκευής ενός GUI φαίνεται στο σχήμα 2.1, στα δεξιά της εικόνας βρίσκεται η εργαλειοθήκη που περιέχει τα στοιχεία ελέγχου, τα οποία μπορείς να τα μεταφέρεις με το ποντίκι στο πεδίο σχεδιασμού (layout area) του GUI, στη

συνέχεια πατώντας το δεξί πλήκτρο του ποντικιού επάνω στο αντικείμενο και επιλέγοντας "Property Inspector" μπορούμε να αλλάξουμε κάποιες ιδιότητες του αντικειμένου που επιλέξαμε. Κάθε αλλαγή που κάνουμε προσθέτει έναν κώδικα στο αρχείο m-file που συνδέεται με το gui, για παράδειγμα η εισαγωγή του pushbutton στο πεδίο σχεδιασμού προσθέτει αυτόματα στο m-file την συνάρτηση:

```
function pushbutton1_Callback(hObject, eventdata, handles
```

στην οποία ενσωματώνουμε τις εντολές που θέλουμε να εκτελεί το κουμπί αφού το πατήσουμε. Με τον ίδιο τρόπο αναπτύχθηκε ο κώδικας που βρίσκεται στο ΠΑΡΑΡΤΗΜΑ Γ, επίσης στο ΠΑΡΑΡΤΗΜΑ Δ υπάρχουν εικόνες από το περιβάλλον διαμόρφωσης του GUI που δημιουργήθηκε.

2.4 Μέθοδος Αναγνώρισης Συστήματος (Identification Toolbox)

Μία από τις εργαλειοθήκες που χρειάστηκε για να μπορέσουμε να προσδιορίσουμε την συνάρτηση μεταφοράς των συστημάτων μας είναι η εργαλειοθήκη αναγνώρισης συστήματος (system identification tool). Η είσοδος στο περιβάλλον της εργαλειοθήκης γίνεται με την εντολή ident ή ακολουθώντας την διαδρομή Start > Toolboxes > More… > System Indetification > System Indetification tool από το περιβάλλον του προγράμματος.

Ως αναγνώριση ορίζεται η διαδικασία κατασκευής του μαθηματικού μοντέλου ενός συστήματος βασισμένη σε δεδομένα μετρήσεων. Αυτό κατά βάση επιτυγχάνεται με τη ρύθμιση παραμέτρων ενός δεδομένου τύπου μοντέλου μέχρι η απόκριση του τελικά να συμπίπτει όσο το δυνατόν καλύτερα με την έξοδο που μετρήθηκε. Όσο περισσότερο συμπίπτει τόσο καλύτερο είναι το μοντέλο μας. Τα δημοφιλέστερα μοντέλα αφορούν περιγραφές διαφορικών εξισώσεων ενώ άλλα χρησιμοποιούν γραμμικά συστήματα στο χώρο κατάστασης. Σε κάθε περίπτωση χρειαζόμαστε ένα κατάλληλο τύπο μοντέλου που θα περιέχει πληροφορίες για την τάξη του μοντέλου (πλήθος πόλων και μηδενικών) καθώς και για τις όποιες χρονικές καθυστερήσεις αυτό εμφανίζει στην απόκριση του. Οι αλγόριθμοι αναγνώρισης εκτελούνται εύκολα μέσω του πακέτου του Matlab το οποίο περιέχει όλες τις γνωστές τεχνικές ρύθμισης παραμέτρων για όλα τα είδη γραμμικών μοντέλων. Επίσης μας επιτρέπει την εποπτεία των ιδιοτήτων του μοντέλου και τον έλεγχο της αξιοπιστίας τους όσο αφορά το εξαγόμενο μοντέλο και την επεξεργασία και ομαλοποίηση των δεδομένων προς ανάλυση και χρήση από τους

προσεγγιστικούς αλγόριθμους. Ο μόνος περιορισμός είναι ότι οι μέθοδοι αυτές απαιτούν τη χρήση γραμμικών συστημάτων ή γραμμικών περιοχών λειτουργίας μη γραμμικών συστημάτων για την επιτυχή εξαγωγή αξιόπιστων αποτελεσμάτων.

Η διαδικασία αναγνώρισης ενός δυναμικού συστήματος από την παρατήρηση εισόδου – εξόδου απαιτεί τρία βασικά συστατικά:

- Τα δεδομένα εισόδου και εξόδου σε επαρκές πλήθος ανάλογα την εφαρμογή.
- Κάποια υποψήφια μοντέλα ή πιθανά είδη συναρτήσεων μεταφοράς.
- Κριτήρια επιλογής τελικού μοντέλου και αξιοπιστίας του.

Η διαδικασία αναγνώρισης απαιτεί την συνεχή αλλαγή δομής του υποψήφιου μοντέλου για την επιλογή του βέλτιστου καθώς και συνεχή εκτίμηση των ιδιοτήτων του. Συνοψίζεται στα εξής γενικά βήματα:

1. Σχεδιασμός του πειράματος και συλλογή δεδομένων εισόδου – εξόδου σε αριθμητικά ζεύγη τιμών προς επεξεργασία.
2. Μελέτη των δεδομένων, τροποποίηση τους ώστε να εξομαλυνθούν ακραίες τιμές και επιλογή κατάλληλων περιοχών δεδομένων προς επεξεργασία όπως π.χ. φιλτράρισμα για ενίσχυση τιμών γύρω από τις συχνότητες που παρουσιάζουν περισσότερο ενδιαφέρον.
3. Επιλογή και ορισμός της γενικής δομής κάποιων μοντέλων μέσα στα οποία περιλαμβάνεται πιθανότατα το τελικά αποδεκτό.
4. Υπολογισμός του βέλτιστου μοντέλου κρίνοντας το ανάλογα με τη συμφωνία που παρουσιάζει με τη μετρούμενη απόκριση.
5. Εξέταση των ιδιοτήτων του μοντέλου που εξήχθη από τον αλγόριθμο.
6. Εάν το μοντέλο είναι αρκετά αξιόπιστο και ικανοποιεί την ακρίβεια μας, τότε σταματάμε. Εάν όχι επιστρέφουμε στο $3^{ο}$ βήμα και συνεχίζουμε τη διερεύνηση.

Το πακέτο Matlab περιέχει εργαλεία για την επιτέλεση κάθε επιμέρους διαδικασίας που προαναφέρθηκε. Στα εργαλεία αυτά συμπεριλαμβάνονται ρουτίνες σχεδίασης δεδομένων σε άξονες, φιλτράρισμα δεδομένων, αποκοπής ακραίων τιμών και επανακατασκευής χαμένων δεδομένων. Τα παραπάνω κρίνονται αναγκαία και απαραίτητα για την περάτωση του δεύτερου βήματος της παραπάνω διαδικασίας. Υπάρχουν δε εντολές που υπολογίζουν έτοιμα μη παραμετρικά μοντέλα σε συνεχή και

διακριτό χρόνο και διευκολύνουν το τέταρτο βήμα της διαδικασίας. Τέλος, όσον αφορά τον έλεγχο αξιοπιστίας ενός μοντέλου των τελευταίων βημάτων, υπάρχουν ρουτίνες παρουσίασης στο πεδίο της συχνότητας, σε μορφή πόλων – μηδενικών και γενικής εξομοίωσης του μοντέλου που συντελούν στην πρόβλεψη της συμπεριφοράς του επίσης οι ρουτίνες μετασχηματισμών από συνεχή σε διακριτό χρόνο και αντίστροφα είναι πολύ διαδεδομένες και πολύ χρήσιμες.

Παρακάτω ακολουθεί μια σύντομη περιγραφή των μοντέλων της εργαλειοθήκης.

2.4.1 Γραμμικά παραμετρικά μοντέλα (Linear parametric model)

Οι τύποι των γραμμικών μοντέλων που θα μελετηθούν στην εργασία είναι οι ARX, ARMAX, και OE.

Το **ARX** (AutoRegresive model with eXternal input) αποτελεί ένα μοντέλο τύπου:

$$\begin{aligned} &y(t) + a_1 y(t-1) + \ldots + a_{na} y(t-na) = \\ &b_1 u(t-nk) + \ldots + b_{nb} u(t-nk-nb+1) \end{aligned} \tag{2.4}$$

Όπου:

y: η έξοδος.

u: η είσοδος.

na: το πλήθος των προηγούμενων καταστάσεων της εξόδου, δηλαδή το πλήθος των πόλων του συστήματος.

nb: το πλήθος των συντελεστών b, άρα και το πλήθος των μηδενικών συν ένα.

nk: η καθυστέρηση. Ο αριθμός δηλαδή των εισόδων πριν από την στιγμή t που επηρεάζουν την έξοδο.

Στις περισσότερες περιπτώσεις το nk είναι 1, δηλαδή δεν υπάρχει καθυστέρηση.

Το **ARMAX** (AutoRegresive model with eXternal input) αποτελεί ένα μοντέλο ψευδο-μη-γραμμικού τύπου, ενώ υποθέτουμε πως έχουμε στοχαστικό θόρυβο. Η συνάρτηση που το εκφράζει στον χρόνο είναι:

$$y(t)+a_1y(t-1)+\ldots+a_{na}y(t-na)=b_1u(t-nk)+\ldots \\ +b_{nb}u(t-nk-nb+1)+c_1e(t-nk)+\ldots+c_{nc}e(t-nk-nc+1) \quad (2.5)$$

Όπου:

y: η έξοδος.

u: η είσοδος.

e: θόρυβος

na: το πλήθος των προηγούμενων καταστάσεων της εξόδου, δηλαδή το πλήθος των πόλων του συστήματος.

nb: το πλήθος των συντελεστών b, άρα και το πλήθος των μηδενικών συν ένα.

nc: το πλήθος των συντελεστών c, άρα και το πλήθος των συντελεστώ που επηρεάζει ο θόρυβος.

nk: η καθυστέρηση. Ο αριθμός δηλαδή των εισόδων πριν από την στιγμή t που επηρεάζουν την έξοδο.

Το **OE** (Output Error model) αποτελεί ένα μοντέλο πολύ κοντινό με το ARMAX αφού πάλι υποθέτουμε πως έχουμε στοχαστικό θόρυβο, αλλά οι συντελεστές αυτή τη φορά δεν είναι ξεχωριστοί για τον θόρυβο αλλά είναι οι συντελεστές της εξόδου (εκεί προστίθεται ο θόρυβος). Η σχέση στον χρόνο που το εκφράζει είναι:

$$y(t)+a_1y(t-1)+\ldots+a_{na}y(t-na)= \\ =b_1u(t-nk)+\ldots+b_{nb}u(t-nk-nb+1)+a_1e(t)+\ldots+a_{na}e(t-na) \quad (2.6)$$

2.4.2 Μη – γραμμικά μοντέλα (non-Linear model)

Οι τύποι των μη-γραμμικών μοντέλων που θα μελετηθούν στην εργασία είναι οι ARX και Hammerstein - Wiener.

Το ARX μοντέλο μπορούμε να πούμε πως είναι μια προέκταση του γραμμικού μοντέλου ARX, μόνο που αντί να έχουμε το άθροισμα με βάρη των εισόδων – εξόδων, έχουμε μια μη γραμμική συνάρτηση που τα συνδέει. Έτσι η επιλογή που μπορούμε να

κάνουμε για το μοντέλο είναι ο αριθμός των προηγούμενων εισόδων εξόδων που επηρεάζουν την έξοδο. Η μαθηματική συνάρτηση του χρόνου που το εκφράζει είναι η παρακάτω:

$$y(t) = f(y(t-1),...,y(t-na),u(t),u(t-1),...,u(t-nb)) \qquad (2.7)$$

Όπου:

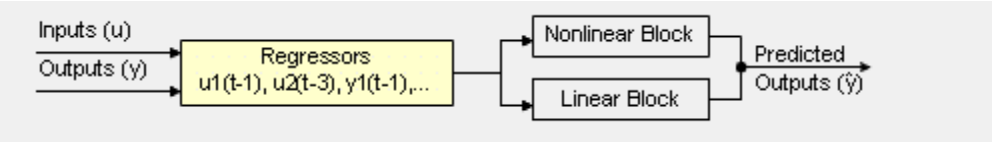
y: η έξοδος.

u: η είσοδος.

na: το πλήθος των προηγούμενων καταστάσεων της εξόδου.

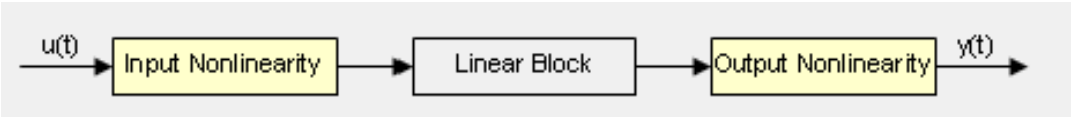
nb: το πλήθος των προηγούμενων καταστάσεων της εισόδου.

Η συνάρτηση f(…) είναι μη-γραμμική.



Σχήμα 2.2 – Μοντέλο nonlinear ARX

Το μοντέλο Hammerstein – Wiener αποτελεί ακόμα ένα μη γραμμικό μοντέλο που μελετάτε σε αυτή την εργασία. Ο τύπος του μοντέλου αυτού είναι αυτός που φαίνεται στον σχήμα 2.3. Δηλαδή περιέχει ένα γραμμικό κομμάτι και δύο μη γραμμικά υποσυστήματα, αυτό της εισόδου και αυτό της εξόδου. Τα τρία αυτά συστήματα τοποθετούνται σε σειρά.



Σχήμα 2.3 – Μοντέλο nonlinear Hammerstein - Wiener

2.4.3 Διαδικαστικά μοντέλα (process model)

Η δομή ενός διαδικαστικού μοντέλου είναι μια απλή συνεχή συνάρτηση μεταφοράς στον χρόνο που περιγράφει ένα γραμμικό δυναμικό σύστημα μέσω ενός ή περισσότερων από τα ακόλουθα στοιχεία:

Στατικό κέρδος Kp.

Μια ή περισσότερες σταθερές χρόνου Tpk. Για μιγαδικούς πόλους, η σταθερά χρόνου ονομάζεται Tω (ίση με το αντίστροφο της φυσικής συχνότητας) και ο συντελεστής απόσβεσης είναι το γνωστό ζ.

Μηδενική διαδικασία Tz.

Πιθανή χρονική καθυστέρηση Td πριν από την στιγμή που η έξοδος του συστήματος ανταποκρίνεται στην είσοδο (νεκρός χρόνος).

Πιθανή επιβολή ολοκλήρωσης (ολοκληρωτής).

Τα διαδικαστικά μοντέλα είναι δημοφιλή για την περιγραφή των δυναμικών συστημάτων σε βιομηχανίες και εφαρμόζονται σε διάφορα περιβάλλοντα παραγωγής. Τα πλεονεκτήματα αυτών των μοντέλων είναι ότι είναι απλά, υποστηρίζουν εκτιμήσεις της καθυστέρησης μεταφοράς και οι συντελεστές μοντέλου έχουν μια εύκολη ερμηνεία ως πόλους και μηδενικά.

Έτσι, μπορούμε να δημιουργήσουμε διαφορετικές δομές μοντέλων με τη μεταβολή του αριθμού των πόλων, προσθέτοντας ένα ολοκληρωτή, ή την προσθήκη ή αφαίρεση μια χρονική καθυστέρηση ή ενός μηδενικού. Μπορούμε, επίσης, να καθορίσουμε μια πρώτης, δεύτερης, ή τρίτης τάξης μοντέλο, και οι πόλοι μπορεί να είναι πραγματικοί ή μιγαδικοί.

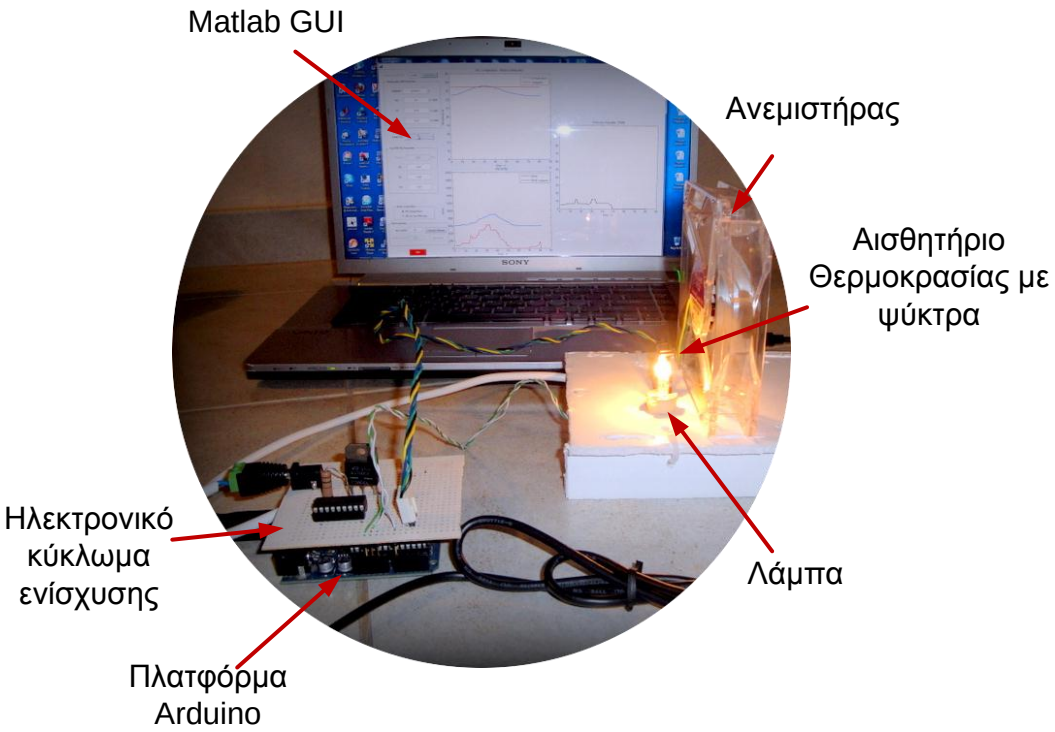
Για παράδειγμα, η ακόλουθη δομή είναι ένα μοντέλο πρώτης τάξεως συνεχούς χρόνου μοντέλο διαδικασίας, όπου το K είναι το στατικό κέρδος. Το T_{p1} είναι μια σταθερά χρόνου, και το T_d είναι η καθυστέρηση είσοδος-προς-εξόδου:

$$G(s) = \frac{K_p}{1 + sT_{p1}} e^{-8T_d} \tag{2.8}$$

ΚΕΦΑΛΑΙΟ 3
Η κατασκευή

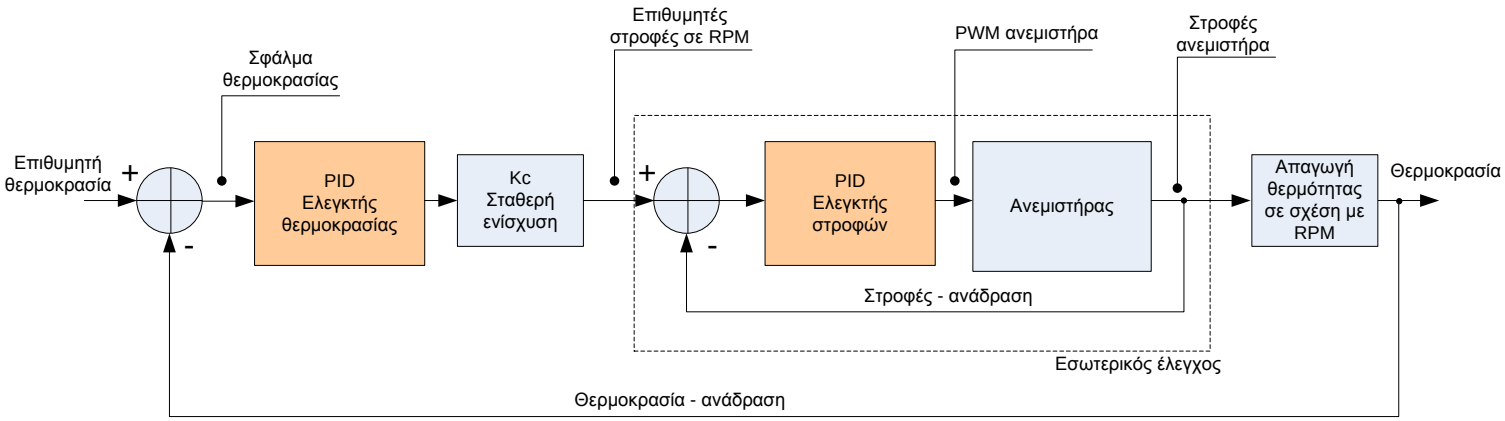
3.1 Εισαγωγή

Ο στόχος της εργασίας είναι η μελέτη των ελεγκτών PID (αναλογικού, ολοκληρωτικού και διαφορικού ελέγχου) για έλεγχο στροφών ανεμιστήρα και για έλεγχο θερμοκρασίας μέσω της πλακέτας Arduino. Έτσι ήταν αναγκαία η υλοποίηση μιας κατασκευής για την ανάκτηση δεδομένων και για την υλοποίηση των πειραμάτων (σχήμα 3.1). Η οποία όμως θα μας έδινε και την δυνατότητα να αλλάζουμε τις παραμέτρους του PID ελεγκτή, κατά την λειτουργία του για την καλύτερη μελέτη της συμπεριφοράς του. Εν συνέχεια εκμεταλλευόμενοι την δυνατότητα καταγραφής δεδομένων μπορέσαμε να δείξουμε πως μπορεί να γίνει χρήση των εργαλείων του Matlab για την βελτιστοποίηση των υπό μελέτη συστημάτων μας. Η ανάλυση αυτής της κατασκευής γίνεται στο παρόν κεφάλαιο.



Σχήμα 3.1 – Υλικά Κατασκευής ελεγκτή PID Arduino

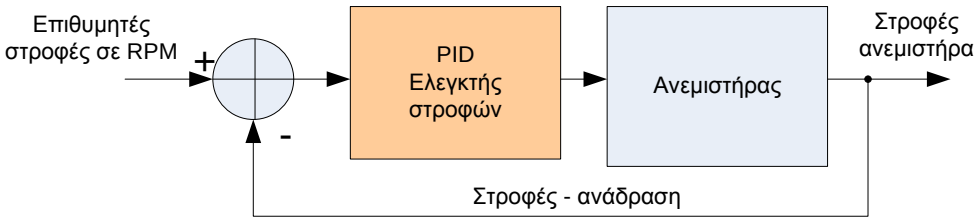
Ο συνολικός τύπος ελέγχου που ερευνάται είναι αυτός που φαίνεται στο σχήμα 3.2.



Σχήμα 3.2 – Συνολικό διάγραμμα βαθμίδων εργασίας.Mode 1

Αρχικά η μελέτη, όμως, ερευνά τον εσωτερικό βρόχο ελέγχου ξεχωριστά. Έτσι γίνεται ανάλυση δύο mode λειτουργίας:

mode 0. Έλεγχος μόνο στροφών κινητήρα ανεμιστήρα, όπως φαίνεται στο σχήμα 3.3. Η πειραματική ανάλυση γίνεται στο Κεφάλαιο 4.



Σχήμα 3.3 – Διάγραμμα βαθμίδων εσωτερικού βρόχου.Mode 0

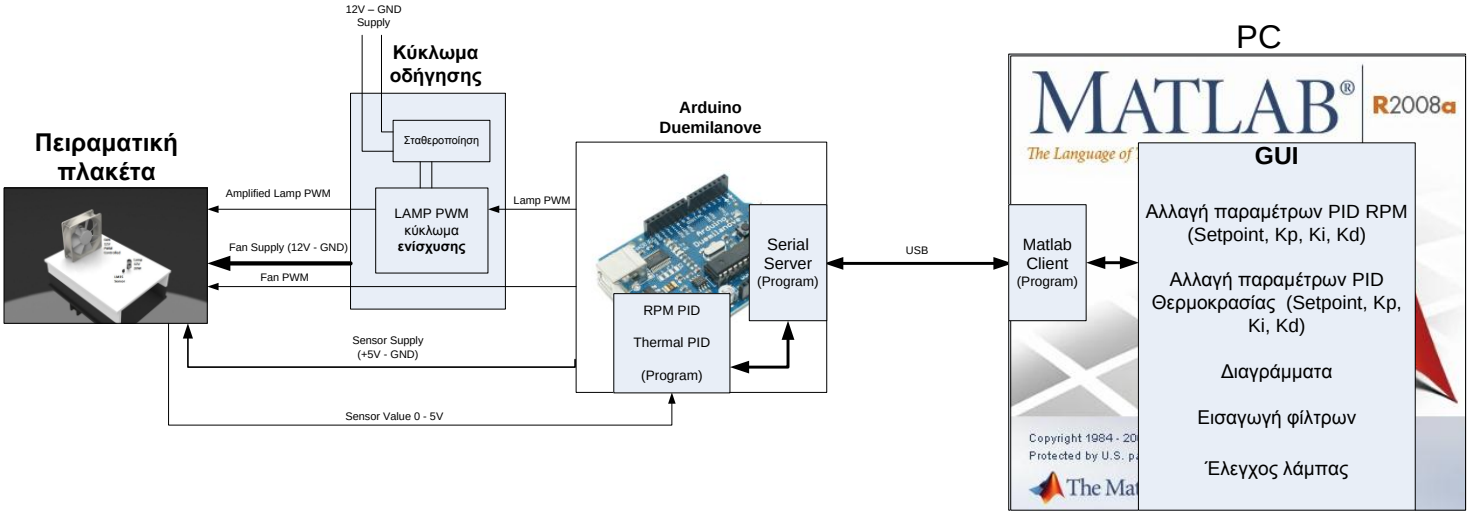
mode 1. Έλεγχος θερμοκρασίας και σε σειρά έλεγχος στροφών κινητήρα ανεμιστήρα, όπως φαίνεται στο σχήμα 3.2. Η πειραματική ανάλυση γίνεται στο Κεφάλαιο 5.

Η ανάλυση που γίνεται συμπεριλαμβάνει την εξαγωγή των συναρτήσεων μεταφοράς για κάθε άγνωστο σύστημα που παρουσιάζεται στο διάγραμμα βαθμίδων και την σύγκριση των θεωρητικών αποτελεσμάτων σε σχέση με τα πειράματα. Επίσης

περιλαμβάνεται η ρύθμιση των παραμέτρων των ελεγκτών με manual τρόπο, αλλά και με αυτόματο τρόπο (auto tuning) μέσω των εργαλείων του Matlab.

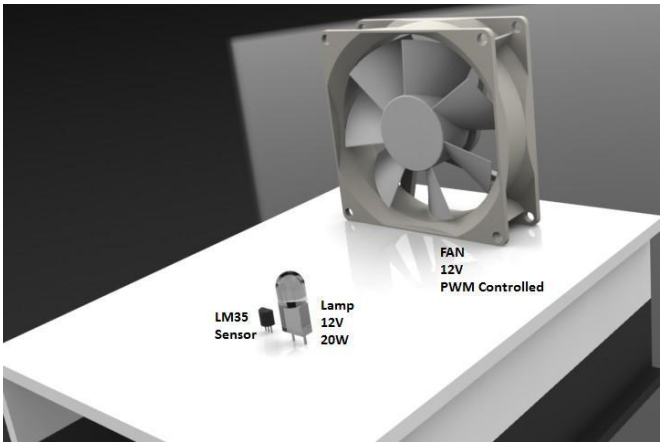
3.2 Δομή της Κατασκευής

Στα πλαίσια της εργασίας δημιουργήθηκε μια κατασκευή για την υλοποίηση του ελέγχου στροφών του ανεμιστήρα και θερμοκρασίας. Στο σχήμα 3.4 παρουσιάζεται το διάγραμμα βαθμίδων της συνολικής κατασκευής που υλοποιήθηκε έτσι ώστε να εκτελεστούν τα πειράματα.

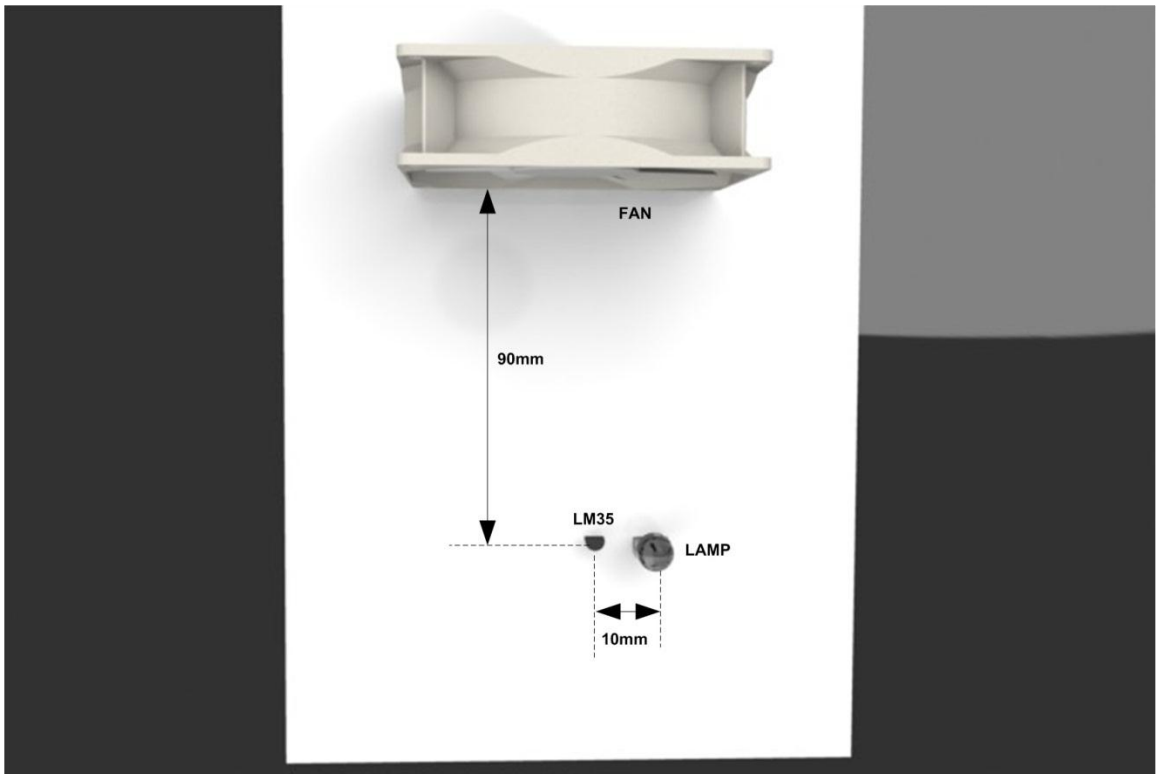


Σχήμα 3.4 – Διάγραμμα βαθμίδων κατασκευής εργασίας.

Η πειραματική πλακέτα (σχήμα 3.4 αριστερά) αποτελεί μια κατασκευή η οποία στερεώνει το σύστημα ανεμιστήρα – αισθητήρα, καθώς περιέχει και την λάμπα για την θέρμανση του περιβάλλοντος. Στα σχήματα 3.5 και 3.6 φαίνεται αναλυτικά η χωροταξία με την οποία είναι τοποθετημένα τα στοιχεία.



Σχήμα 3.5 – Πειραματική πλακέτα.



Σχήμα 3.6 – Πειραματική πλακέτα (κάτοψη).

Η τοποθέτηση των στοιχείων έγινε έτσι ώστε να έχουμε την μέγιστη δυνατή εξάρτηση των στροφών του ανεμιστήρα σε σχέση με την πτώση θερμοκρασίας. Έτσι η απόσταση του ανεμιστήρα από τον αισθητήρα αλλάζει για να επηρεάζει περισσότερο ή λιγότερο την πτώση θερμοκρασίας (δεν είναι σταθερά 90mm).

Στην συνέχεια έχουμε την πλακέτα Arduino Duemilanove η οποία εκτελεί μέσω software τον PID έλεγχο τόσο για τις στροφές του ανεμιστήρα, όσο και για την θερμοκρασία. Επίσης υπάρχει κομμάτι προγράμματος σύμφωνα με το οποίο υλοποιείται ένας σειριακός εξυπηρετητής (server) για την ανταλλαγή δεδομένων με προσωπικό υπολογιστή. Τα παραπάνω αναλύονται στις αντίστοιχες ενότητες.

Το κύκλωμα οδήγησης αποτελεί κύκλωμα το οποίο ενισχύει και τροφοδοτεί κατάλληλα τα στοιχεία της πειραματικής πλακέτας. Για την ακρίβεια ενισχύει το PWM σήμα που πρόκειται να τροφοδοτήσει την λυχνία και τροφοδοτεί με 12 Volt τον ανεμιστήρα.

Τέλος έχουμε τον ηλεκτρονικό υπολογιστή ο οποίος είναι συνδεδεμένος μέσω USB με το Arduino. Για την απεικόνιση, διεπαφή με τον χρήστη και επικοινωνία με το Arduino χρησιμοποιήθηκαν προγράμματα στην γλώσσα Matlab. Συγκεκριμένα

υλοποιήθηκε αντικείμενο Matlab για την υλοποίηση σειριακού Client για την επικοινωνία με τον Server του Arduino. Επίσης υλοποιήθηκε γραφικό περιβάλλον για τον χρήστη μέσα από το οποίο είναι σε θέση να τροποποιήσει τις παραμέτρους για τους δύο PID ελεγκτές. Έτσι μπορεί να αλλάξει τα Setpoint, Ki, Kp, Kd για τους ελεγκτές στροφών και θερμοκρασίας. Επίσης μέσω του γραφικού περιβάλλοντος γίνεται η απεικόνιση των εσωτερικών μεταβλητών του συστήματος σε διαγράμματα. Επίσης δίνεται η δυνατότητα στον χρήστη για αλλαγή της φωτεινότητας (θέρμανσης) της λάμπας. Άλλες επιλογές του γραφικού περιβάλλοντος αποτελούν η αποθήκευση όλου του πειράματος και των τιμών του και η εισαγωγή ενός φίλτρου κινούμενου μέσου (moving average) για την καλύτερη οπτικοποίηση των διαγραμμάτων. Τα παραπάνω αναλύονται στις παρακάτω ενότητες ενώ εικόνες από το γραφικό περιβάλλον βρίσκονται στο Παράρτημα Δ'.

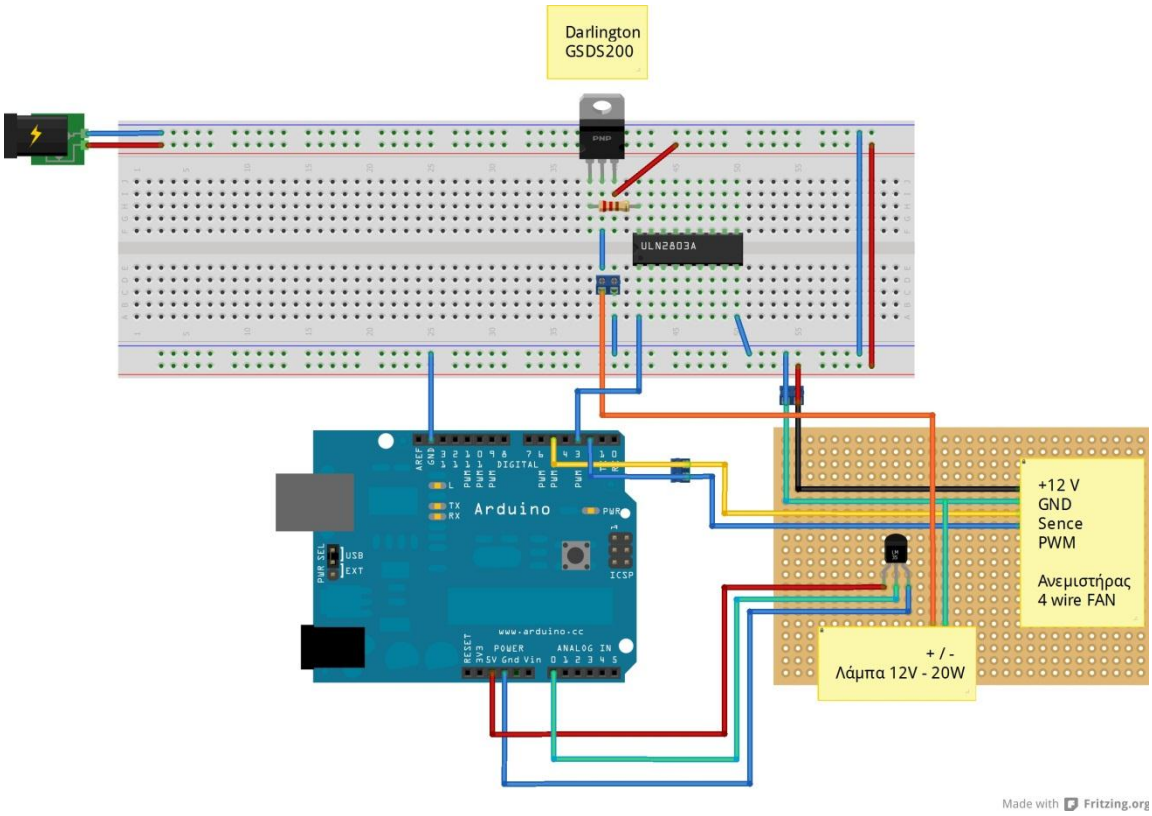
Τέλος, κατασκευάστηκε συνάρτηση στο Matlab με την οποία έχουμε την συνολική εκτύπωση των μεταβλητών του συστήματος ενός πειράματος το οποίο είχαμε αποθηκεύσει μέσω του GUI.

Θα πρέπει να σημειωθεί πως υπάρχουν δύο MODE λειτουργίας των πειραμάτων:

- **MODE 1:** Έλεγχος θερμοκρασίας (PID) και στην συνέχεια σε σειρά έλεγχος στροφών (PID) έτσι ώστε τελικά να ελέγχεται η θερμοκρασία (έτσι όπως αναλύεται στον σχήμα 3.2).

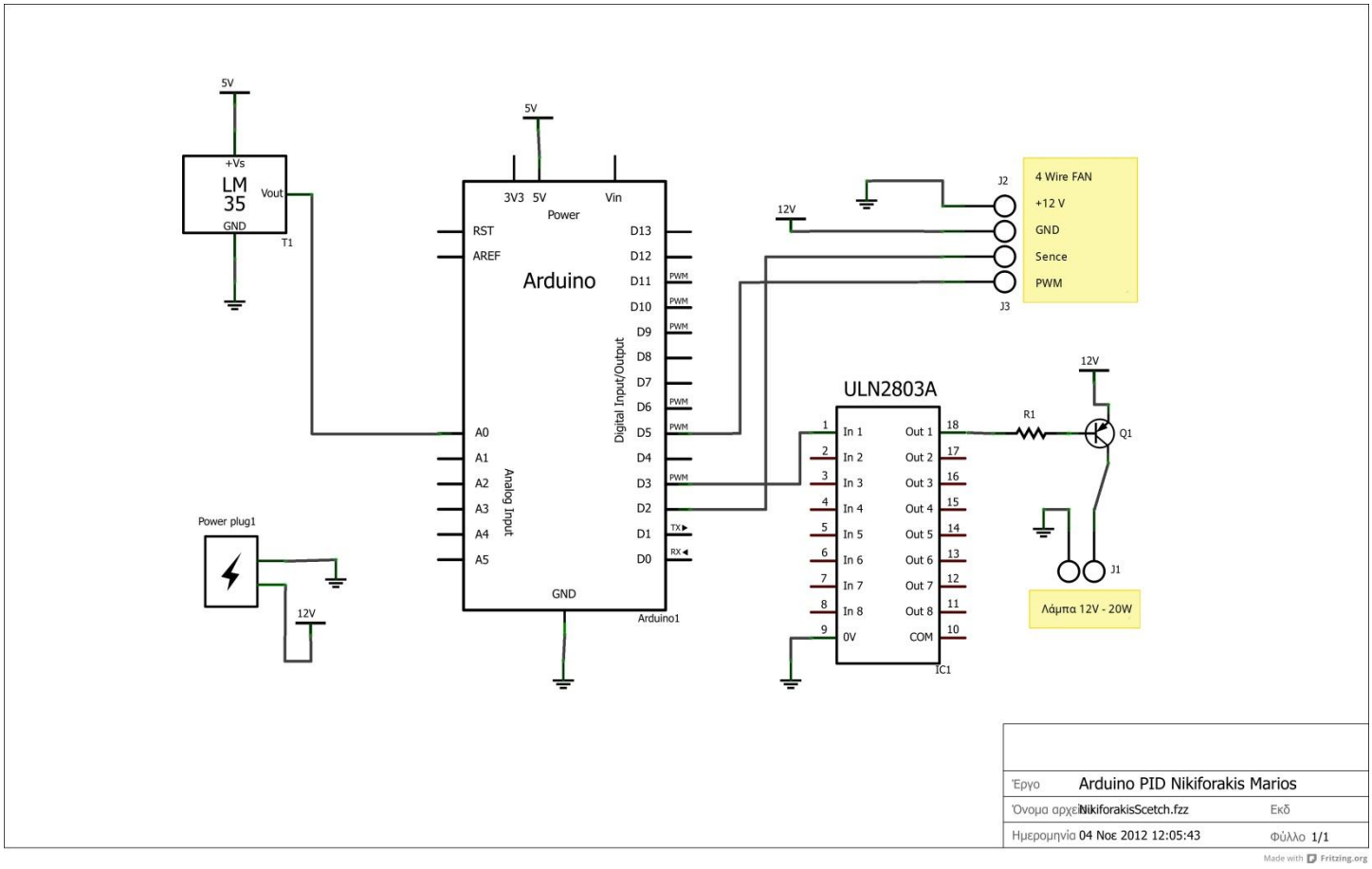
- **MODE 0:** Μόνο έλεγχος στροφών (PID). (έτσι όπως αναλύεται στον σχήμα 3.3).

3.3 Ηλεκτρονικό Κύκλωμα



Σχήμα 3.7 – Breadboard κύκλωμα σε Fritzing

Στο σχήμα 3.7 φαίνεται το κύκλωμα σε δοκιμαστική πλακέτα όπως υλοποιήθηκε με το λογισμικό Fritzing και στο σχήμα 3.8 το σχηματικό. Το Fritzing είναι ένα open-source λογισμικό για την υποστήριξη ηλεκτρονικού σχεδίου για επαγγελματίες αλλά και για χομπίστες που θα τους βοηθήσει μέσα από ένα απλό και απόλυτα κατανοητό μενού να σχεδιάσουν και να φτιάξουν δημιουργικά διαδραστικά ηλεκτρονικά κυκλώματα (σχέδιο) και τυπωμένα κυκλώματα.

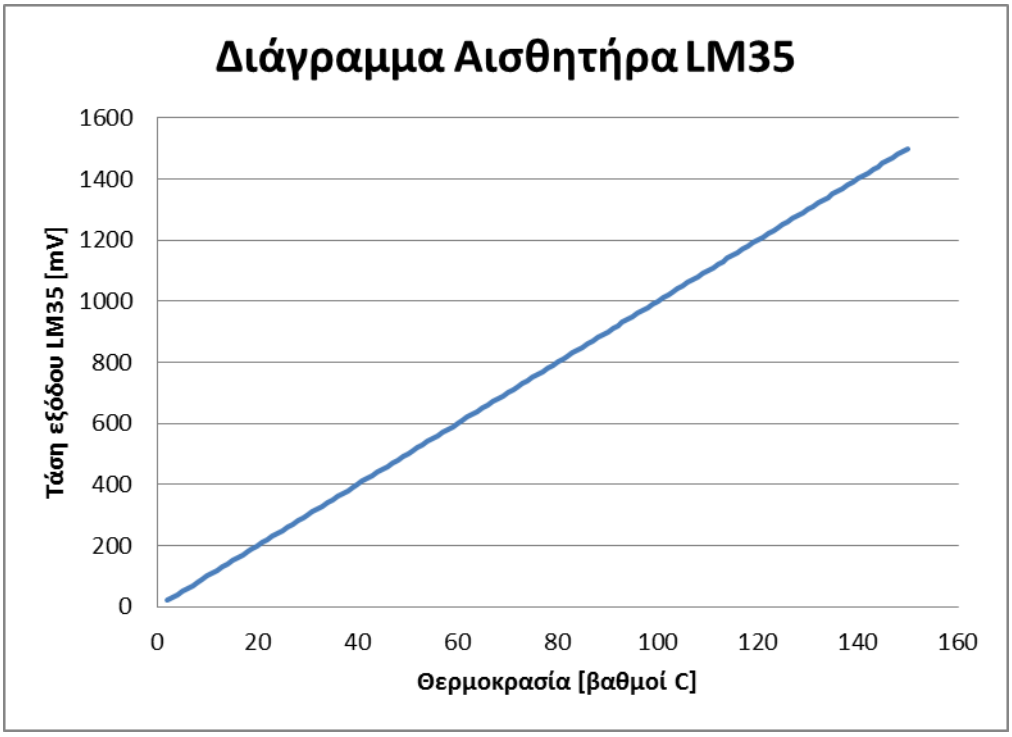


Σχήμα 3.8 – Σχηματικό κύκλωμα κατασκευής.

Το μοντέλο Arduino που χρησιμοποιείται είναι το Duemilanove του οποίου τα χαρακτηριστικά αναφέρονται παραπάνω.

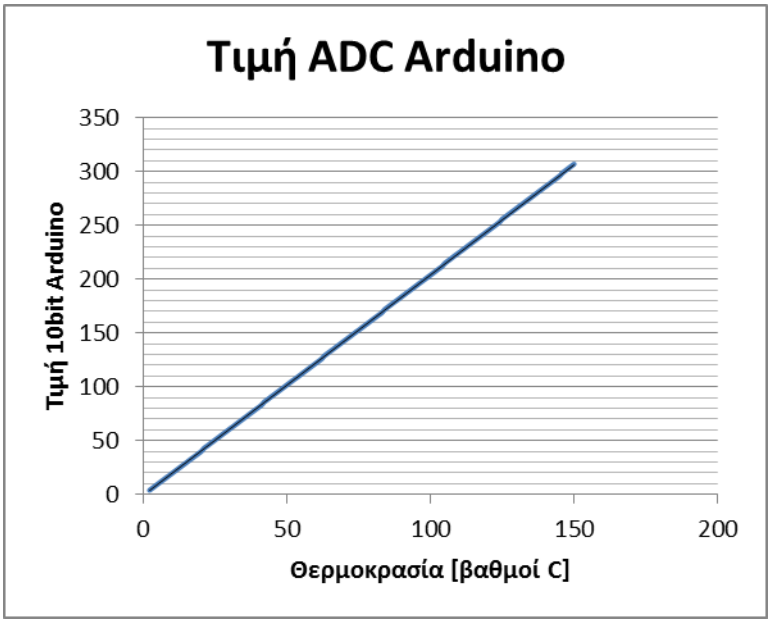
Για την τροφοδοσία των 12V του κυκλώματος χρησιμοποιείται τροφοδοτικό των 12 V και 3 A, ενώ η τροφοδοσία των 5 V γίνεται μέσω της πλακέτας Arduino, άρα μέσω του USB του υπολογιστή.

Ο αισθητήρας θερμοκρασίας που χρησιμοποιείται είναι ο LM35. Ο αισθητήρας αυτός είναι ένας μεγάλης ακρίβειας αισθητήρας θερμοκρασίας σε μορφή ολοκληρωμένου κυκλώματος. Η τάση εξόδου του αισθητήρα είναι γραμμικά ανάλογη με την θερμοκρασία σε βαθμούς Κελσίου. Για κλίμακα μεταξύ 2 και 150ο C η έξοδος του αισθητήρα παραμένει γραμμική, ενώ η κλίση της ευθείας είναι 10 mV / oC, όπως φαίνεται και στο σχήμα 3.9.



Σχήμα 3.9 – Διάγραμμα γραμμικής περιοχής αισθητήρα.

Η έξοδος του αισθητήρα είναι συνδεδεμένη με την πρώτη αναλογική είσοδο του Arduino. Σε αυτό το σημείο θα πρέπει να σημειώσουμε πως η μετατροπή από αναλογικό σε ψηφιακό που περιέχει το Arduino (λόγο του μικροελεγκτή AVR) είναι της ακρίβειας των 10bit, όπως αναφέρεται και σε παραπάνω ενότητα. Έτσι οι τιμές της θερμοκρασίας σε σχέση με την τιμή που θα διαβάζεται από την πλακέτα φαίνεται στο σχήμα 3.10.



Σχήμα 3.10 – Τιμές του ADC του Arduino για τιμές θερμοκρασίας.

Σχήμα 3.11 – Αισθητήρας θερμοκρασίας LM35

Θα πρέπει να σημειωθεί πως χρησιμοποιήθηκε ψήκτρα τοποθετημένη στον αισθητήρα, έτσι ώστε να υπάρχει καλύτερη απαγωγή της θερμότητας κατά την λειτουργία του ανεμιστήρα.

Ο ανεμιστήρας που χρησιμοποιήθηκε είναι τύπου τεσσάρων καλωδίων. Έτσι ο κινητήρας είναι ένας κινητήρας συνεχούς ρεύματος του οποίου η ταχύτητα ελέγχεται μέσω PWM (Pulse Width Modulation), ενώ περιέχει hall sensor για την μέτρηση των στροφών του. Έτσι κάθε πλήρης περιστροφή δίνει δύο παλμούς, κατά τους οποίους γειώνεται ο αντίστοιχος ακροδέκτης. Λόγο του παραπάνω, για να δεχτούμε σαν σήμα τους παλμούς αυτούς, χρειάζεται μια pull-up αντίσταση η οποία, στο συγκεκριμένο κύκλωμα, περιέχεται (και ενεργοποιείται) στην ψηφιακή είσοδο του Arduino. Έτσι οι ακροδέκτες που προσφέρει ο ανεμιστήρας είναι οι εξής:

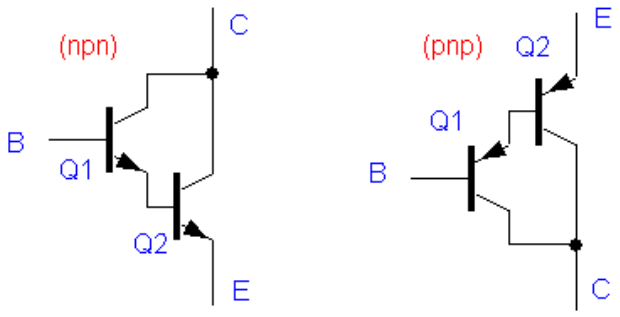
Πίνακας 3.1 – Καλώδια ανεμιστήρα

Pin	Όνομα	Χρώμα	Περιγραφή
1	GND	Μαύρο	Γείωση
2	12V	Κίτρινο	Τροφοδοσία ανεμιστήρα 12 V
3	Sense (tach.)	Πράσινο	Ανάδραση αισθητήρα ταχύτητας
4	Control (PWM)	Μπλε	Σήμα PWM για καθορισμό ταχύτητας

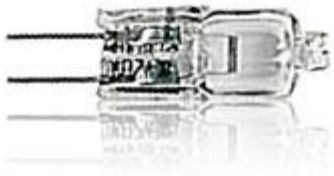
Το ακριβές μοντέλο του ανεμιστήρα που χρησιμοποιήθηκε είναι το D12SL-12 (σχήμα 3.12). Το εύρος του σε στροφές είναι από 800 RPM ως 1800 RPM περίπου για τιμές του Duty Cycle του PWM από 0% μέχρι 100% αντίστοιχα.

Σχήμα 3.12 – Ανεμιστήρας D12SL-12

Η λάμπα για την θέρμανση που χρησιμοποιήθηκε είναι λαμπτήρας 12 V – 20 W (σχήμα 3.14). Για την αυξομείωση της φωτεινότητας, άρα και της θέρμανσης, η τροφοδοσία της υλοποιείται μέσω PWM το οποίο ενισχύεται κατάλληλα και ελέγχεται από το Arduino. Για την ενίσχυση του σήματος χρησιμοποιείτε το ολοκληρωμένο ULN2803 και στην συνέχεια ένα τρανζίστορ τύπου Darlington. Το ολοκληρωμένο ULN2803 αποτελείται από μια σειρά τρανζίστορ Darlington τύπου NPN ενώ επιτρέπει την διέλευση ρεύματος μικρότερου από 500mA. Έτσι χρειάζεται ακόμα μια βαθμίδα ενίσχυσης, αφού η λάμπα ζητά 1.6 A. Η βαθμίδα αυτή υλοποιείται με την χρήση του GSDS200, ενός Darlington τρανζίστορ τύπου PNP. Η συνδεσμολογία Darlington φαίνεται στο σχήμα 3.13.



Σχήμα 3.13 – Συνδεσμολογία Darlington



Σχήμα 3.14 – Λυχνία 12V / 20W

3.4 Λογισμικό - Software

Το software μέρος της κατασκευής αποτελείται από τα εξής κομμάτια κώδικα:

- Κώδικας Wiring του Arduino για την υλοποίηση του Serial Server
- Κώδικας Wiring του Arduino για την υλοποίηση των ελεγκτών
- Κώδικας Matlab για την υλοποίηση του Serial Client
- Κώδικας Matlab για την υλοποίηση του Γραφικού περιβάλλοντος διεπαφής
- Κώδικας Matlab για την τύπωση των αποτελεσμάτων σε διαγράμματα

Ο συνολικός κώδικας για καθένα από τα παραπάνω κομμάτια βρίσκεται στα παραρτήματα Α', Β', Γ' και Ε'. Παρακάτω γίνεται η ανάλυση αυτών των κομματιών για τα προγράμματα που αφορούν το Arduino και το Matlab ξεχωριστά.

3.5 Ο προγραμματισμός του Arduino

Για την επικοινωνία του Arduino με το γραφικό περιβάλλον του υπολογιστή δημιουργήθηκε πρόγραμμα σειριακού εξυπηρετητή εντός του Arduino. Σκοπός αυτού του προγράμματος είναι να αναμένει για πιθανή εντολή, και να δρα κατάλληλα. Έτσι, στην πραγματικότητα, το παραπάνω σύστημα υλοποιείται με μια μηχανή καταστάσεων, κατά την οποία κάθε κώδικας ενός Client μπορεί να αποδώσει ένα συγκεκριμένο αποτέλεσμα. Έτσι οι πιθανές εντολές για τον εξυπηρετητή είναι αυτές που εμφανίζονται στον Πίνακα 3.2.

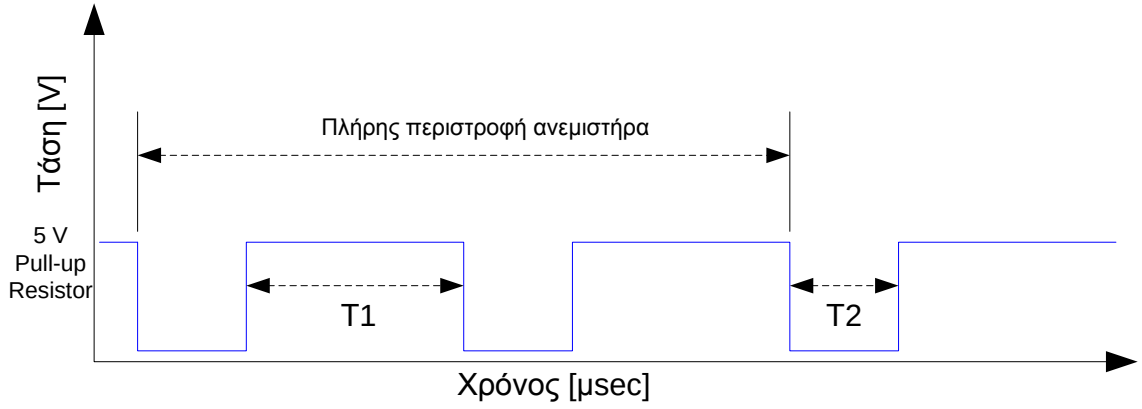
Πίνακας 3.2 – Εντολές Σειριακού Server στο Arduino

Εντολή	ASCII code	Περιγραφή
0	**0x30**	**Read setpoint PID RPM:** Επιστρέφει την τιμή του Setpoint για τον PID ελεγκτή των στροφών του ανεμιστήρα σε RPM.
1	**0x31**	**Read RPM:** Επιστρέφει την τιμή των στροφών του ανεμιστήρα σε RPM.
2XXX	**0x32**	**Set PWM Lamp:** Διαβάζει την τιμή XXX (0 ως 100) και ρυθμίζει το PWM της λυχνίας σε αυτή την τιμή.
3XXXX	**0x33**	**Set Setpoint of RPM:** Διαβάζει την τιμή XXXX (0 ως 2000) και θέτει την τιμή αυτή στο Setpoint του ελεγκτή των στροφών.
4$XXXX	**0x34**	**Set Kp,i or d of PID of RPM**: Διαβάζει την τιμή XXXX (0 ως 9999) και την διαιρεί με το 100, δημιουργώντας έτσι τον πραγματικό αριθμό XX.XX και στην συνέχεια τον τοποθετεί στο Kp, Ki ή Kd αν το $ είναι 1, 2 ή 3 αντίστοιχα. Παράδειγμα: αν δοθεί η εντολή 425618 τότε θα αποθηκευτεί η τιμή 56.18 στον Ki του ελεγκτή στροφών του ανεμιστήρα.
5	**0x35**	**Read Sensor:** Επιστρέφει την τιμή του Analog2Digital του αισθητήρα θερμοκρασίας (από 0 ως 1023)
6X	**0x36**	**Set mode:** Αν το X είναι ίσο με 0 τότε έχουμε λειτουργία μόνο του PID ελεγκτή για τις στροφές, αν το X είναι ίσο με 1 τότε έχουμε λειτουργία και των δύο PID ελεγκτών όπως ορίζεται στο συνολικό block διάγραμμα σε παραπάνω ενότητα.
7XXXX	**0x37**	**Set Setpoint of Temperature:** Διαβάζει την τιμή XXXX (0 ως 1023) και θέτει την τιμή αυτή στο Setpoint του ελεγκτή της θερμοκρασίας. (μονάδες όχι βαθμούς Κελσίου, αλλά μονάδες A2D αισθητήρα)
8$XXXX	**0x38**	**Set Kp,i or d of PID of Temperature:** Διαβάζει την τιμή XXXX (0 ως 9999) και την διαιρεί με το 100, δημιουργώντας έτσι τον πραγματικό αριθμό XX.XX και στην συνέχεια τον τοποθετεί στο Kp, Ki ή Kd αν το $ είναι 1, 2 ή 3 αντίστοιχα. Παράδειγμα: αν δοθεί η εντολή 836529 τότε θα αποθηκευτεί η τιμή 65.29 στον Kd του ελεγκτή θερμοκρασίας.
9	**0x39**	**Read PWM:** Επιστρέφει την τιμή του duty cycle του PWM που δίνεται αυτή την στιγμή στον ανεμιστήρα. (Εσωτερική μεταβλητή)

Το πρόγραμμα στο Arduino περιέχει και το κομμάτι που αφορά τους ελεγκτές PID οι οποίοι υλοποιούνται μέσω της βιβλιοθήκης PID του Arduino. Έτσι στην συνάρτηση αρχικοποίησης γίνεται ο ορισμός του τύπου των ελεγκτών. Ο ελεγκτής ο οποίος έχει σαν είσοδο τα επιθυμητά RPM, την ανάδραση των RPM και σαν έξοδο το PWM που πρέπει να δώσει, ορίζεται ως ευθύς (DIRECT) αφού όσο αυξάνεται το duty cycle του PWM τόσο αυξάνονται και οι στροφές του κινητήρα. Από την άλλη ο ελεγκτής ο οποίο έχει ως είσοδο την επιθυμητή τιμή θερμοκρασίας και την ανάδραση θερμοκρασίας και σαν έξοδο επιθυμητή τιμή RPM, ορίζεται ως αντίστροφος (REVERSE) αφού αυξάνοντας τα RPM μειώνεται η θερμοκρασία. Σε αυτό το σημείο θα πρέπει να θυμηθεί ο αναγνώστης πως η έξοδος του τελευταίου ελεγκτή εισέρχεται ως είσοδος (επιθυμητά RPM) στον πρώτο ελεγκτή. Επίσης θα πρέπει να σημειωθεί πως έγινε ένα calibration ενός κέρδους ενδιάμεσα από τους δύο ελεγκτές, αφού η τιμή της θερμοκρασίας έχει μεγάλη διαφορά σε τάξεις μεγέθους από αυτή των RPM. Χωρίς το παραπάνω κέρδος οι αλλαγές που έθετε ο ελεγκτής ήταν πολύ μικρές για φυσιολογικές τιμές των Kp, Ki και Kd.

Θα πρέπει να τονιστεί σε αυτό το σημείο πως η περίοδος δειγματοληψίας του PID του Arduino είναι προκαθορισμένη στην τιμή Ts=0.2s, δηλαδή Fsard=5Hz.

Στο πρόγραμμα του Arduino υλοποιήθηκε και συνάρτηση ανάγνωσης των στροφών του ανεμιστήρα. Το συγκεκριμένο μοντέλο ανεμιστήρα είναι γνωστό πως περιέχει αισθητήρες που δίνουν δύο αρνητικούς παλμούς σε κάθε στροφή, όπως φαίνεται στο σχήμα 3.15.



Σχήμα 3.15 – Σήμα του αισθητήρα στροφών του ανεμιστήρα.

Έτσι ο χρόνος μιας πλήρους περιστροφής υπολογίζεται σύμφωνα με την παρακάτω σχέση (Σχέση 3.1).

$$T_{FAN} = 2(T_1 + T_2) \tag{3.1}$$

Για να μετρηθεί ο χρόνος του θετικού παλμού (T1) και ο χρόνος του αρνητικού (T2) χρησιμοποιείται η συνάρτηση "pulseIn()" της βιβλιοθήκης του Arduino, για όρισμα HIGH και LOW αντίστοιχα. Έτσι η τιμές των παλμών διαβάζονται σε μSeconds.

Στην συνέχεια για τον υπολογισμό των στροφών του κινητήρα σε Revolutions Per Minute (RPM) υπολογίζεται η παρακάτω σχέση:

$$F_{RPM} = 60\frac{1}{T_{FAN}}10^6 \tag{3.2}$$

Στο επαναλαμβανόμενο κομμάτι του κώδικα (Loop) γίνεται η ανάγνωση των τιμών ανάδρασης (RPM και θερμοκρασίας). Η ανάγνωση των RPM γίνεται μέσω ψηφιακής εισόδου και της συνάρτησης που αναλύεται παραπάνω. Η ανάγνωση της θερμοκρασίας γίνεται μέσω αναλογικής εισόδου του Arduino, έτσι αποτελεί μια τιμή από 0 ως 1023 (10bits). Στην συνέχεια οι τιμές αυτές εισέρχονται στους κατάλληλους ελεγκτές, εξάγεται το αποτέλεσμα από αυτούς, το οποίο εισάγεται στις κατάλληλες εξόδους. Το αποτέλεσμα που αφορά το PWM εξάγεται στον ακροδέκτη του Arduino που είναι συνδεδεμένο το PWM του ανεμιστήρα, ενώ το αποτέλεσμα που αφορά την επιθυμητή τιμή στροφών περνά στον ελεγκτή στροφών. Επίσης στο επαναλαμβανόμενο κομμάτι καλείται και ο σειριακός Server συνεχώς και χωρίς καθυστέρηση.

Θα πρέπει να σημειωθεί πως ο κώδικας, που αφορά την ενότητα αυτή, βρίσκεται στο Παράρτημα Α'.

3.6 Ο προγραμματισμός στο Matlab

Τα κομμάτια προγράμματος που υλοποιήθηκαν στο Matlab χωρίζονται σε τρείς ομάδες:

- Σειριακή επικοινωνία (Serial Client), δημιουργία αντικειμένου. Παράρτημα Β'
- Γραφικό περιβάλλον. Παράρτημα Γ'
- Κώδικας για την τύπωση των αποτελεσμάτων των πειραμάτων. Παράρτημα Ε'

3.6.1 Σειριακή επικοινωνία

Για την σειριακή επικοινωνία του Matlab με το Arduino χρησιμοποιήθηκαν οι εντολές που ορίζονται στον πίνακα 3.2. Έτσι κατασκευάστηκε αντικείμενο το οποίο κατά την δημιουργία του ανοίγει μια σειριακή σύνδεση με το Arduino, ταχύτητας 115200 bps. Κατά την διαγραφή του αντικειμένου κλείνει αυτή την σειριακή σύνδεση. Το αντικείμενο αυτό περιέχει τις συναρτήσεις που φαίνονται στον πίνακα 3.3.

Πίνακας 3.3 –Συναρτήσεις Σειριακού Client στο Matlab

Συνάρτηση	Εντολή	Ορίσματα	Περιγραφή
`a=PIDarduino(comPort)`	---	**comPort:** όνομα σειριακής θύρας. Π.χ. 'COM3'	Ο **constructor** του αντικειμένου. Ορίζει την σειριακή θύρα.
`delete(a)`	---	---	Διαγράφει το αντικείμενο και κλείνει την σειριακή θύρα.
`val = getSpointRPM(a)`	0	---	**Read setpoint PID RPM:** Επιστρέφει την τιμή του Setpoint για τον PID ελεγκτή των στροφών του ανεμιστήρα σε RPM.
`val = getRPM(a)`	1	---	**Get RPM:** Επιστρέφει την τιμή των στροφών του ανεμιστήρα σε RPM.
`val = setLamp(a,val)`	2XXX	**val:** τιμή του Duty cycle του PWM.	**Set PWM Lamp:** ρυθμίζει το PWM της λυχνίας σε αυτή την τιμή.
`val = setSpointRPM(a , val)`	3XXXX	**val:** νέα τιμή του Setopoint για τον ελεγκτή στροφών.	**Set Setpoint of RPM:** θέτει την τιμή val στο Setpoint του ελεγκτή των στροφών.

val = setKRpm(a , i , val)	4$XXXX	**i:** δηλώνει σε ποιον συντελεστή αναφερόμαστε. 1 για Kp, 2 για Ki και 3 για Kd. **val:** Η νέα τιμή του συντελεστή.	**Set Kp, i or d of PID of RPM**: τοποθετεί την τιμή **val** στο Kp, Ki ή Kd αν το **i** είναι 1, 2 ή 3 αντίστοιχα.
val = getTemp(a)	5	---	**Get temperature:** Επιστρέφει την τιμή της θερμοκρασίας σε βαθμούς ^{o}C.
val = changeMode(a , mode)	6X	**mode:** το νέο mode που είναι επιθυμητό.	**Set mode:** Αν το mode είναι ίσο με **'R'** τότε έχουμε λειτουργία μόνο του PID ελεγκτή για τις στροφές, αν το mode είναι ίσο με **'T'** τότε έχουμε λειτουργία και των δύο PID ελεγκτών όπως ορίζεται στο συνολικό block διάγραμμα.
val = setSpointTemp(a , temper)	7XXXX	**temper:** νέα τιμή του Setopoint για τον ελεγκτή στροφών.	**Set Setpoint of Temperature:** θέτει την τιμή αυτή στο Setpoint του ελεγκτή της θερμοκρασίας. Οι μονάδες σε βαθμούς Κελσίου ^{o}C.
val = setKTemp(a , i , val)	8$XXXX	**i:** δηλώνει σε ποιον συντελεστή αναφερόμαστε. 1 για Kp, 2 για Ki και 3 για Kd. **val:** Η νέα τιμή του συντελεστή.	**Set Kp, i or d of PID of Temperature:** τοποθετεί την τιμή **val** στο Kp, Ki ή Kd αν το **i** είναι 1, 2 ή 3 αντίστοιχα.
val = getPWM (a)	9	---	**Read PWM:** Επιστρέφει την τιμή του duty cycle του PWM που δίνεται αυτή την στιγμή στον ανεμιστήρα. (Εσωτερική μεταβλητή)

Θα πρέπει να σημειωθεί πως ενδιάμεσα από κάθε χρήση της σειριακή θύρας γίνεται αναμονή 1.4 ms έτσι ώστε να μην είναι απασχολημένη (σε περίπτωση εγγραφής) ή δεν έχει φτάσει ακόμα το αποτέλεσμα (σε περίπτωση ανάγνωσης). Για παράδειγμα σε περίπτωση που επιθυμούμε ανάγνωση μιας τιμής και είναι γνωστό πως μετά την αποστολή της εντολής θα υπάρχει η λήψη μιας τιμής, τότε υπάρχει ενδιάμεσα μια αναμονή των 1.4 ms.

Για την αποστολή πολλών στην σειρά ψηφίων, όπως απατούν αρκετές εντολές όπως έχουν οριστεί δημιουργήθηκε η συνάρτηση [digits] = splitDigits(number,Ndigits) η οποία δέχεται έναν ακέραιο και επιστρέφει σε έναν πίνακα N στοιχείων τα δεκαδικά ψηφία της τιμής αυτής.

Κατά την κλήση της συνάρτησης val = getTemp(a) γίνεται μια μετατροπή της τιμής που λαμβάνεται από το Arduino σε βαθμούς Κελσίου ^{o}C. Η αντίστροφη μετατροπή γίνεται κατά την κλήση της συνάρτησης val = setSpointTemp(a , temper) όπου εκεί μια τιμή σε βαθμούς Κελσίου μετατρέπεται σε τιμή που μπορεί να δεχθεί ο Analog to Digital μετατροπέας του Arduino.

Η μετατροπή αυτή γίνεται σύμφωνα με την καμπύλη που φαίνεται στο σχήμα 3.10, ενώ η σχέση που υπολογίζεται είναι η εξής:

$$Temp_{^{o}C} = factor \cdot Temp_{ADC} \tag{3.3}$$

Όπου:

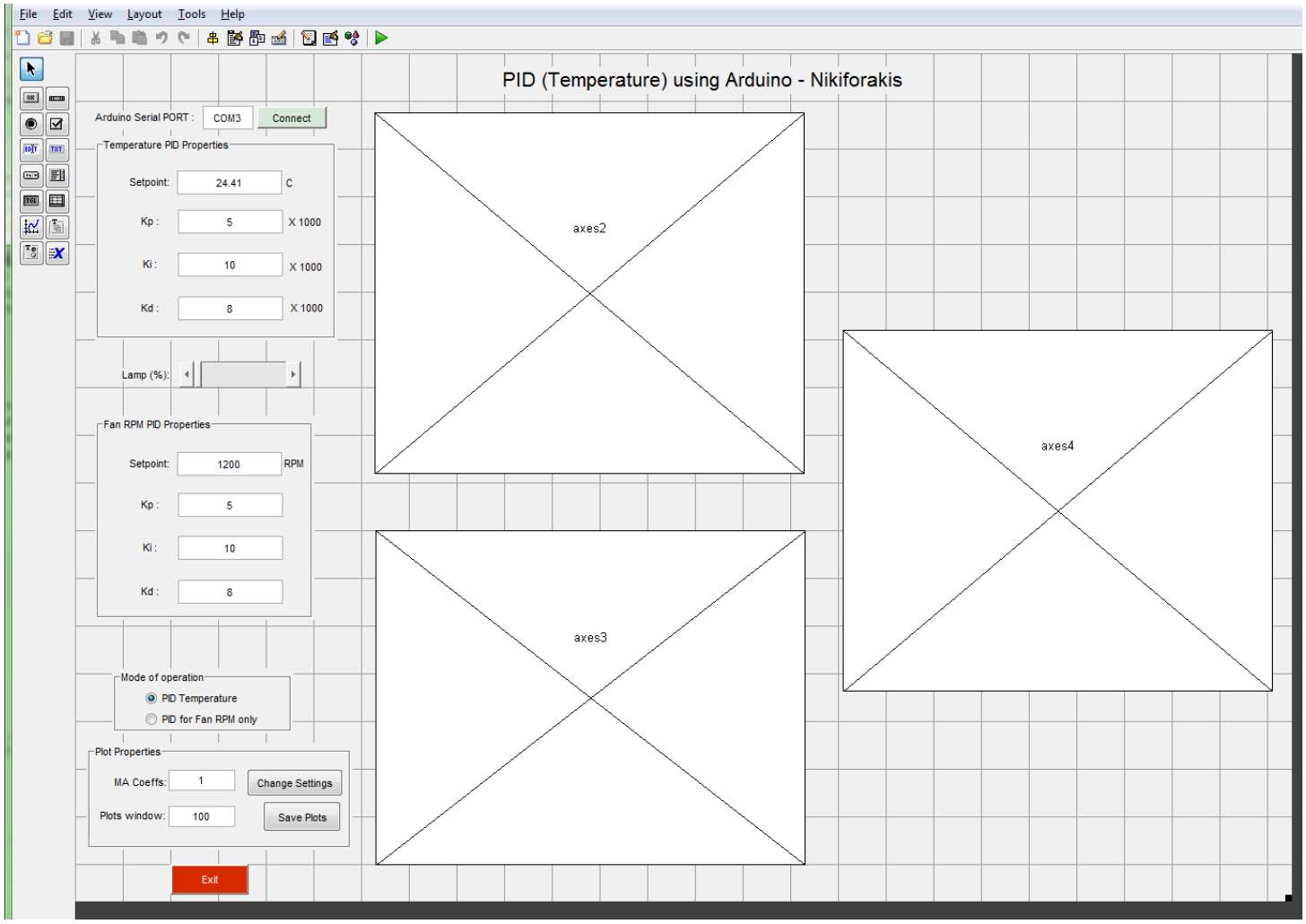
$$factor = \frac{V_{REF}}{2^{N}\dfrac{dV}{dT}} \tag{3.4}$$

Όπου για το Arduino και για τον συγκεκριμένο αισθητήρα (LM35):

$V_{REF} = 5V$ *(Τάση αναφοράς της μετατροπής Αναλογικού σε ψηφιακό ADC)*
$N = 10$ *(έχουμε κβαντισμό των 10 bits, ADC)*
$\frac{dV}{dT} = 0.01\ V/^{o}C$ *(αφού η κλίση Volts /* ^{o}*C του LM35 είναι 10mV/*^{o}*C, σχήμα 4.7)*

3.6.2 Γραφικό περιβάλλον GUI

Το γραφικό περιβάλλον σε Matlab βοηθά τον χρήστη να ορίσει νέες τιμές στο σύστημα για τις παραμέτρους του, και να λάβει απεικονίσεις των εσωτερικών μεταβλητών του συστήματος και των εισόδων/ εξόδων αυτού. Ο κώδικας του συγκεκριμένου κομματιού υπάρχει στο Παράρτημα Γ', ενώ ενδεικτικές εικόνες αυτού υπάρχουν στο Παράρτημα Δ'. Στο σχήμα 3.16 παρατηρούμε το γραφικό περιβάλλον κατά την σχεδίαση του Front panel.



Σχήμα 3.16 – Γραφικό περιβάλλον κατά την σχεδίαση του.

Αρχικά, πάνω αριστερά μπορεί ο χρήστης να ορίσει το όνομα του σειριακού port στο οποίο είναι συνδεδεμένο το Arduino, και στην συνέχεια να γίνει η σύνδεση με αυτό. Τα υπόλοιπα στοιχεία του γραφικού περιβάλλοντος δεν γίνονται ενεργά αν δεν έχει γίνει επιτυχής σύνδεση με το Arduino.

Ένα από τα πιο σημαντικά πεδία του περιβάλλοντος είναι αυτό της επιλογής του mode λειτουργίας, στο οποίο μέσω δύο radio buttons γίνεται η επιλογή από τον χρήστη σε ποια λειτουργία θα βρίσκεται το σύστημα. Έτσι μπορεί να επιλέξει μεταξύ:

- Mode «0»: Λειτουργία του ελεγκτή στροφών μόνο (χωρίς έλεγχο θερμοκρασίας).
- Mode «1»: Λειτουργία και των δύο ελεγκτών όπως φαίνεται στο συνολικό block διάγραμμα (θερμοκρασίας και στροφών, συνδεδεμένοι σε σειρά).

Στον χώρο "Temperature PID properties" ο χρήστης μπορεί να αλλάξει τις παραμέτρους του ελεγκτή θερμοκρασίας όταν το σύστημα λειτουργεί στο mode «1» (δηλαδή και οι δύο ελεγκτές PID ενεργοί). Έτσι βλέπουμε τα πεδία Setpoint, Kp, Ki και Kd τα οποία είναι ενεργά μόνο όταν βρισκόμαστε στο κατάλληλο mode.

Στον χώρο "RPM PID properties" ο χρήστης μπορεί να αλλάξει τις παραμέτρους του ελεγκτή στροφών ανεμιστήρα και για τα δύο mode «0» ή «1» (δηλαδή όταν και οι δύο ελεγκτές PID ενεργοί, ή όταν έχουμε μόνο έλεγχο στροφών). Η μόνη παράμετρος που δε μπορεί να αλλαχθεί κατά το mode «1» είναι το Setpoint για τις στροφές, αφού όταν έχουμε έλεγχο θερμοκρασίας το Setpoint του ελεγκτή στροφών ορίζεται σε κάθε κύκλο από τον ελεγκτή PID της θερμοκρασίας. Έτσι βλέπουμε, και σε αυτή τη περίπτωση, τα πεδία Setpoint, Kp, Ki και Kd τα οποία αυτή τη φορά αφορούν τον έλεγχο στροφών.

Ενδιάμεσα από τους χώρους αυτούς υπάρχει μια μπάρα ολίσθησης μέσω της οποίας ο χρήστης ελέγχει την φωτεινότητα της λυχνίας θέρμανσης, από 0% ως 100%.

Στον χώρο "Plot Properties" ο χρήστης μπορεί να αλλάξει τις επιλογές που αφορούν την εμφάνιση των δεδομένων στα γραφήματα. Έτσι μπορεί να περάσει τις μετρήσεις μέσα από ένα χαμηλοπερατό (Low Pass) φίλτρο τύπου Moving Average (MA) ορίζοντας το πλήθος N των συντελεστών του, αυξάνοντας δηλαδή την συχνότητα αποκοπής του φίλτρου. Στην σχέση 3.5 βλέπουμε συνάρτηση μεταφοράς του φίλτρου.

$$H(z) = \frac{1}{N} \sum_{k=1}^{N} z^{-(k-1)} \tag{3.5}$$

Έτσι όταν το N είναι ίσο με 1, το φίλτρο έχει ως συνάρτηση μεταφοράς H(z)=1, άρα το φίλτρο δίνει το σήμα ως έχει. Το παραπάνω φίλτρο εισέρχεται για την καλύτερη οπτικοποίηση των μετρήσεων αφού λόγω των αισθητήρων και της μετατροπής από αναλογικό σε ψηφιακό εισέρχεται αρκετός υψηλής συχνότητας θόρυβος, αλλοιώνοντας την εικόνα των γραφημάτων.

Επίσης δίνεται στον χρήστη η ικανότητα αλλαγής του χρονικού παραθύρου που είναι ορατό στα γραφήματα, καθώς και η ικανότητα αποθήκευσης όλων των μεταβλητών ως αρχείο .mat με κάθε φορά άλλο όνομα. Το όνομα του αρχείου κάθε

φορά αλλάζει ως εξής: experiment#, όπου # είναι ο αριθμός του πειράματος που αποθηκεύτηκε.

Τέλος, στην δεξιά πλευρά της οθόνης υπάρχουν τρία γραφήματα. Το πάνω αριστερά γράφημα αφορά την απεικόνιση της θερμοκρασίας και του Setpoint του ελεγκτή αυτής, το κάτω αριστερά γράφημα αφορά την μέτρηση των στροφών του κινητήρα και του Setpoint του ελεγκτή στροφών, ενώ το δεξί γράφημα δίνει την εντολή για τις στροφές που δίνει το Arduino (PWM duty cycle). Ενδεικτικές απεικονίσεις των γραφημάτων φαίνονται στο Παράρτημα Δ'. Η ανανέωση των γραφημάτων γίνεται κάθε 0.5sec μέσω κατάλληλου timed event στον κώδικα του GUI. Έτσι η συχνότητα δειγματοληψίας του GUI είναι Fsgui=2Hz, ενώ του Arduino PID είναι όπως είναι προκαθορισμένο με περίοδο 0.2s δηλαδή Fsard=5Hz. Αυτό είναι σημαντικό, αφού όπως θα δούμε παρακάτω θα πρέπει να γίνουν κατάλληλοι μετασχηματισμοί των συντελεστών PID.

Θα πρέπει να σημειωθεί πως κατά την έξοδο (μέσω του button "Exit") από το γραφικό περιβάλλον διαγράφεται το αντικείμενο τύπου arduinoPID (σειριακού Client).

3.6.3 Πρόγραμμα για την εμφάνιση των αποτελεσμάτων

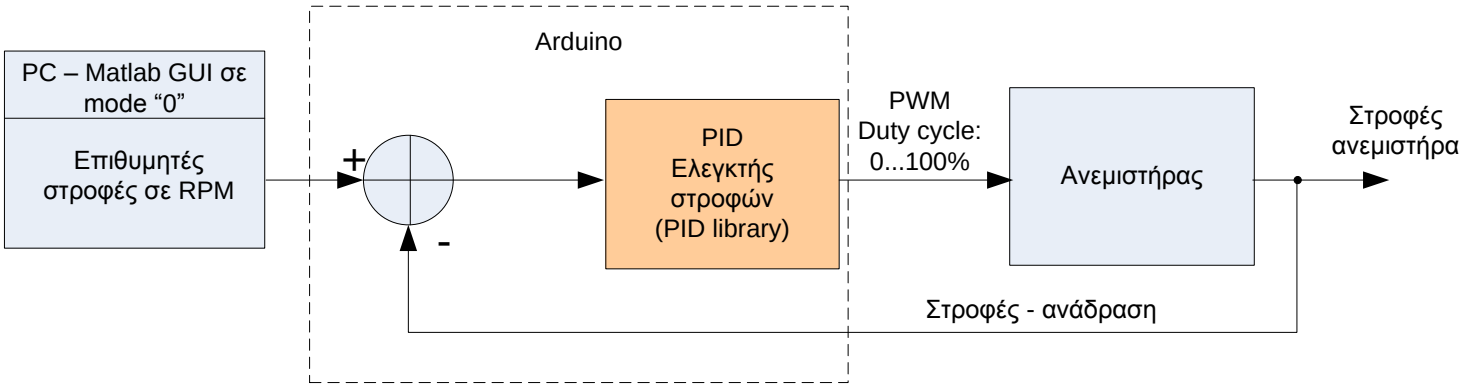
Το πρόγραμμα που κατασκευάστηκε για την εμφάνιση των αποτελεσμάτων έχει ως σκοπό την εμφάνιση των εσωτερικών μεταβλητών και εισόδων εξόδων του συστήματος σαν σήματα σε γραφήματα τα οποία είναι συγχρονισμένα μεταξύ τους.

Τα ορίσματα της συνάρτησης αυτής είναι ο αριθμός του πειράματος που αποθηκευτικέ. Έτσι, στην πράξη, φορτώνεται το αρχείο του οποίου το όνομα είναι "experiment#", όπου # ο αριθμός του πειράματος, και στην συνέχεια γίνεται η κατάλληλη απεικόνιση.

ΚΕΦΑΛΑΙΟ 4
Έλεγχος στροφών ανεμιστήρα

4.1 Εισαγωγή

Ο έλεγχος στροφών του ανεμιστήρα γίνεται μέσω ενός ελεγκτή PID ο οποίος υλοποιείται εντός του Arduino. Το διάγραμμα βαθμίδων του σχήματος 3.3 αποτελεί την δομή ελέγχου που υλοποιείται, ενώ μια πιο αναλυτική μορφή αποτελεί το σχήμα 4.1.



Σχήμα 4.1 – Διάγραμμα βαθμίδων έλεγχου στροφών

Θα πρέπει να σημειωθεί πως για τον έλεγχο της συγκεκριμένης λειτουργίας από τον χρήστη (ξεχωριστά) θα πρέπει το συνολικό σύστημα να βρίσκεται στο mode “0”, γιατί στην περίπτωση του mode “1” την επιθυμητή τιμή των στροφών δε θα την έδινε ο χρήστης, αλλά ο ελεγκτής θερμοκρασίας.

Στο κεφάλαιο αυτό γίνεται αρχικά μια μαθηματική ανάλυση του μοντέλου του ανεμιστήρα και στην συνέχεια μέσω εργαλείων του Matlab επιλέγεται η καλύτερη μοντελοποίηση, και γίνεται το auto-tuning του συστήματος μέσω εργαλείων του Matlab. Τέλος γίνεται η σύγκριση του θεωρητικού μοντέλου σε σχέση με το πραγματικό στο οποίο έχει προηγηθεί manual tuning.

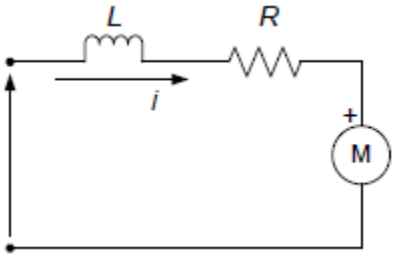
Η μοντελοποίηση του άγνωστου για μας συστήματος θα μελετηθεί μέσω κάποιων από τους τύπους μοντέλων που προσφέρει το εργαλείο δηλαδή:

- Γραμμικό παραμετρικό μοντέλο (Linear parametric model)
  - ARX
  - ARMAX
  - OE
- Μη – γραμμικό μοντέλο (non-Linear model)
  - ARX
  - Hammerstein - Wiener

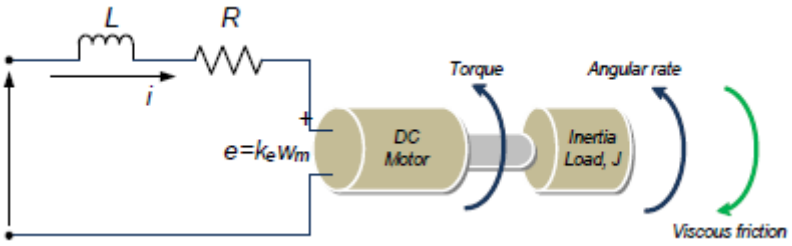
Οι παραπάνω τύποι μοντελοποίησης αναλύονται στο Κεφάλαιο 2.

4.2 Μαθηματικό μοντελο ενός DC κινητήρα

Το ισοδύναμο ηλεκτρικό κύλωμα ενός τυπικού dc κινητήρα απεικονίζεται στο σχήμα 4.2 και 4.3:



Σχήμα 4.2 – Ισοδύναμο ηλεκτρικό κύκλωμα κινητήρα DC.



Σχήμα 4.3 – Διάταξη ηλεκτρομηχανικού συστήματος ενός τυπικού DC κινητήρα.

Τα βασικά ηλεκτρικά στοιχεία είναι η αντίσταση του ρότορα R και η επαγωγή του ρότορα L και η αντιηλεκτρεργετική δύναμη του κινητήρα (ΑΗΕΔ). Η ΑΗΕΔ είναι η τάση που αναπτύσσεται στο τύλιγμα του οπλισμού καθώς αυτός στρέφεται μέσα στο

κύριο μαγνητικό πεδίο του στάτη. Οι σχέσεις που αναπαριστούν τα παραπάνω διαγράμματα 4.2 και 4.3 φαίνονται στην συνέχεια.

Χρησιμοποιώντας τον νόμο τάσης του Kirchhoff, έχουμε την σχέση 4.1 παρακάτω:

$$V_s = R_i + L\frac{di}{dt} + e \tag{4.1}$$

Στη μόνιμη κατάσταση ισορροπίας (steady state) δηλαδή (στην DC κατάσταση μηδενικής συχνότητας), $V_s = R_i + e$

Ως εκ τούτου, όταν βρισκόμαστε σε μεταβατική κατάσταση, η σχέση 4.1 μετασχηματίζεται στην σχέση 4.2 η οποία εμπεριέχει την αντιηλεκτρεργετική δύναμη:

$$e = -R_i - L\frac{di}{dt} + V_s \tag{4.2}$$

Όπου,

$V_s =$ τάση τροφοδοσίας DC

$i =$ ηλεκτρική ένταση στον ρότορα

Ομοίως, λαμβάνοντας υπόψη τις μηχανικές ιδιότητες του dc κινητήρα, σύμφωνα με τον δεύτερο νόμο του Νεύτωνα (νόμο της κίνησης), το αλγεβρικό άθροισμα των ροπών που δρουν πάνω σε ένα στερεό σώμα το οποίο περιστρέφεται γύρω από σταθερό άξονα ισούται με το γινόμενο της ροπής αδρανείας (υπολογιζόμενης ως προς τον άξονα περιστροφής) και της γωνιακής επιτάχυνσης του σώματος. Έτσι προκύπτουν οι εξισώσεις 4.3 και 4.4

$$J\frac{d\omega_m}{dt} = \sum T_i \tag{4.3}$$

$$T_e = k_f\omega_m + J\frac{d\omega_m}{dt} + T_L \tag{4.4}$$

Όπου,

$T_e =$ ηλεκτρική ροπή

$k_f =$ σταθερά τριβής

$J =$ ροπή αδρανείας

$\omega_m =$ γωνιακή ταχύτητα

$T_L =$ συνδεμένο μηχανικό φορτίο,

Όπου η ηλεκτρική ροπή και ή ηλεκτρεργετική δύναμη μπορούν να γραφούν ως:

$$e = k_e \omega_m \text{ και } T_e = k_t \omega_m \tag{4.5}$$

Όπου,

$k_e =$ η σταθερά αντιηλεκτρεγερτικής δύναμης

$k_t =$ σταθερά ροπής

Οπότε ξαναγράφοντας τις εξισώσεις 4.2 και 4.3 έχουμε τις εξισώσεις 4.6 και 4.7,

$$\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{4.6}$$

$$\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L \tag{4.7}$$

Χρησιμοποιώντας τον μετασχηματισμό Laplace στις σχέσεις 4.6 και 4.7, έχουμε τα ακόλουθα (όλες οι αρχικές συνθήκες υπολογίζονται ως μηδενικές):

Για την σχέση 4.6,

$$L\left\{\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s\right\} \tag{4.8}$$

Οπότε έχουμε,

$$si = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{4.9}$$

Για την σχέση 4.7,

$$L\left\{\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L\right\} \tag{4.10}$$

Άρα,

$$s\omega_m = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L \tag{4.11}$$

Όταν TL=0 η σχέση 4.11 γίνεται:

$$s\omega_m = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m \tag{4.12}$$

Από την σχέση 4.12, λύνοντας ως προς i έχουμε:

$$i = \frac{s\omega_m + \frac{k_f}{J}\omega_m}{\frac{k_t}{J}} \tag{4.13}$$

Και αντικαθιστώντας στην σχέση 4.9

$$\left(\frac{s\omega_m + \frac{k_f}{J}\omega_m}{\frac{k_t}{J}}\right)\cdot\left(s + \frac{R}{L}\right) = -\frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{4.14}$$

Από την σχέση 4.14 έχουμε:

$$\left\{\left(\frac{s^2 J}{k_t} + \frac{sk_f}{k_t} + \frac{sRJ}{k_t L} + \frac{k_f R}{k_t L}\right) + \frac{k_e}{L}\right\}\omega_m = \frac{1}{L}V_s \tag{4.15}$$

Και λύνοντας ως προς V_s:

$$V_s = \left\{\frac{s^2 JL + sk_f L + sRJ + k_f R + k_e k_t}{k_t}\right\}\omega_m \tag{4.16}$$

Ως συνάρτηση μεταφοράς ενός γραμμικού συστήματος ορίζεται ως ο λόγος του μετασχηματισμού Laplace της μεταβλητής που εκφράζει την έξοδο προς τον μετασχηματισμό Laplace της μεταβλητής που εκφράζει την είσοδο. Οπότε έχουμε:

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2 JL + sk_f L + sRJ + k_f R + k_e k_t} \tag{4.17}$$

Από την σχέση 4.17 συνεπάγεται

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2 JL + (k_f L + RJ)s + k_f R + k_e k_t} \tag{4.18}$$

Και λαμβάνοντας υπόψη τις ακόλουθες υποθέσεις:

Θεωρώντας ότι, η σταθερά τριβής είναι πολύ μικρή, δηλαδή εάν το, kf τείνει στο 0, έχουμε;

$RJ \gg K_f L$, και

$k_e k_t \gg R k_f$

Επομένως η συνάρτηση μεταφοράς απλοποιείται στην σχέση 4.19

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2 JL + RJs + k_e k_t} \tag{4.19}$$

Και πολλαπλασιάζοντας τον αριθμητή και τον παρανομαστή με τον όρο $\left\{ \frac{R}{k_e k_t} \times \frac{1}{R} \right\}$,

η σχέση 4.19 γίνεται:

$$G(s) = \frac{\omega_m}{V_s} = \frac{\frac{1}{k_e}}{\frac{RJ}{k_e k_t} \cdot \frac{L}{R} \cdot s^2 + \frac{RJ}{k_e k_t} \cdot s + 1} \tag{4.20}$$

Από την σχέση 4.13, η ακόλουθη ποσότητα ορίζεται ως η μηχανική (χρονική σταθερά),

$$\tau_m = \frac{RJ}{k_e k_t} \tag{4.21}$$

ενώ η ηλεκτρική (χρονική σταθερά),

$$\tau_e = \frac{L}{R} \tag{4.22}$$

Αντικαθιστώντας τις 4.21 και 4.22 στην σχέση 4.20, έχουμε:

$$G(s) = \frac{\frac{1}{K_e}}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \tag{4.23}$$

4.3 Χαρακτηριστικά κινητήρων συνεχούς ρεύματος χωρίς ψήκτρες.

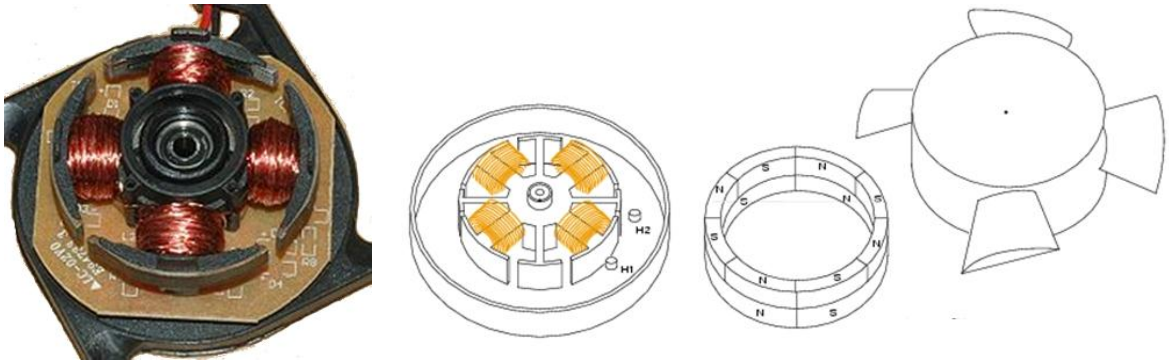
Οι ηλεκτροκινητήρες Brushless συνεχούς ρεύματος (Brushless Direct Current), που από εδώ και στο εξής θα αναφέρονται ως BLDC κινητήρες, διαφέρουν από τους μέχρι σήμερα ηλεκτρικούς κινητήρες συνεχούς ρεύματος, όπως υποδηλώνει και το

όνομα τους (Brush-less), στο γεγονός ότι δεν χρησιμοποιούν ψήκτρες για την μεταγωγή τους. Αντί αυτού η μεταγωγή επιτυγχάνεται ηλεκτρονικά.

Οι κυριότεροι λόγοι που οι κινητήρες BLDC κερδίζουν έδαφος έναντι των κλασσικών ηλεκτροκινητήρων συνεχούς ρεύματος και οφείλονται στην κατασκευαστική τους ιδιαιτερότητα είναι οι εξής:

- Καλύτερα αποτελέσματα απόδοσης ταχύτητας προς ροπή.
- Υψηλά δυναμικά απόκρισης
- Υψηλή αποδοτικότητα.
- Μεγάλη διάρκεια ζωής. Η έλλειψη ψηκτρών και άλλων τριβόμενων μερών πάνω στον ρότορα, έχει ως αποτέλεσμα τριπλάσιο χρόνο λειτουργίας έναντι των κλασσικών ηλεκτροκινητήρων, που μπορεί να φτάσει ακόμη και τις 18000 ώρες λειτουργίας.
- Αθόρυβη λειτουργία.
- Επίτευξη υψηλών στροφών λειτουργίας, φτάνοντας έως και 80.000 στροφές ανά λεπτό.

Οι κινητήρες BLDC είναι σύγχρονου τύπου (sychronous). Αυτό σημαίνει ότι το μαγνητικό πεδίο που παράγεται από τον στάτορα και το μαγνητικό πεδίο που παράγεται από τον ρότορα, περιστρέφονται με την ίδια συχνότητα, είναι δηλαδή συμφασικά. Έτσι οι BLDC κινητήρες δεν παρουσιάζουν το φαινόμενο της ολίσθησης που παρατηρείται σε έναν επαγωγικό κινητήρα. Η κατασκευή τους μπορεί να αποτελείται από μια, δύο ή και τρείς φάσεις. Αντίστοιχα ο στάτορας έχει τον αντίστοιχο αριθμό τυλιγμάτων. Ο ποιό συχνός τύπος είναι εκείνος με τις τρεις φάσεις. Στον ανεμιστήρα που χρησιμοποιήσαμε έχουμε δύο φάσεις. Όπως φαίνεται και στο σχήμα 4.4.



Σχήμα 4.4 – Τέσσερις πόλοι στον στάτορα ενός BLDC κινητήρα δύο φάσεων. Η εικόνα είναι από ανεμιστήρα υπολογιστή από τον οποίο έχει αφαιρεθεί ο ρότορας.

4.4 Μαθηματικό μοντέλο ενός BLDC κινητήρα

Τυπικά, το μαθηματικό μοντέλο ενός BLDC κινητήρα δεν διαφέρει από τους κοινούς DC κινητήρες. Βασική τους διαφορά είναι οι φάσεις, οι οποίες επηρεάζουν τα αποτελέσματα του BLDC μοντέλου. Οι φάσεις επηρεάζουν την αντίσταση και την επαγωγή του BLDC κινητήρα.

Οπότε για τις σχέσεις 4.20 – 4.22, η διαφορά μεταξύ DC και BLDC κινητήρα θα επηρεάσει τις μηχανικές και ηλεκτρικές σταθερές.

Για την μηχανική χρονική σταθερά (με συμμετρική διάταξη), η σχέση 4.21 γίνεται:

$$\tau_m = \sum \frac{RJ}{K_e K_t} = \frac{J \sum R}{K_e K_t} \tag{4.24}$$

Για την ηλεκτρική χρονική σταθερά,

$$\tau_e = \sum \frac{L}{R} = \frac{L}{\sum R} \tag{4.25}$$

Επομένως, δεδομένου ότι υπάρχει μια συμμετρική διάταξη δύο φάσεων, η μηχανική και η ηλεκτρική σταθερά μετασχηματίζονται σε:
Μηχανική σταθερά,

$$\tau_m = \frac{J \cdot 2R}{K_e K_t} \tag{4.26}$$

Και η ηλεκτρική σταθερά,

$$\tau_e = \frac{L}{2 \cdot R} \tag{4.27}$$

Επομένως έχουμε

$$\tau_m = \frac{2 \cdot R_{\varnothing} \cdot J}{(K_{e(L-L)} / \sqrt{2}) \cdot K_t} \tag{4.28}$$

Όπου Ke η φασική τιμή της αντιηλεκτρεργετικής σταθεράς (τάσης);

$$K_e = K_{e(L-L)} / \sqrt{2} \tag{4.29}$$

Επίσης, υπάρχει μια σχέση μεταξύ των Ke και Kt. Χρησιμοποιώντας την ηλεκτρική ισχύ (κανόνας του αριστερού χεριού) και την μηχανική ισχύ (κανόνας δεξιού χεριού) προκύπτει:

$$\sqrt{2} \times E \times I = \frac{2\pi}{60} \times N \times T$$

$$\frac{E}{N} = \frac{T}{I} \times \frac{2\pi \times 1}{60 \times \sqrt{2}}$$

$$K_e = K_t \times \frac{2\pi \times 1}{60 \times \sqrt{2}}$$

$$K_e = K_t \times 0.074 \tag{4.30}$$

Όπου,

$K_e = \left[\frac{v - \sec s}{rad} \right]$:η ηλεκτρική ροπή

$K_t = \left[\frac{N - m}{A} \right]$:η σταθερά ροπής

Οπότε η συνάρτηση μεταφοράς ενός BLDC κινητήρα είναι ίδια με την σχέση 4.23, λαμβάνοντας υπόψη ότι οι σταθερές επηρεάζονται λόγω των δύο φάσεων.

$$G(s) = \frac{\frac{1}{K_e}}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \tag{4.31}$$

Παρατηρούμε ότι καταλήξαμε σε μια συνάρτηση μεταφοράς με δύο πόλους και κανένα μηδενικό, η τελική συνάρτηση μεταφοράς όμως που θα προσομοιώσουμε σε επόμενη υποενότητα μπορεί να διαφέρει αρκετά από αυτό το βασικό μαθηματικό μοντέλο, αφού δεν έχουμε υπολογίσει το φορτίο του κινητήρα (φτερωτή του ανεμιστήρα στην περίπτωση του μοντέλου που θέλουμε να μελετήσουμε) όπως και την επίδραση στο μοντέλο μας της ύπαρξης του αισθητήρα του φαινομένου hall όσο αφορά το μαθηματικό μοντέλο της πειραματικής μας διάταξης, αφού με την χρήση ενός τέτοιου αισθητήρα γίνεται ο υπολογισμός των στροφών ανά λεπτών του ανεμιστήρα.

4.5 Χειροκίνητη ρύθμιση του ελεγκτή – Manual tuning

Το manual tuning του πειραματικού ελέγχου έγινε ακολουθώντας κάποιους βασικούς κανόνες οι οποίοι ορίζουν τον τρόπο αλλαγής των παραμέτρων για το σωστό tuning. Οι κανόνες και η διαδικασία είναι τα εξής:

Η μέθοδος που ακολουθείται είναι αυτή του κλειστού βρόχου μέσω της απόκρισης στον χρόνο σε βηματικές μεταβολές της επιθυμητής τιμής.

Αρχικά γίνεται μια αλλαγή της επιθυμητής τιμής (u) του ελέγχου. Στην συνέχεια παρατηρούμε τις αλλαγές στην εσωτερική μεταβλητή PWM και στην έξοδο (στροφές του κινητήρα).

1. Αν η έξοδος δεν αλλάζει αρκετά ή δεν έχουμε την επιθυμητή υπερύψωση ή ταχύτητα απόκρισης, τότε αυξάνουμε το αναλογικό κέρδος kp κατά 50%.
2. Αν η εσωτερική μεταβλητή PWM έχει σταθερή ταλάντωση ή γενικότερα ταλάντωση και η υπερύψωση ξεπερνά το 25%, τότε μειώνουμε το αναλογικό κέρδος kp κατά 50% και το ολοκληρωτικό κέρδος ki κατά 50%.

3. Αν η ταλάντωση της εσωτερικής μεταβλητής παραμένει, όπως και η υπερύψωση, τότε μειώνουμε το αναλογικό κέρδος kp κατά 20% και το ολοκληρωτικό ki κατά 50%.
4. Αν συμβαίνουν πάνω από 3 κορυφώσεις κατά την αλλαγή της επιθυμητής τιμής, τότε μειώνουμε το ολοκληρωτικό κέρδος ki κατά 30% και αυξάνουμε το διαφορικό κέρδος kd κατά 50%.
5. Αν η τιμή της εξόδου παραμένει για μεγάλο χρονικό διάστημα πιο κάτω από την επιθυμητή τιμή (long tail scenario), τότε αυξάνουμε το ολοκληρωτικό κέρδος ki κατά 100%.
6. Αν η τιμή της εξόδου παραμένει για μεγάλο χρονικό διάστημα πιο κάτω από την επιθυμητή τιμή (long tail scenario), χωρίς να υπάρχει ιδιαίτερη ταλάντωση, τότε αυξάνουμε το ολοκληρωτικό κέρδος ki κατά 100% και μειώνουμε το διαφορικό kd κατά 100%.

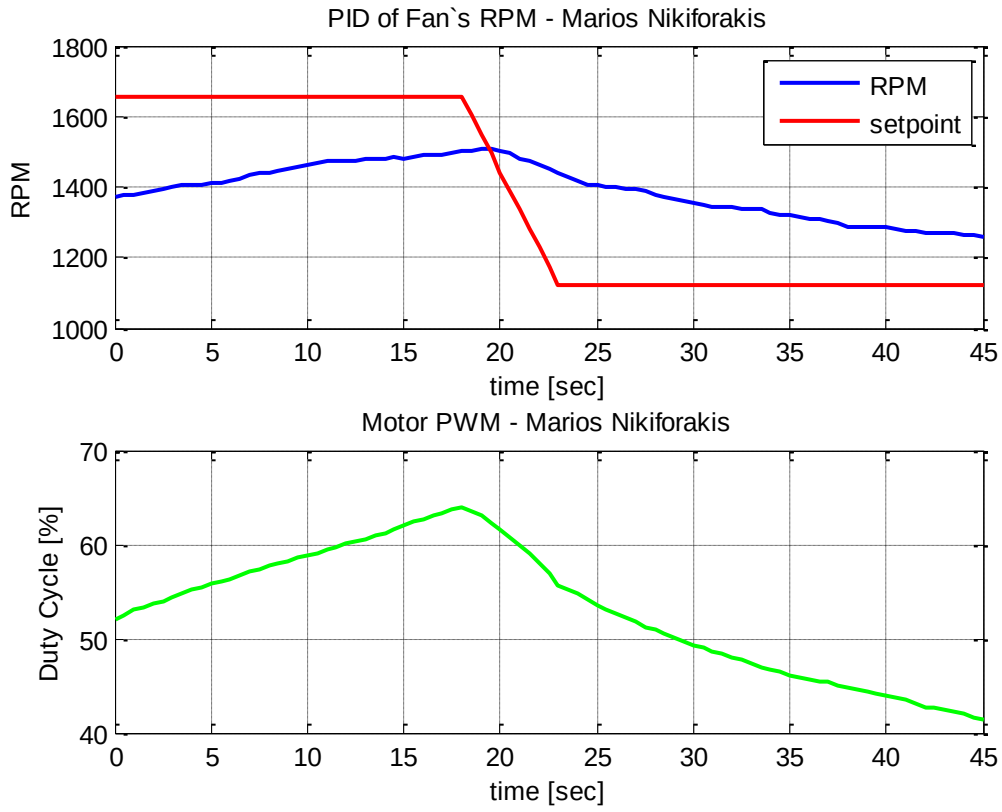
Στην συνέχεια επαναλαμβάνεται η διαδικασία από την αρχή μέχρι να πετύχουμε το επιθυμητό αποτέλεσμα.

Αρχικά ξεκινάμε από τις εξής τιμές για τον ελεγκτή :

Βήμα 1:

Kp = 0.04, Ki = 0.01, Kd = 0.01

Τα αποτελέσματα που προέκυψαν φαίνονται στο σχήμα 4.5. Εύκολα διακρίνεται πως η τιμή της εξόδου αλλάζει με πολύ αργό ρυθμό.



Σχήμα 4.5 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.04, Ki=0.01, Kd=0.01

Λόγο του κανόνα 1 ο οποίος φαίνεται πως πρέπει να εφαρμοστεί στην περίπτωση αυτή γίνεται η αλλαγή που απαιτείται, δηλαδή αλλαγή του Kp κατά 50%. Έτσι έχουμε τις τιμές :

Βήμα 2:

Kp = 0.06, Ki = 0.01, Kd = 0.01

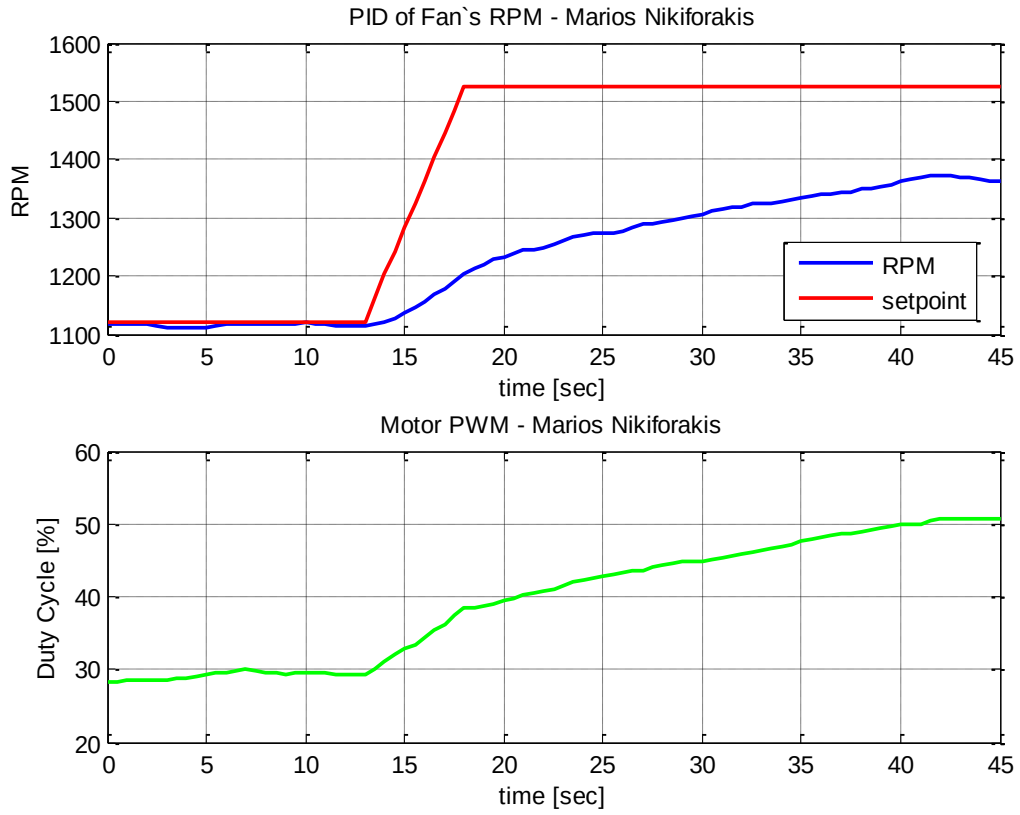
Στο σχήμα 4.6 βλέπουμε την απόκριση του βήματος 2, στο οποίο βλέπουμε πως το σύστημα παραμένει εξίσου αργό.

Έτσι, λόγο του κανόνα 1 ο οποίος φαίνεται πως πρέπει να εφαρμοστεί πάλι στην περίπτωση αυτή γίνεται πάλι αύξηση του Kp κατά 50%.

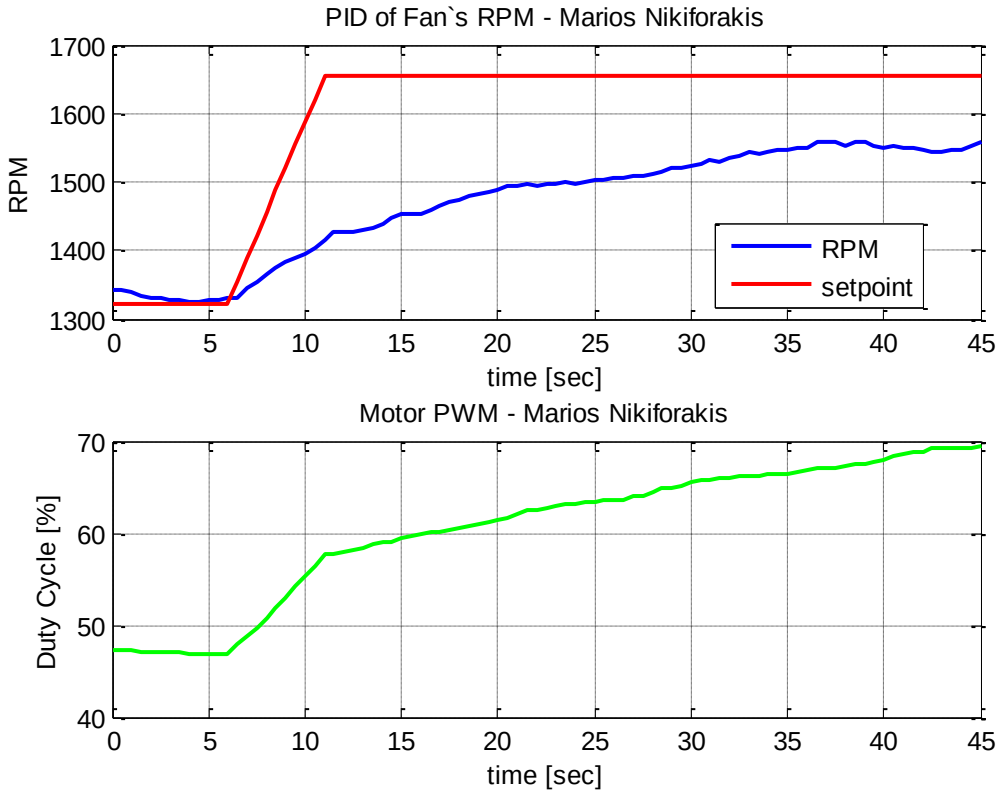
Βήμα 3:

Kp = 0.09, Ki = 0.01, Kd = 0.01

Τα αποτελέσματα του ελεγκτή του βήματος 3 φαίνεται στο σχήμα 4.7.



Σχήμα 4.6 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.06, Ki=0.01, Kd=0.01



Σχήμα 4.7 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.09, Ki=0.01, Kd=0.01

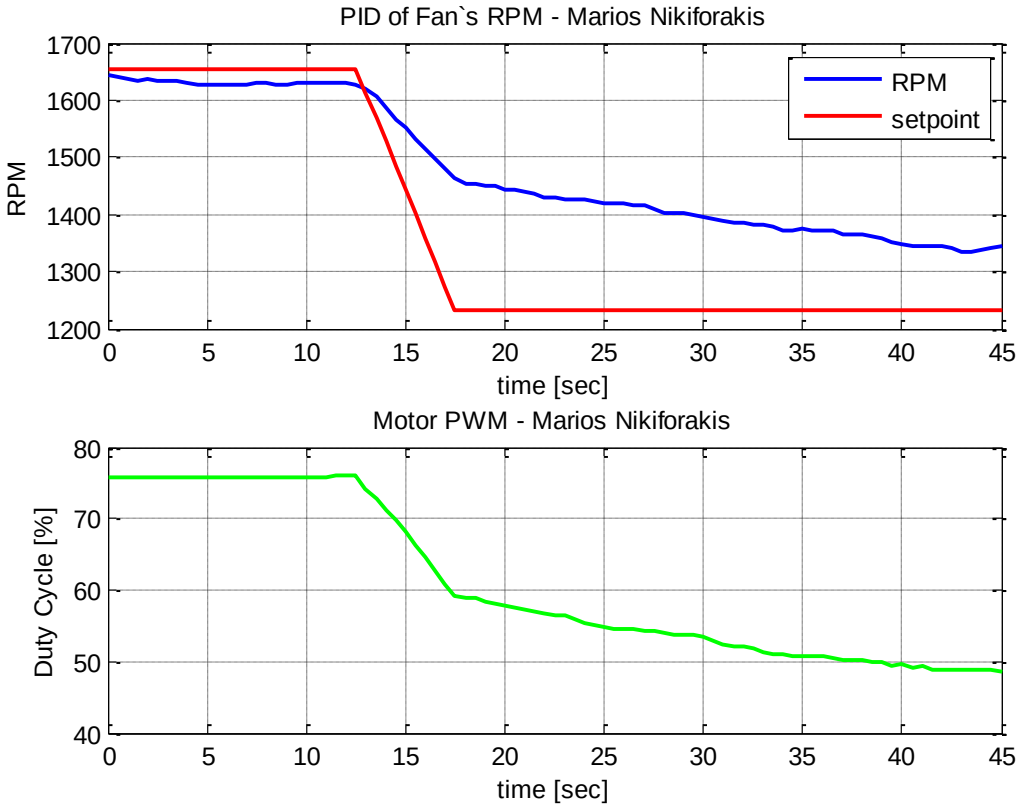
Στο σχήμα 4.8 βλέπουμε την απόκριση του βήματος 3, στο οποίο βλέπουμε πως το σύστημα παραμένει ακόμα αρκετά αργό.

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 1 ακόμα μια φορά. Έτσι αυξάνουμε το kp κατά 50%. Έτσι έχουμε τις τιμές:

Βήμα 4:

Kp = 0.13, Ki = 0.01, Kd = 0.0.1

Στο σχήμα 5.34 βλέπουμε την απόκριση του βήματος 4, στο οποίο βλέπουμε πως στο σύστημα υπάρχει το φαινόμενο του long tail.



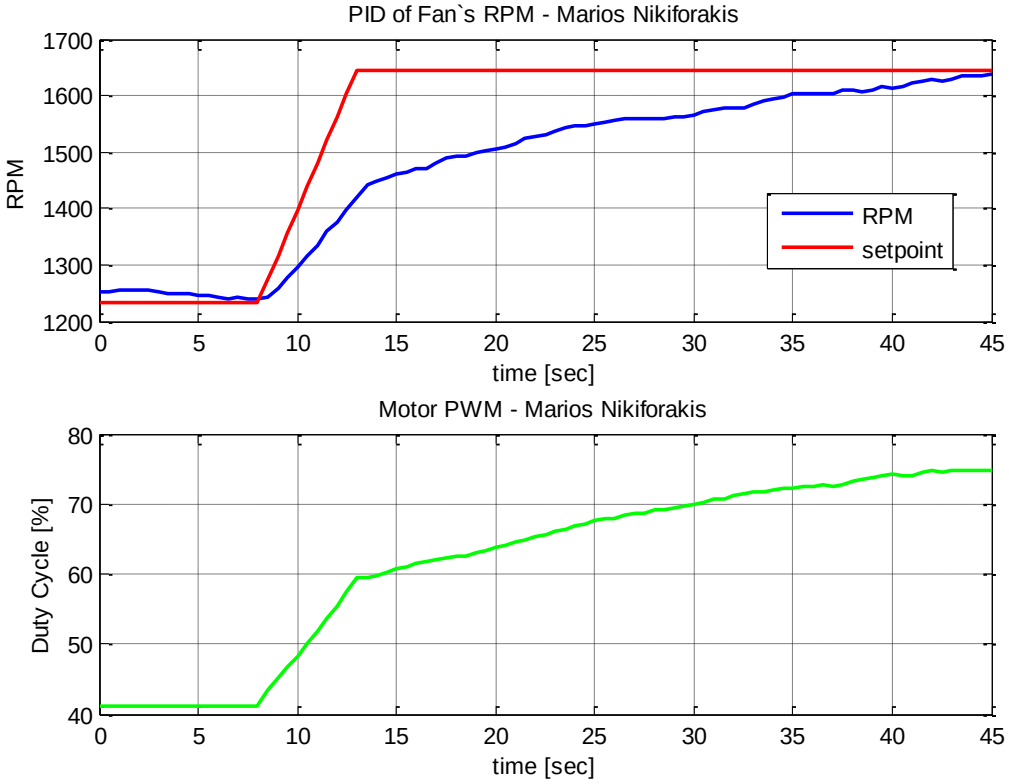
Σχήμα 4.8 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.14, Ki=0.01, Kd=0.01

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 5. Έτσι αυξάνουμε το ki κατά 100%. Έτσι έχουμε τις τιμές:

Βήμα 5:

Kp = 0.13, Ki = 0.02, Kd = 0.0.1

Στο σχήμα 4.9 βλέπουμε την απόκριση του βήματος 5, στο οποίο βλέπουμε πως στο σύστημα υπάρχει το φαινόμενο του long tail, αλλά έχει μειωθεί σε κάποιο βαθμό.



Σχήμα 4.9 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.14, Ki=0.02, Kd=0.01

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 5. Έτσι αυξάνουμε το ki κατά 100%. Έτσι έχουμε τις τιμές:

Βήμα 6:

Kp = 0.13, Ki = 0.04, Kd = 0.0.1

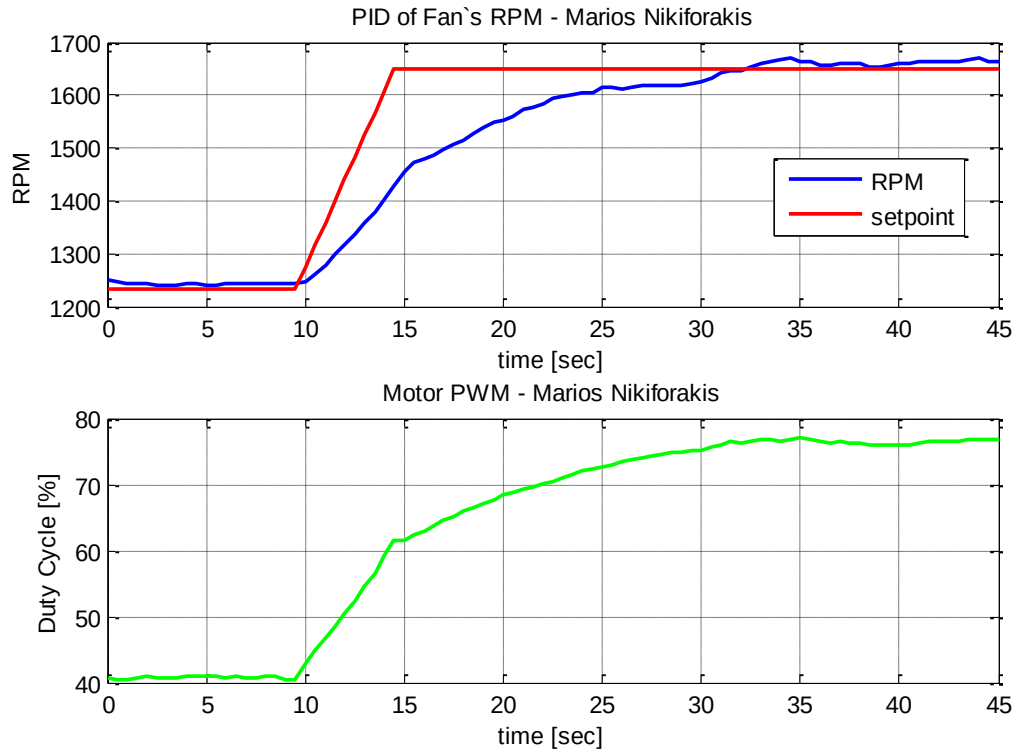
Στο σχήμα 4.10 βλέπουμε την απόκριση του βήματος 6, στο οποίο βλέπουμε πως στο σύστημα έχει μειωθεί αρκετά το φαινόμενο του long tail, αλλά το σύστημα παραμένει αρκετά αργό ακόμα.

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 1. Για αυτόν το λόγο αυξάνουμε το kp κατά 50%. Έτσι έχουμε τις τιμές:

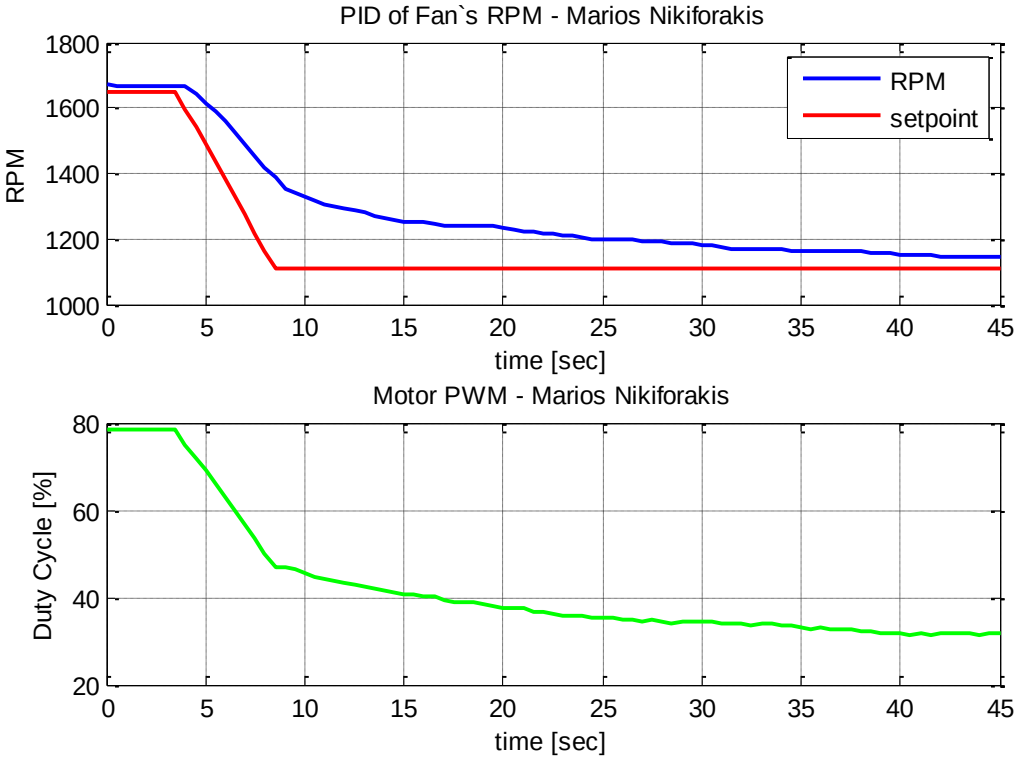
Βήμα 7:

Kp = 0.2, Ki = 0.04, Kd = 0.0.1

Στο σχήμα 4.11 βλέπουμε την απόκριση του βήματος 7, στο οποίο βλέπουμε πως το σύστημα έχει αρκετά πιο γρήγορο αλλά αυξήθηκε το φαινόμενο του σταθερού σφάλματος.



Σχήμα 4.10 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.14, Ki=0.04, Kd=0.01



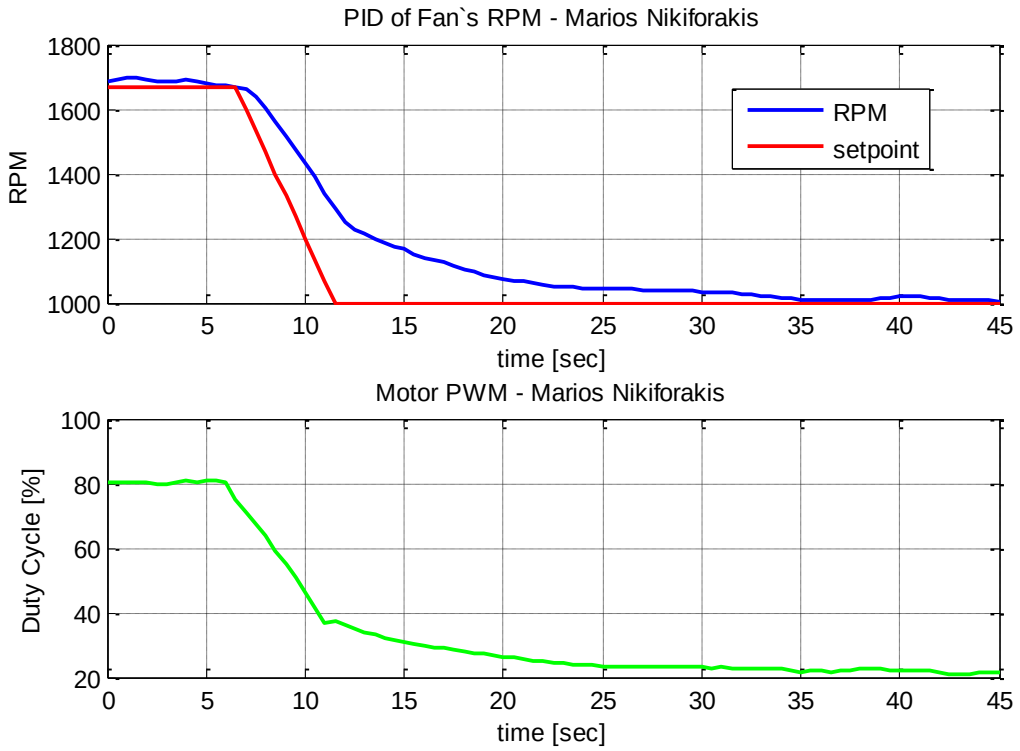
Σχήμα 4.11 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.2, Ki=0.04, Kd=0.01

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 5. Για αυτόν το λόγο αυξάνουμε το ki κατά 100%. Έτσι έχουμε τις τιμές:

Βήμα 8:

Kp = 0.2, Ki = 0.08, Kd = 0.0.1

Στο σχήμα 4.12 βλέπουμε την απόκριση του βήματος 8, στο οποίο βλέπουμε πως στο σύστημα έχει μειωθεί το φαινόμενο του σταθερού σφάλματος, αλλά ακόμα παραμένει αρκετά αργό.



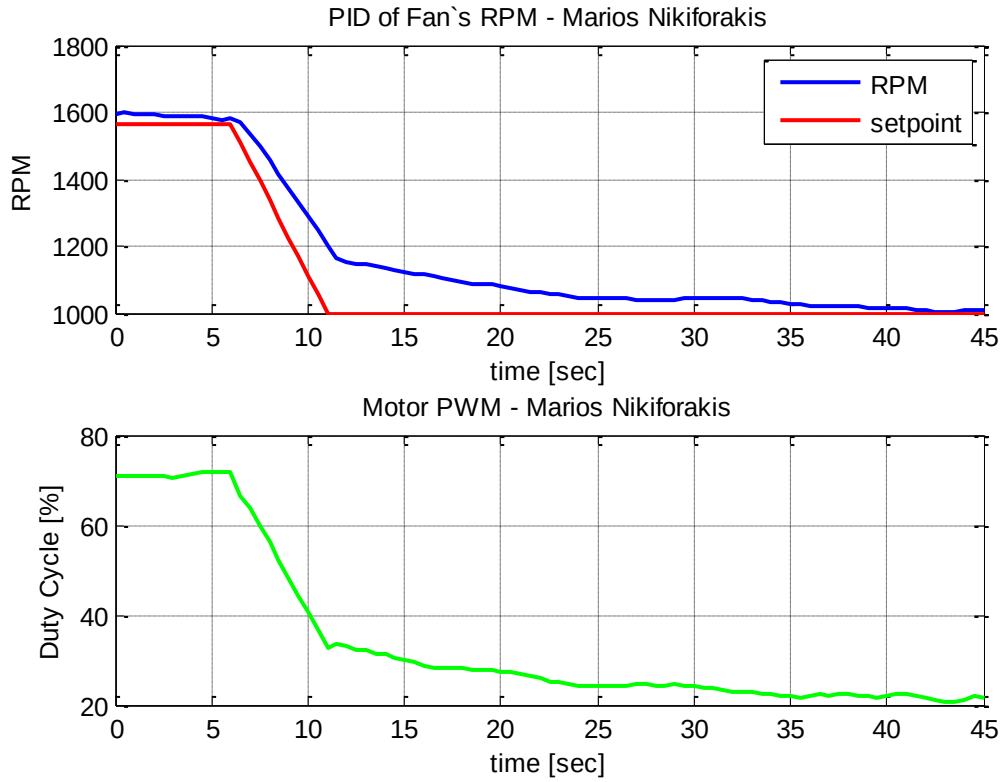
Σχήμα 4.12 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.2, Ki=0.08, Kd=0.01

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 1. Για αυτόν το λόγο αυξάνουμε το kp κατά 50%. Έτσι έχουμε τις τιμές:

Βήμα 9:

Kp = 0.3, Ki = 0.08, Kd = 0.0.1

Στο σχήμα 4.13 βλέπουμε την απόκριση του βήματος 9, στο οποίο βλέπουμε πως το σύστημα έχει αρκετά πιο γρήγορη απόκριση αλλά αυξήθηκε το φαινόμενο του σταθερού σφάλματος.



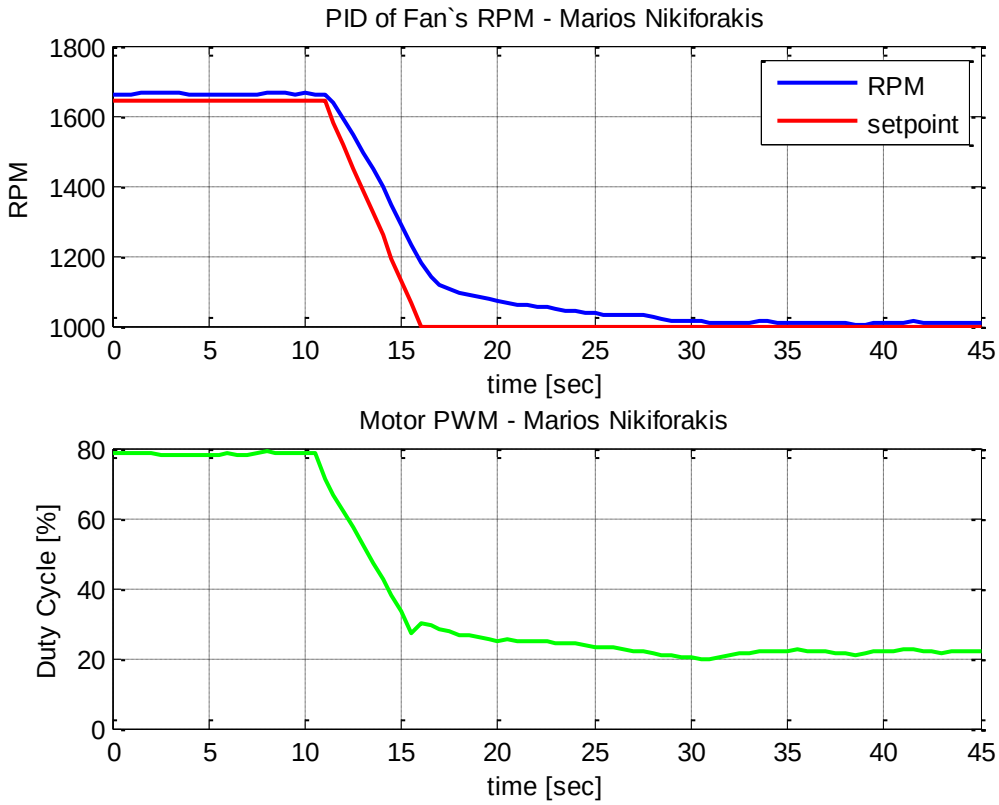
Σχήμα 4.13 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.3, Ki=0.08, Kd=0.01

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 5. Για αυτόν το λόγο αυξάνουμε το ki κατά 100%. Έτσι έχουμε τις τιμές:

Βήμα 10:

Kp = 0.3, Ki = 0.16, Kd = 0.0.1

Στο σχήμα 4.14 βλέπουμε την απόκριση του βήματος 10, στο οποίο βλέπουμε πως στο σύστημα έχει μειωθεί το φαινόμενο του σταθερού σφάλματος, αλλά μπορεί να βελτιωθεί από άποψη ταχύτητας.



Σχήμα 4.14 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.3, Ki=0.16, Kd=0.01

Έτσι μπορούμε να αυξήσουμε το κέρδος kp κατά 50%.

Βήμα 11:

Kp = 0.45, Ki = 0.16, Kd = 0.0.1

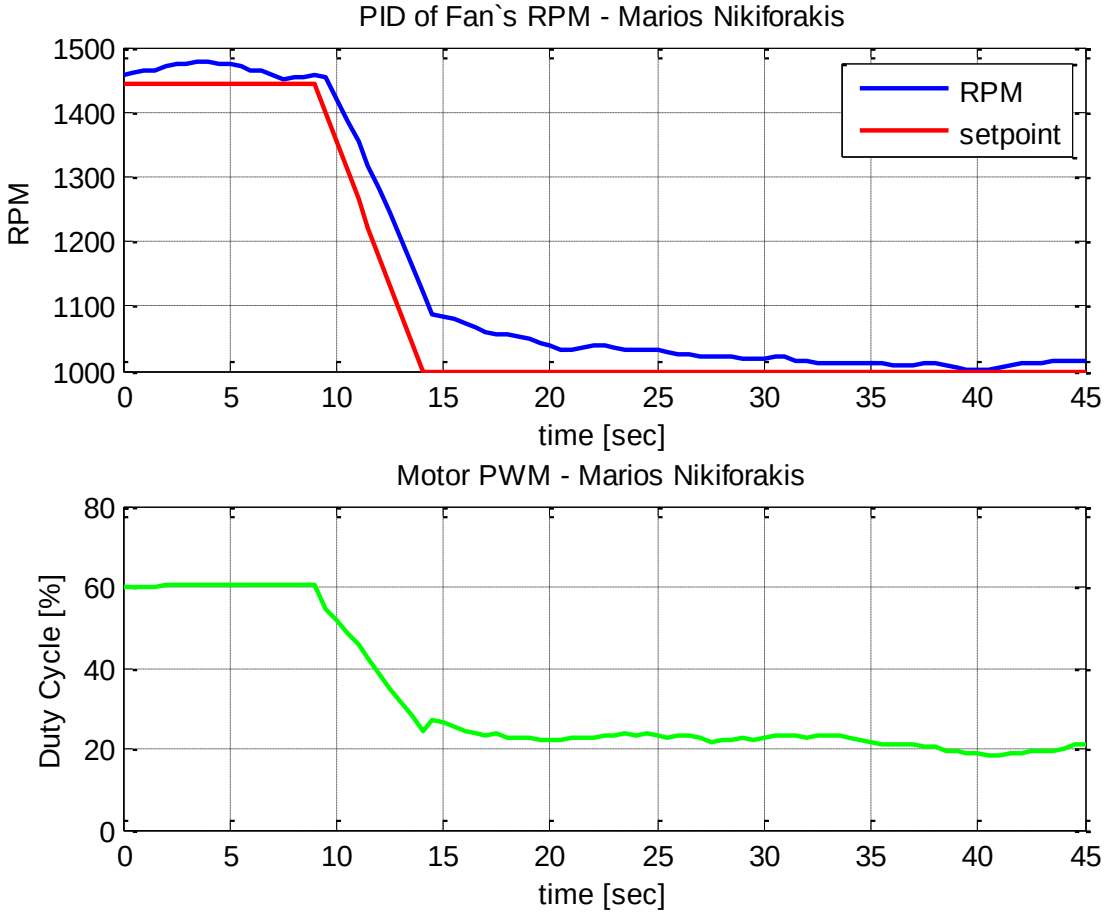
Στο σχήμα 4.15 βλέπουμε την απόκριση του βήματος 11, στο οποίο βλέπουμε πως το σύστημα έχει αρκετά πιο γρήγορη απόκριση αλλά αυξήθηκε το φαινόμενο του σταθερού σφάλματος.

Έτσι φαίνεται να χρειάζεται να εφαρμοστεί ο κανόνας 5. Για αυτόν το λόγο αυξάνουμε το ki κατά 100%. Έτσι έχουμε τις τιμές:

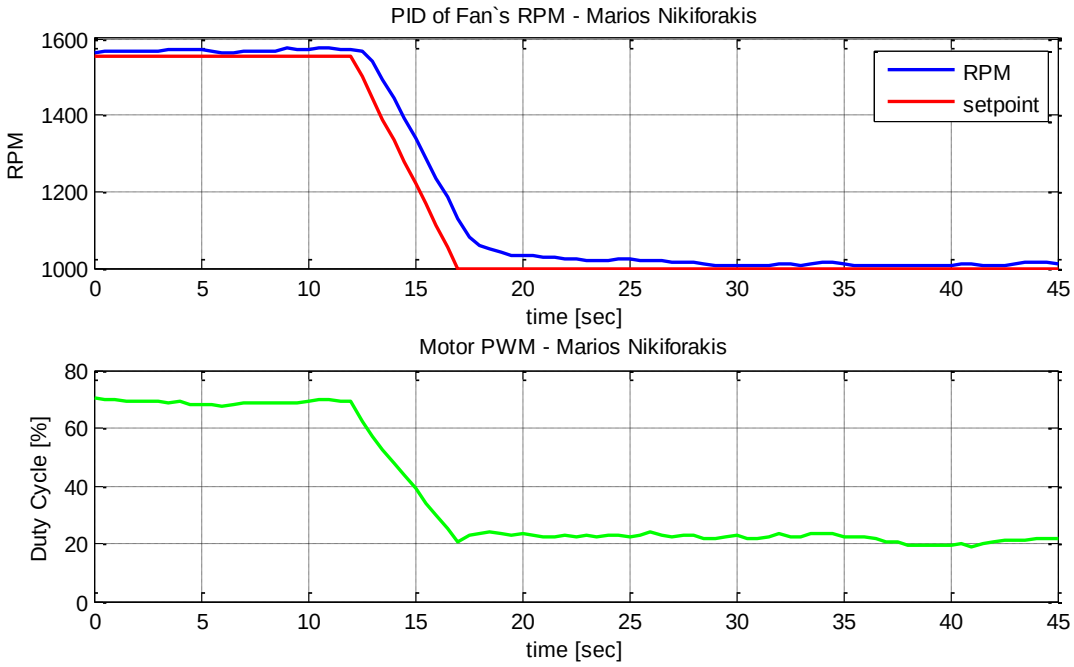
Βήμα 12:

Kp = 0.45, Ki = 0.32, Kd = 0.0.1

Στο σχήμα 4.16 βλέπουμε την απόκριση του βήματος 12, στο οποίο βλέπουμε πως στο σύστημα έχει μειωθεί το φαινόμενο του σταθερού σφάλματος, αλλά μπορεί να βελτιωθεί από άποψη ταχύτητας.



Σχήμα 4.15 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.45, Ki=0.16, Kd=0.01

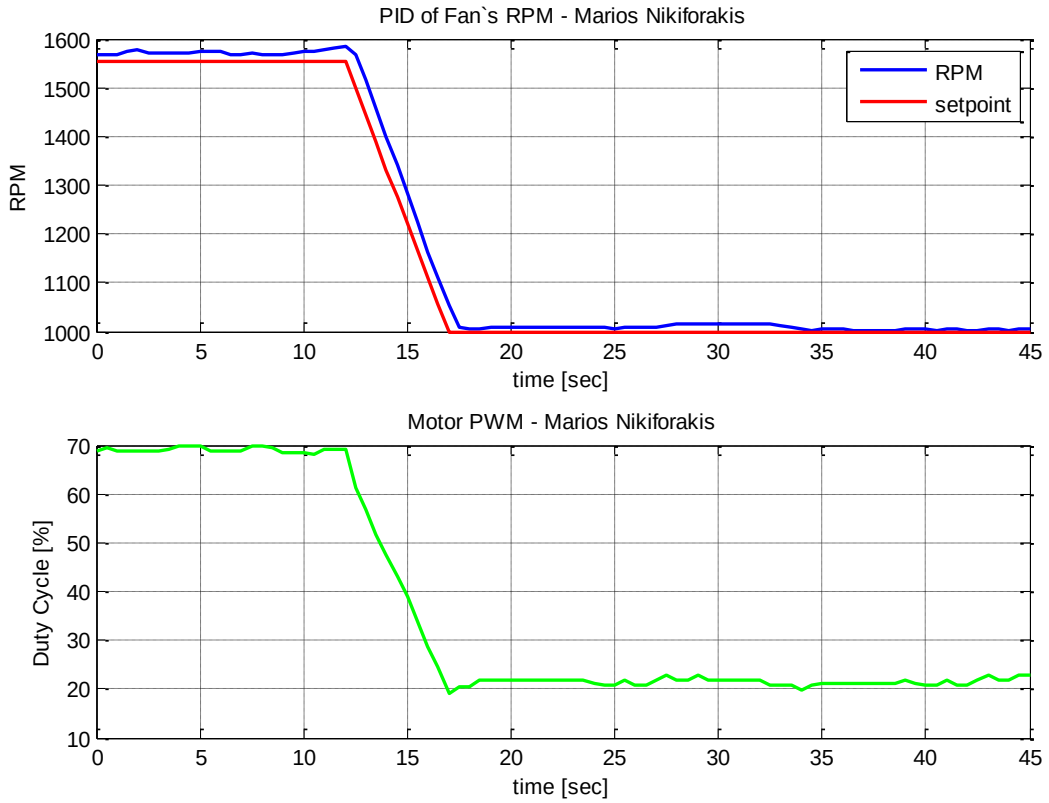


Σχήμα 4.16 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.45, Ki=0.32, Kd=0.01

Εκτελώντας διαδοχικά τα βήματα 1 και 5 όπως παραπάνω, καταλήγουμε στην απόκριση του σχήματος 4.17 του οποίου οι παράμετροι έχουν τις τιμές:

Βήμα 13:

Kp = 0.67, Ki = 0.64, Kd = 0.0.1 (καλύτερα αποτελέσματα)

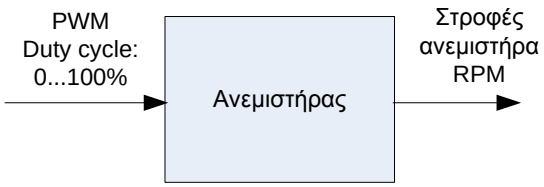


Σχήμα 4.17 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.67, Ki=0.64, Kd=0.01

Το βήμα 13 αποτελεί και το τελευταίο βήμα αφού αλλάζοντας έστω και λίγο τις παραπάνω παραμέτρους προέκυπτε χειρότερο αποτέλεσμα, καθώς επετεύχθητε αρκετά καλός έλεγχος χωρίς ταλαντώσεις.

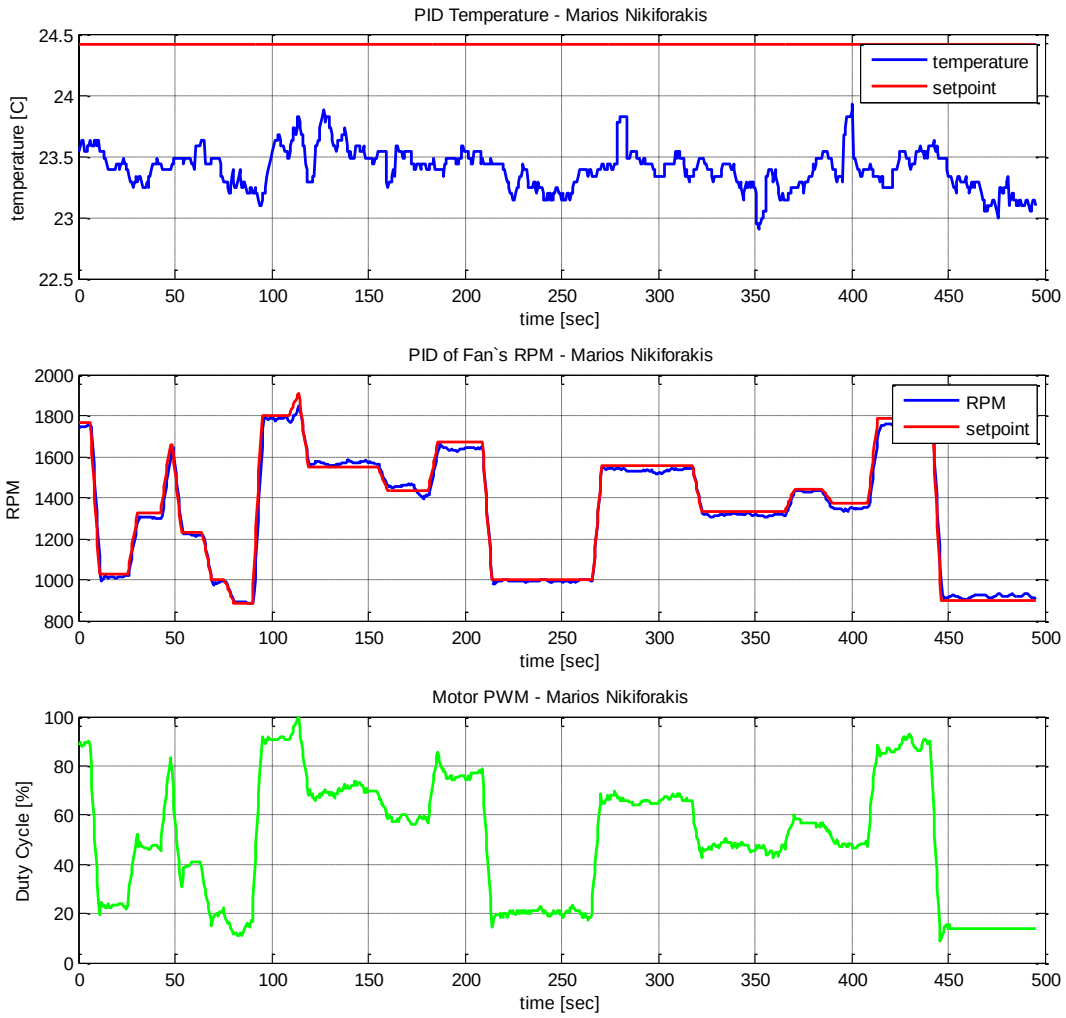
4.6 Εκτίμηση - Μοντελοποίηση συστήματος ανεμιστήρα

Για την μοντελοποίηση του ανεμιστήρα θα ακολουθηθεί η μέθοδος που αναλύεται παρακάτω. Στόχος μας είναι να εξάγουμε πληροφορία για το άγνωστο προς εμάς σύστημα, που είναι σε αυτό το στάδιο το σύστημα του ανεμιστήρα. Έτσι θέλουμε να βρούμε το μαθηματικό μοντέλο που εκφράζει την σχέση της τάσης εισόδου - εντολής σε PWM που δίνουμε προς τον κινητήρα με τις στροφές του RPM, όπως φαίνεται στο σχήμα 4.18



Σχήμα 4.18 – Διάγραμμα βαθμίδων ανοιχτού βρόγχου έλεγχου στροφών

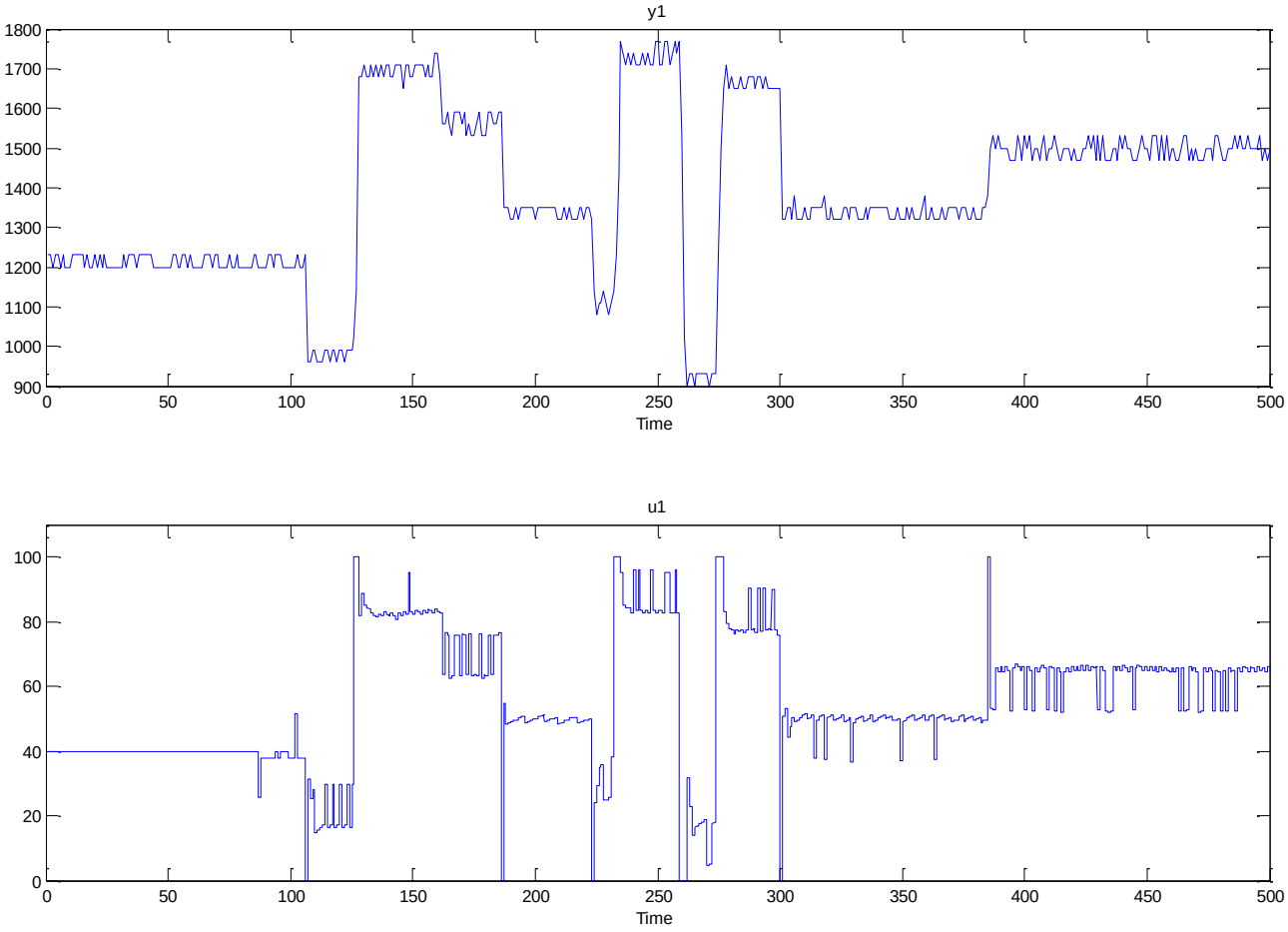
Αρχικά καταγράφουμε τις τιμές των στροφών του ανεμιστήρα δίνοντας συγκεκριμένες τιμές PWM, οι αλλαγές των τιμών του PWM έγιναν ενώ είχαμε συνδεμένο τον PID (σχήμα 4.1), αλλά από την στιγμή που μελετάμε ως είσοδο τον παλμό PWM και ως έξοδο τις στροφές RPM είναι σαν να μελετάμε τον ανοιχτό βρόγχο όπως φαίνεται στο σχήμα 4.18. Έχοντας λοιπόν ρυθμίσει τον ελεγκτή με τον τρόπο που περιγράψαμε στην προηγούμενη ενότητα, αλλάζουμε διαδοχικά το εύρος των παλμών (Duty Cycle) PWM μεταβάλλοντας το setpoint, και αποθηκεύουμε την μεταβολή της εξόδου του συστήματος μας, δηλαδή τις στροφές RPM. Οι μετρήσεις δε θα είναι χρονικά ανεξάρτητες, αφού όλα τα πραγματικά συστήματα έχουν κάποια μη πεπερασμένου χρόνου απόκριση. Το πείραμα θα πρέπει να είναι μεγάλου χρονικού διαστήματος και να περιέχει αρκετές περιπτώσεις εναλλαγής των μεγεθών. Το set δεδομένων αυτό αποτελεί, όπως θα δούμε και παρακάτω το set εκμάθησης της μοντελοποίησης. Στο σχήμα 4.19 βλέπουμε τα γραφήματα που αποτελούν αυτό το set δεδομένων.



Σχήμα 4.19 – Σετ εκπαίδευσης για την μοντελοποίηση. Διαγράμματα από μετρήσεις πειράματος ελέγχου στροφών κινητήρα. Mode '0'

Από το σχήμα 4.19 μπορούμε να βγάλουμε το συμπέρασμα πως ο έλεγχος θερμοκρασίας δε λειτουργεί, λογικό αφού βρισκόμαστε στο mode "0" όπου έχουμε μόνο έλεγχο στροφών κινητήρα. Επίσης βλέπουμε πως γίνεται μια ικανοποιητική μορφή ελέγχου εκτός από τις φάσεις που η επιθυμητή τιμή βγαίνει εκτός εύρους του κινητήρα. Τα δεδομένα είναι συγχρονισμένα και θα πρέπει να σημειωθεί πως έχουν φιλτραριστεί μέσω του φίλτρου Moving Average που αναλύεται στο Κεφάλαιο 3. Το φίλτρο επιλέχθηκε ως MA – 10, δηλαδή δέκα συντελεστών. Ο συνολικός χρόνος λήψης των μετρήσεων είναι 8.3 λεπτά, αρκετά μεγάλο χρονικό διάστημα για την μελέτη της απόκρισης του ανεμιστήρα (γρήγορες αντιδράσεις).

Ως testing set χρησιμοποιήθηκε μέτρηση που έγινε σε άλλο χρονικό διάστημα, έπειτα από 24 ώρες. Το testing set που χρησιμοποιήθηκε φαίνεται στο σχήμα 4.20.



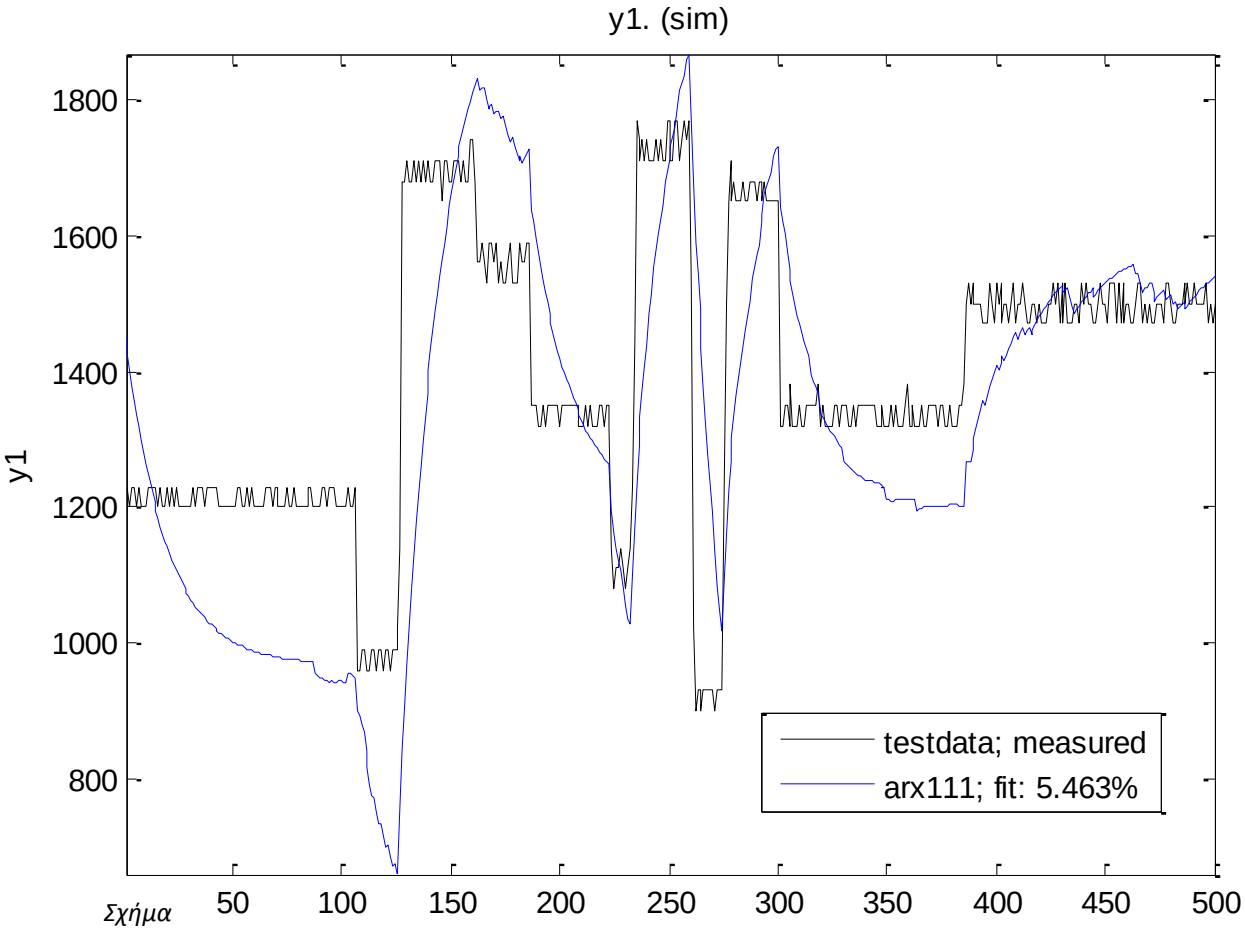
Σχήμα 4.20 – Testing set για την μοντελοποίηση.

Στην συνέχεια εισάγουμε τα δεδομένα που καταγράφθηκαν από το πείραμα στο "identification toolbox" του Matlab.

Στα δεδομένα δε χρειάζεται προ – επεξεργασία. Έτσι εισάγοντας σαν είσοδο τις πραγματικές τιμές του Duty cycle του PWM και σαν έξοδο τις πραγματικές στροφές του κινητήρα (training set σχήμα 4.19), εξάγονται τα εξής μοντέλα:

4.6.1 Γραμμικό μοντέλο, ARX:

Για **na = 1** (αριθμός πόλων), **nb = 1** (αριθμός μηδενικών)[1] και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.21, δίνοντας του το testing set ως είσοδο.

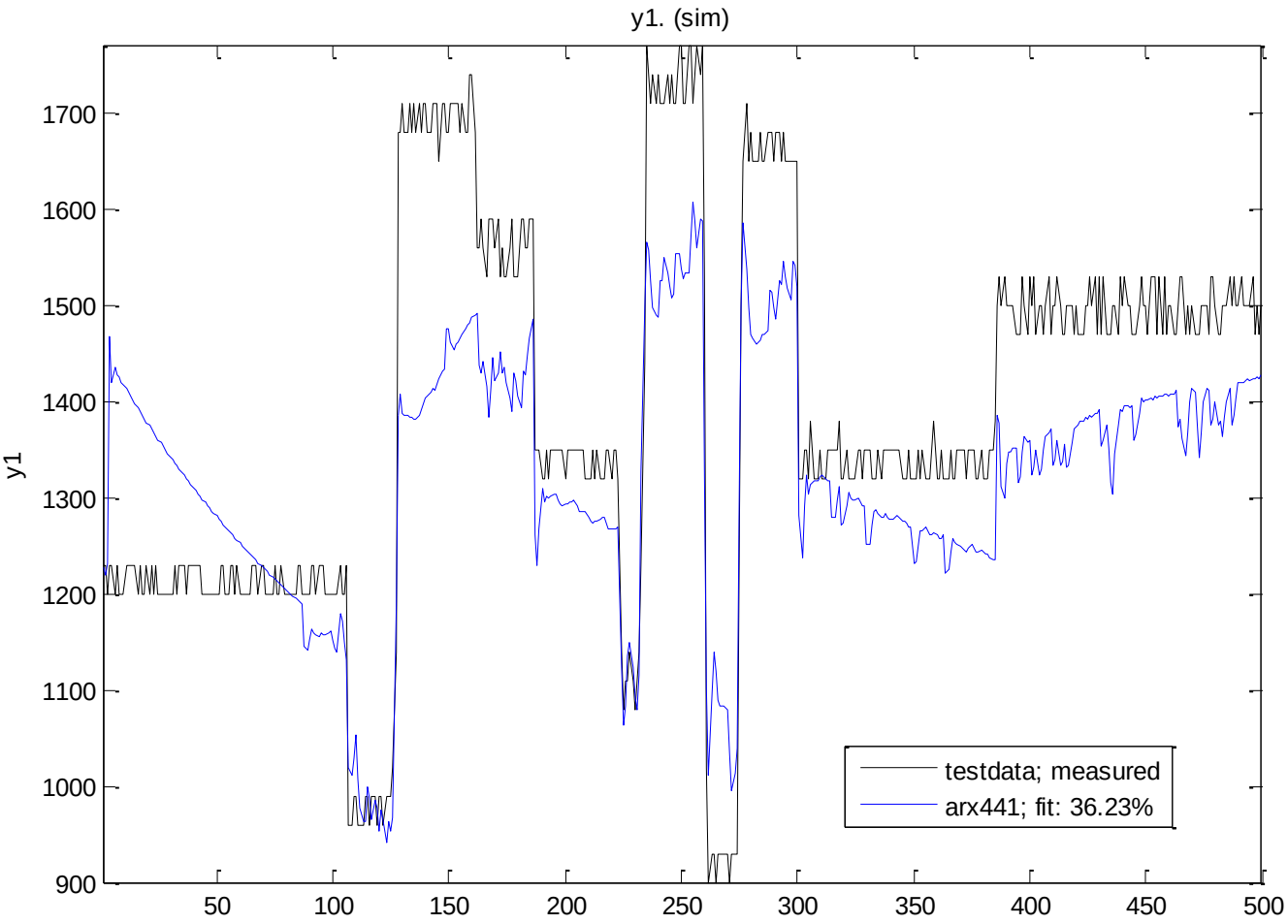


Σχήμα 4.21 – Αποτελέσματα μοντελοποίηση ARX na=1,nb=1, no delay

Όπως φαίνεται αποτελεί μια κακή μοντελοποίηση, αφού έχουμε fit: 5.463% ομοιότητα των πειραματικών δεδομένων με αυτά που εξήχθησαν από το θεωρητικό μοντέλο.

[1] Οι παράμετροι na, nb, nk όπως και όλα τα πιθανά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

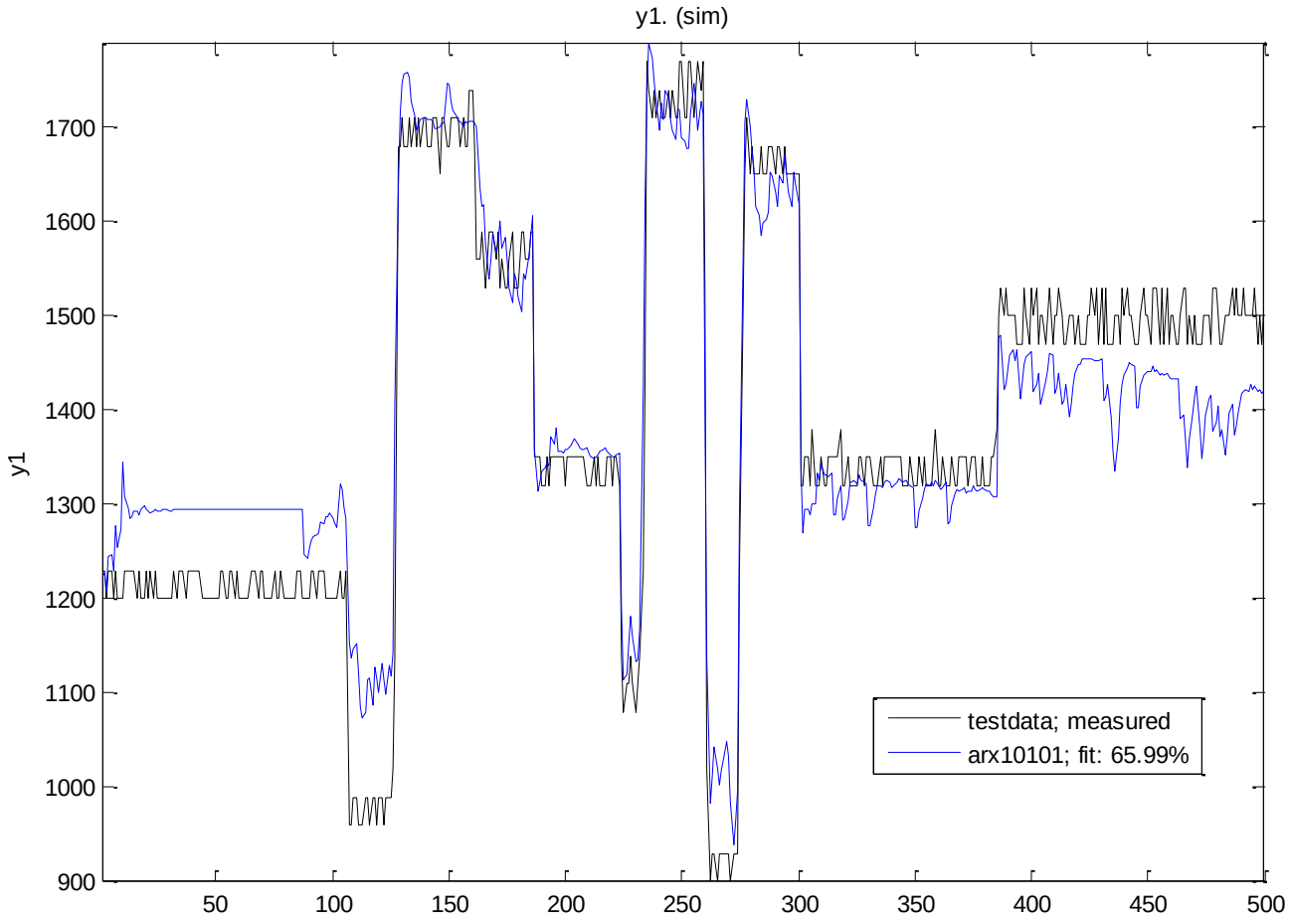
Για **na = 4** (αριθμός πόλων), **nb = 4** (αριθμός μηδενικών) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.22, δίνοντας του το testing set ως είσοδο.



Σχήμα 4.22 – Αποτελέσματα μοντελοποίηση ARX na=4,nb=4, no delay

Φαίνεται πως αποτελεί μια καλύτερη από την προηγούμενη μοντελοποίηση, αφού έχουμε fit: 36.23% ομοιότητα με τα πειραματικά δεδομένα, αλλά πάλι δεν αποτελεί πάρα πολύ καλή προσέγγιση, αφού ταιριάζουν κατά λιγότερο από 50%.

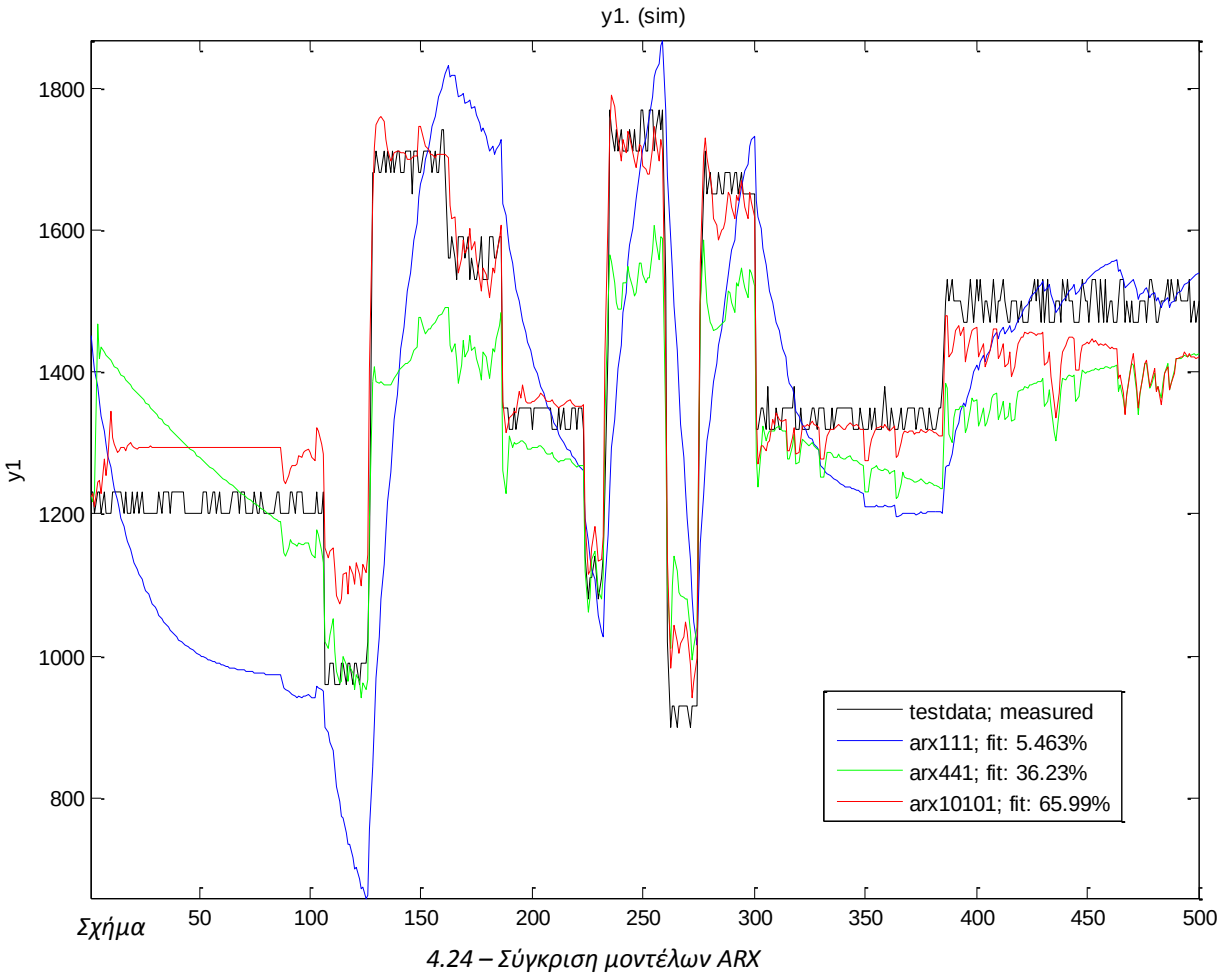
Για **na = 10** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.23, δίνοντας του το testing set ως είσοδο.



Σχήμα 4.23 – Αποτελέσματα μοντελοποίηση ARX na=10,nb=10, no delay

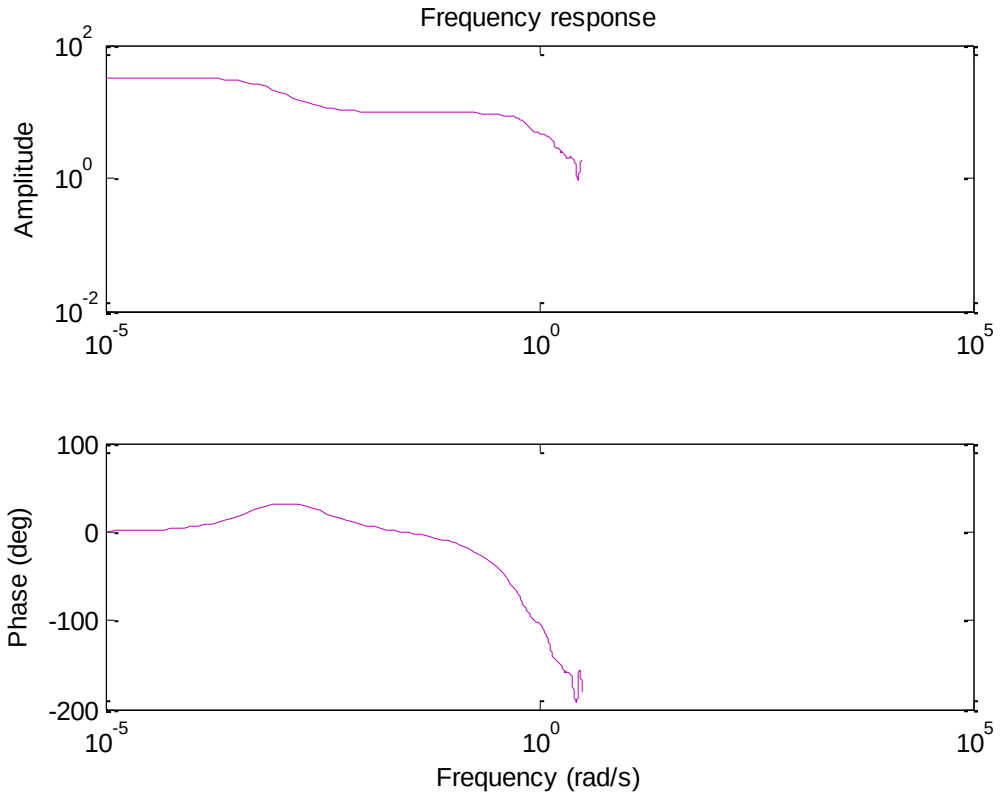
Εδώ είναι ξεκάθαρο πως το θεωρητικό μοντέλο προσεγγίζει αρκετά καλά το πραγματικό μοντέλο, αφού τα αποτελέσματα ταιριάζουν κατά fit: 65.99%. Θα πρέπει να σημειωθεί πως αυξάνοντας τον αριθμό των πόλων και μηδενικών πάνω από 10 δεν υπήρχε ουσιαστική αλλαγή στα δεδομένα.

Έτσι μπορούμε σε αυτό το στάδιο να συγκρίνουμε τα μοντέλα που εξήχθησαν μέσω του σχήματος 4.24, όπου φαίνεται και οπτικά η διαφορά μεταξύ των μοντέλων.

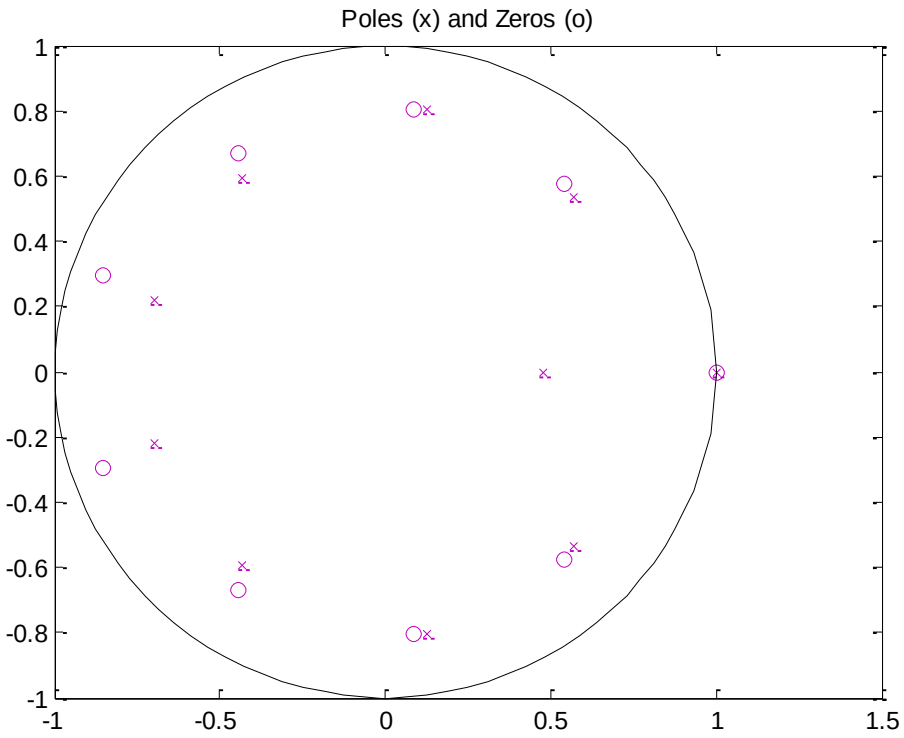


Σχήμα 4.24 – Σύγκριση μοντέλων ARX

Σε αυτό το σημείο μπορούμε να πούμε πως θα επιλέγαμε το μοντέλο ARX 10 10 1 από τα μοντέλα που εξάγονται τύπου ARX. Η συμπεριφορά του συγκεκριμένου συστήματος στον χώρο της συχνότητας φαίνεται στο σχήμα 4.25 και οι πόλοι – μηδενικά του στο σχήμα 4.26.



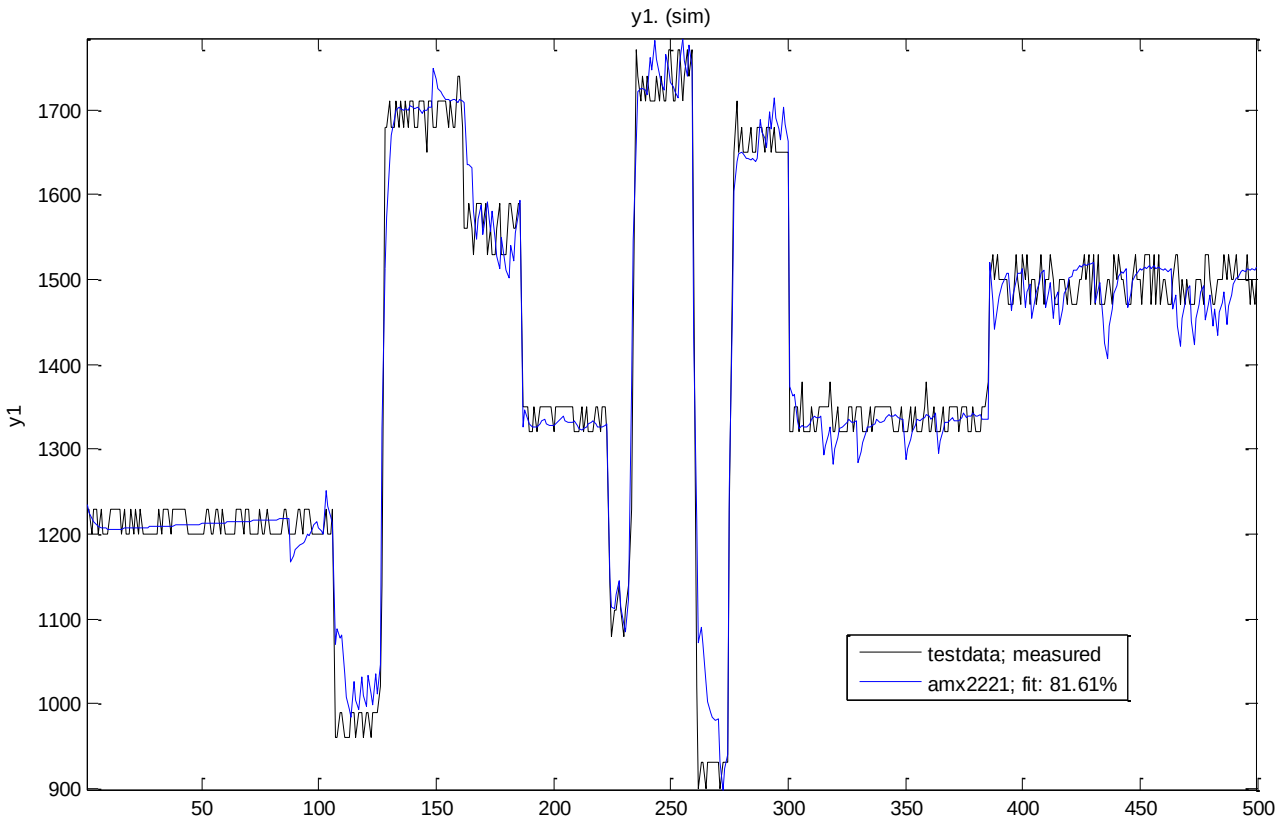
Σχήμα 4.25 – απόκριση συχνότητας μοντέλου ARX 10 10 1



Σχήμα 4.26 – Πόλοι και μηδενικά μοντέλου ARX 10 10 1 (z Transform)

4.6.2 Γραμμικό μοντέλο, ARMAX:

Για **na = 2** (αριθμός πόλων), **nb = 2** (αριθμός μηδενικών), **nc = 2** (αριθμός συντελεστών στοχαστικού θορύβου) και χωρίς καθυστέρηση[2], το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.27, δίνοντας του το testing set ως είσοδο.

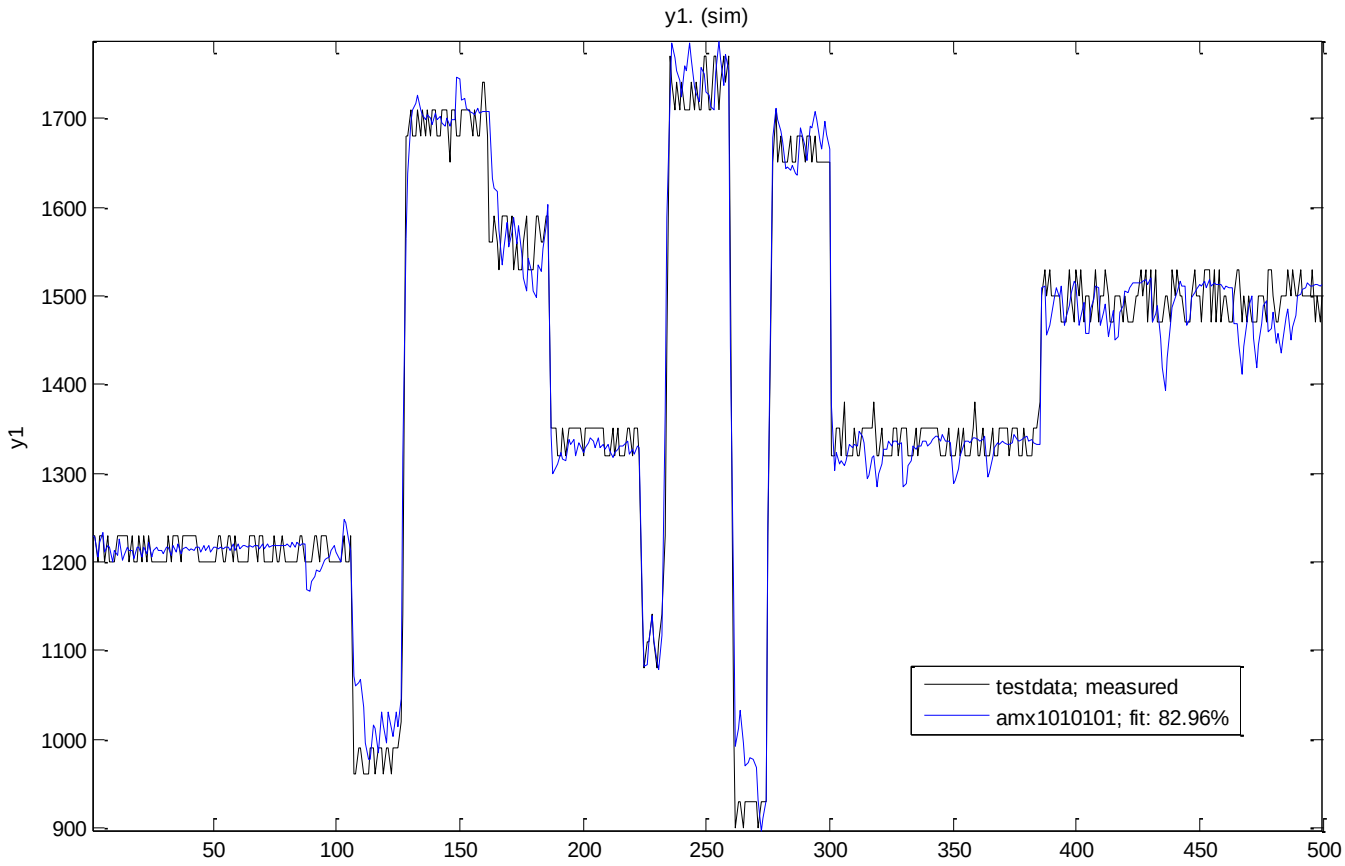


Σχήμα 4.27 – Αποτελέσματα μοντελοποίηση ARMAX na=2,nb=2, nc=2 no delay

Στο διάγραμμα του σχήματος 4.27 βλέπουμε πως έχουμε ένα μοντέλο του οποίου τα αποτελέσματα μοιάζουν σε μεγάλο βαθμό, κατά 81.61% με τα αποτελέσματα του πειράματος. Αυτό μπορεί να οφείλεται στο γεγονός ότι ο κινητήρας λόγο της αδράνειας του αποτελεί ένα χαμηλοπερατό φίλτρο μοιάζοντας περισσότερο με κάποια χαμηλοπερατό φίλτρο τύπου moving average (MA).

[2] Οι παράμετροι na, nb, nk, nc όπως και όλα τα πιθανά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

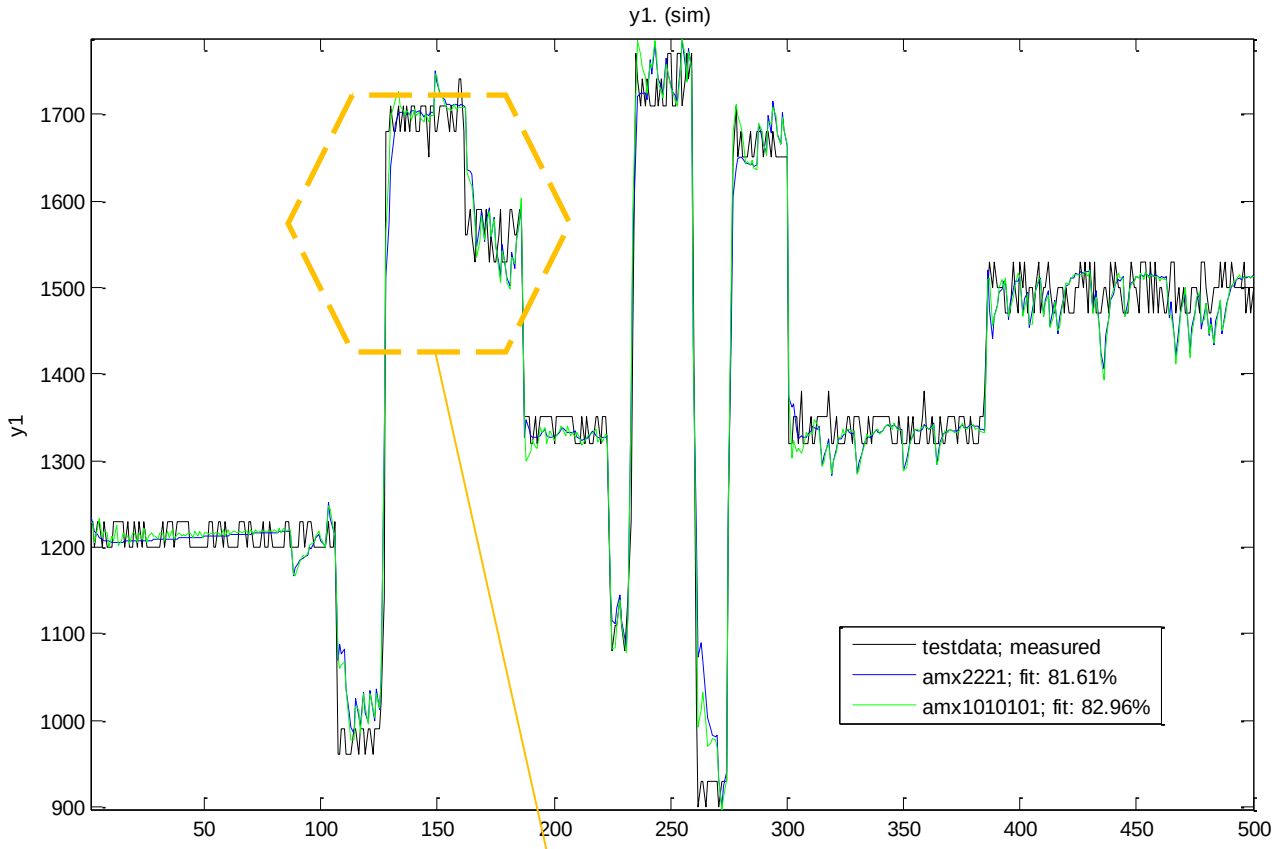
Για **na = 10** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών), **nc = 10** (αριθμός συντελεστών στοχαστικού θορύβου) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.28, δίνοντας του το testing set.



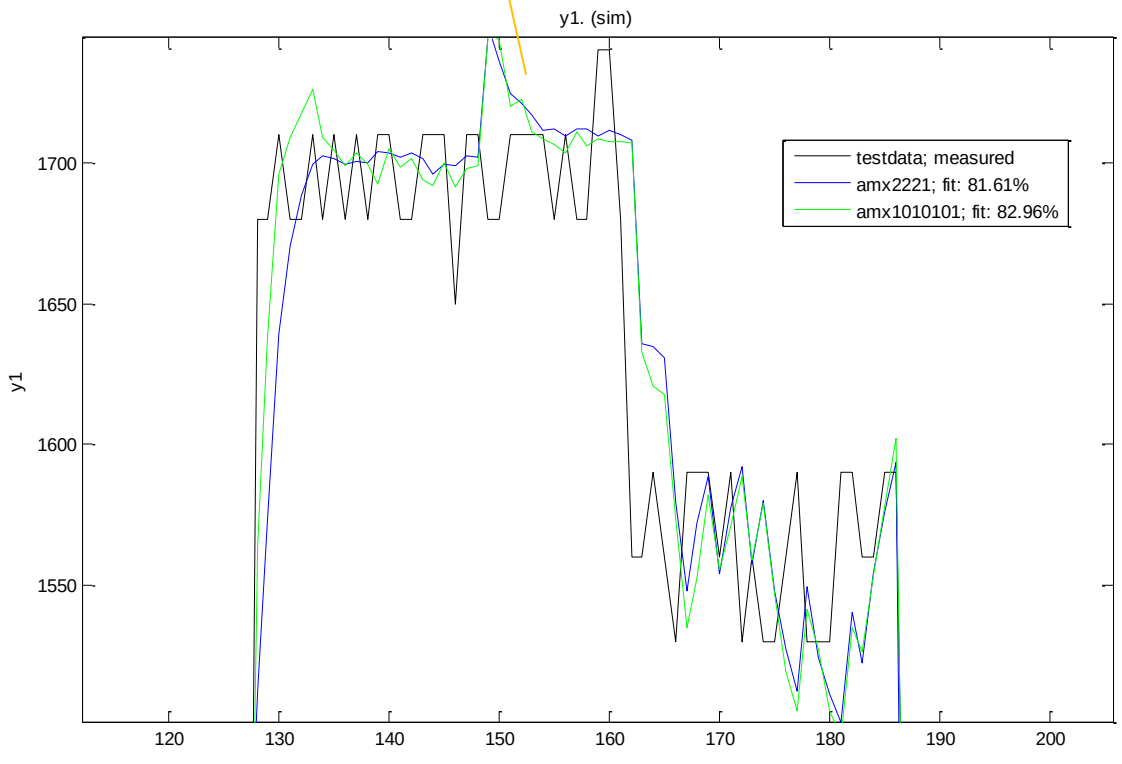
Σχήμα 4.28 – Αποτελέσματα μοντελοποίηση ARMAX na=10,nb=10, nc=10 no delay

Εδώ μπορούμε να πούμε πως το θεωρητικό μοντέλο προσεγγίζει ακόμα καλύτερα το πραγματικό μοντέλο, αφού τα αποτελέσματα ταιριάζουν κατά fit: 82.96%. Θα πρέπει να σημειωθεί πως αυξάνοντας τον αριθμό των πόλων και μηδενικών πάνω από 10 δεν υπήρχε ουσιαστική αλλαγή στα δεδομένα, όπως επίσης στην περίπτωση που έχουμε διαφορετικό αριθμό πόλων, μηδενικών και συντελεστών θορύβου έχουμε και χειρότερη προσέγγιση.

Έτσι μπορούμε σε αυτό το στάδιο να συγκρίνουμε τα μοντέλα που εξήχθησαν μέσω του σχήματος 4.29, όπου φαίνεται και οπτικά η διαφορά μεταξύ των μοντέλων.



α.

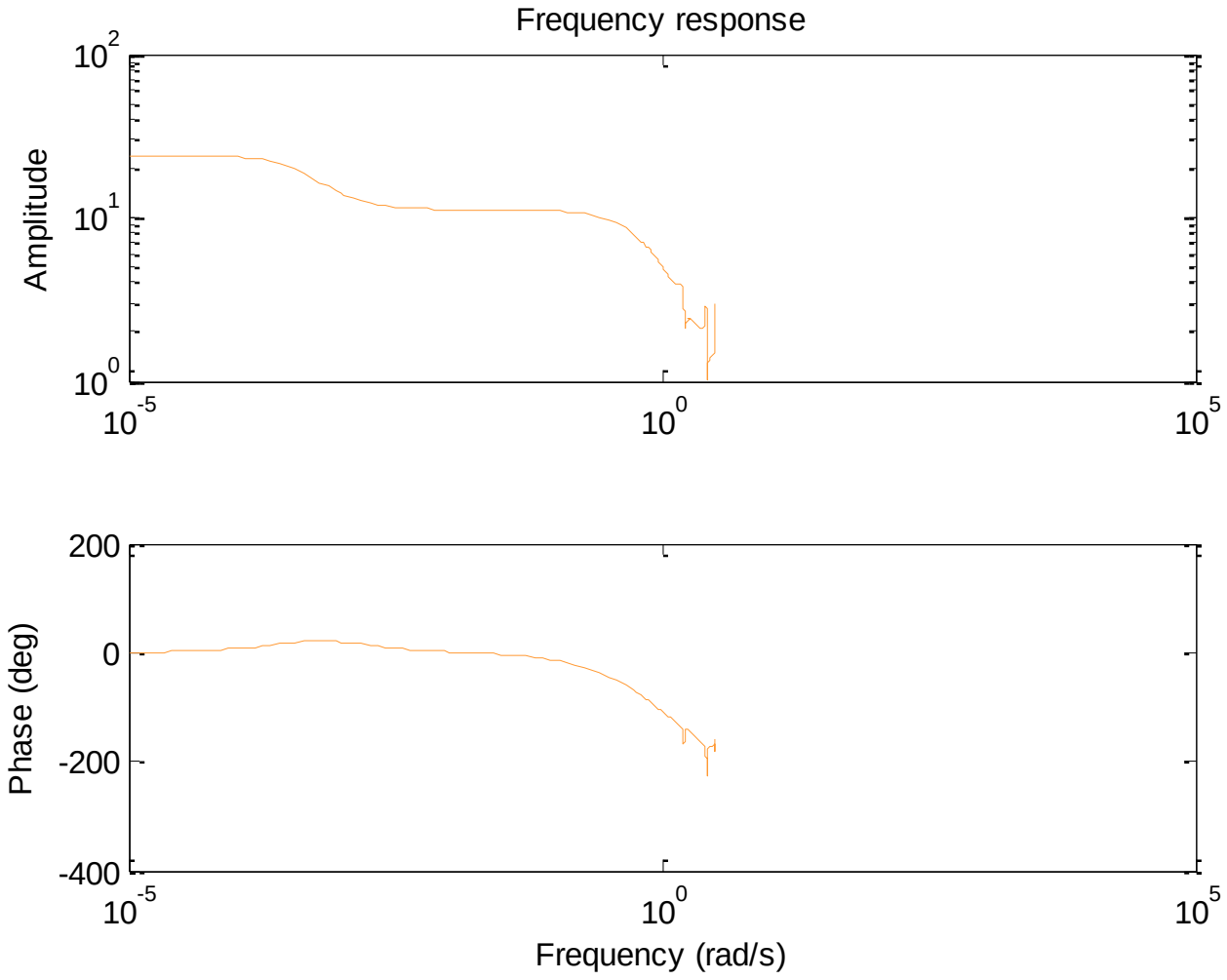


β.

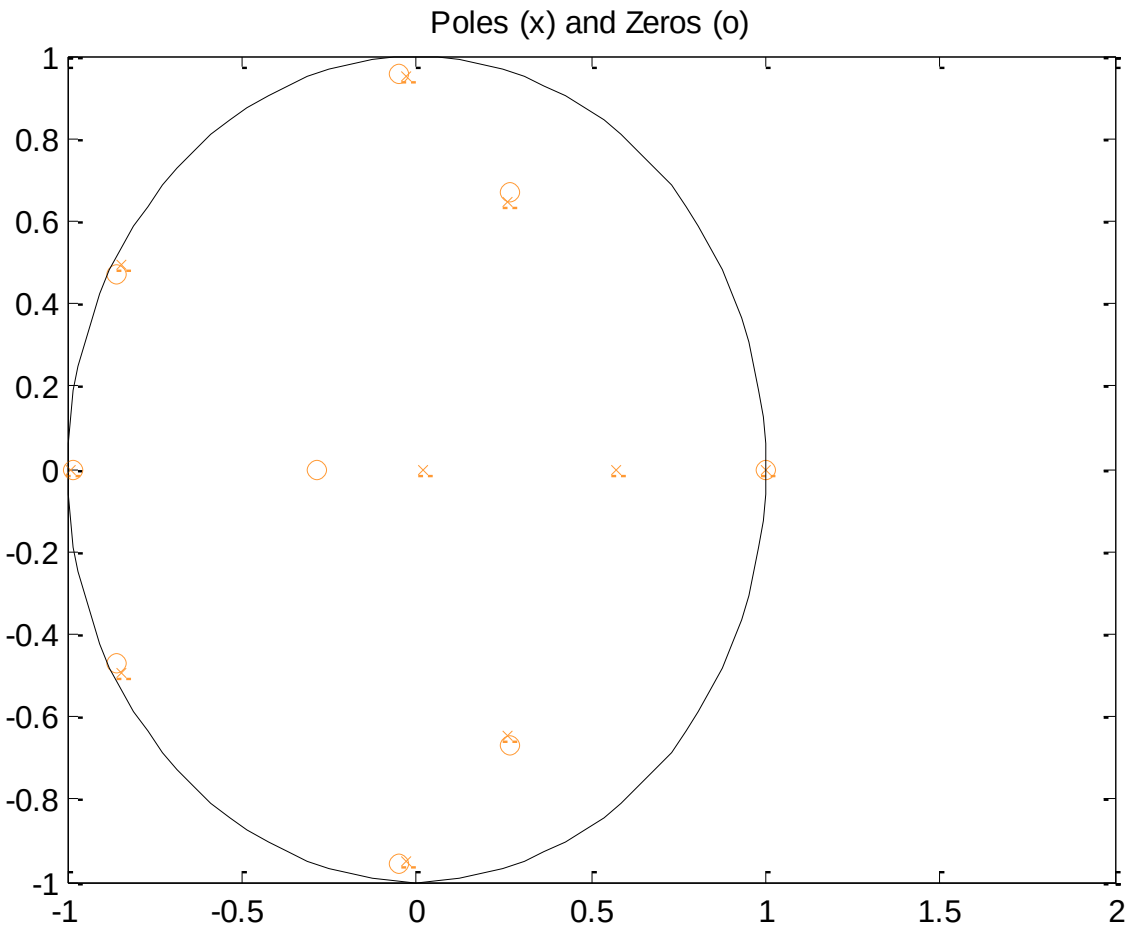
Σχήμα 4.29 – α .Σύγκριση μοντέλων ARMAX, β.Zoom in

Για τα μοντέλα τύπου ARMAX συμπεραίνουμε πως μπορούν και προσεγγίζουν και τα δύο πολύ καλά το πραγματικό σύστημα. ΤΟ σύστημα ARMAX 10 10 10 1 όμως φαίνεται να έχει κάνει ελαφρώς καλύτερη εκτίμηση η οποία φαίνεται στο σχήμα 4.29β και στον συντελεστή fit ο οποίος είναι κατά 1.3% μεγαλύτερος. Έτσι η επιλογή του καλύτερου ARMAX μοντέλου που κάνουμε είναι αυτή του ARMAX 10 10 10 1.

Στο σχήμα 4.30 βλέπουμε την απόκριση συχνότητας του ARMAX 10 10 10 1, ενώ στο σχήμα 4.31 βλέπουμε τους πόλους και τα μηδενικά του ιδίου συστήματος.



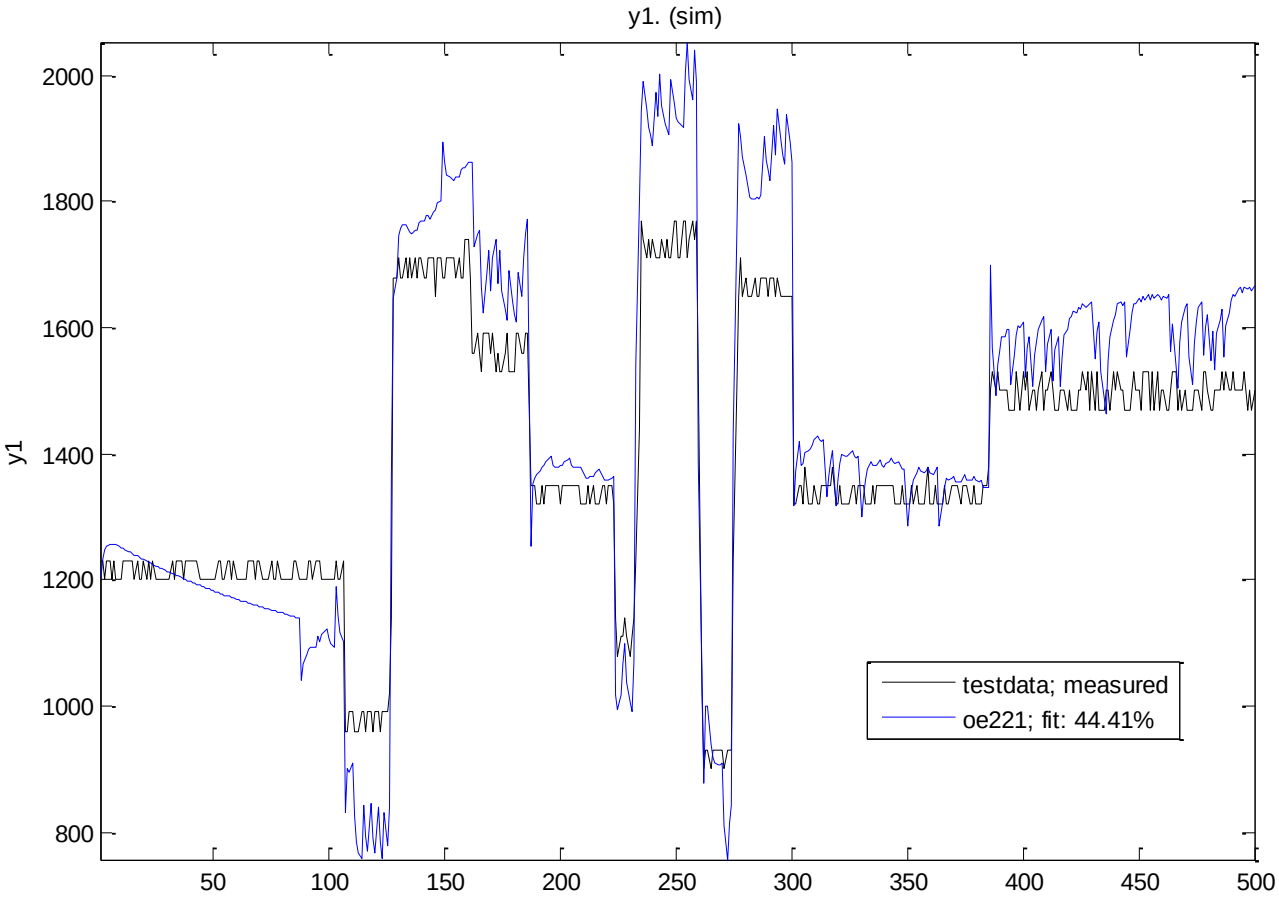
Σχήμα 4.30 – απόκριση συχνότητας μοντέλου ARMAX 10 10 10 1



Σχήμα 4.31 – Πόλοι και μηδενικά μοντέλου ARMAX 10 10 10 1 (z Transform)

4.6.3 Γραμμικό μοντέλο, OE:

Για **na = 2** (αριθμός πόλων), **nb = 2** (αριθμός μηδενικών)[3] και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.32, δίνοντας του το testing set ως είσοδο.



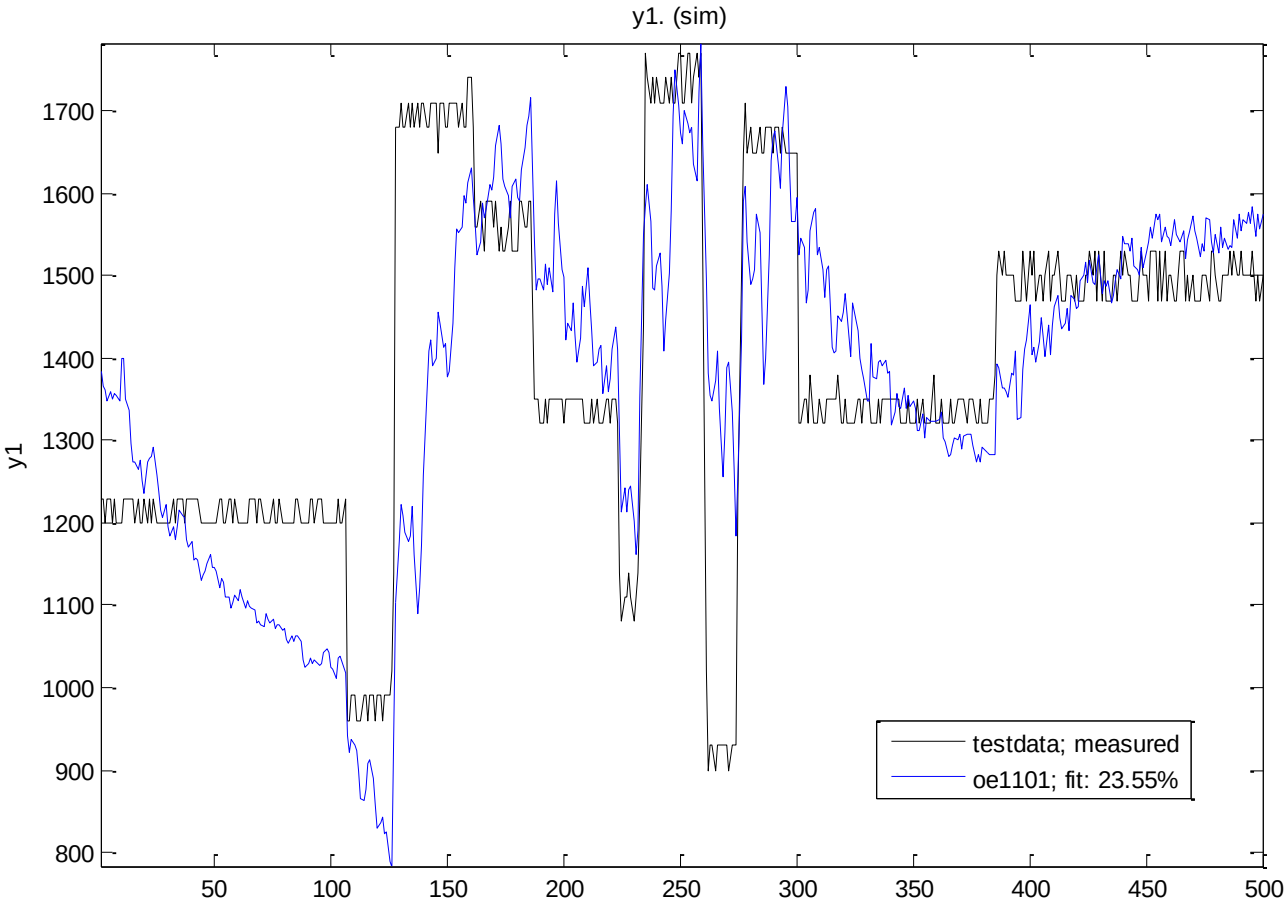
Σχήμα 4.32 – Αποτελέσματα μοντελοποίηση OE na=2,nb=2, no delay

Φαίνεται πως αποτελεί μια κακή σχετικά μοντελοποίηση, αφού έχουμε fit: 44.41% ομοιότητα με τα πειραματικά δεδομένα, αφού ταιριάζουν κατά λιγότερο από 50%.

[3] Οι παράμετροι na, nb, nk όπως και όλα τα πιθανά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

Για **na = 1** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.33, δίνοντας του το testing set ως είσοδο.

Στο γράφημα του σχήματος 5.16 βλέπουμε πως έχουμε μια ακόμα χειρότερη μοντελοποίηση, αφού η εκτίμηση είναι κατά 23.55% όμοια, δηλαδή απέχει ακόμα περισσότερο από το πραγματικό σύστημα.

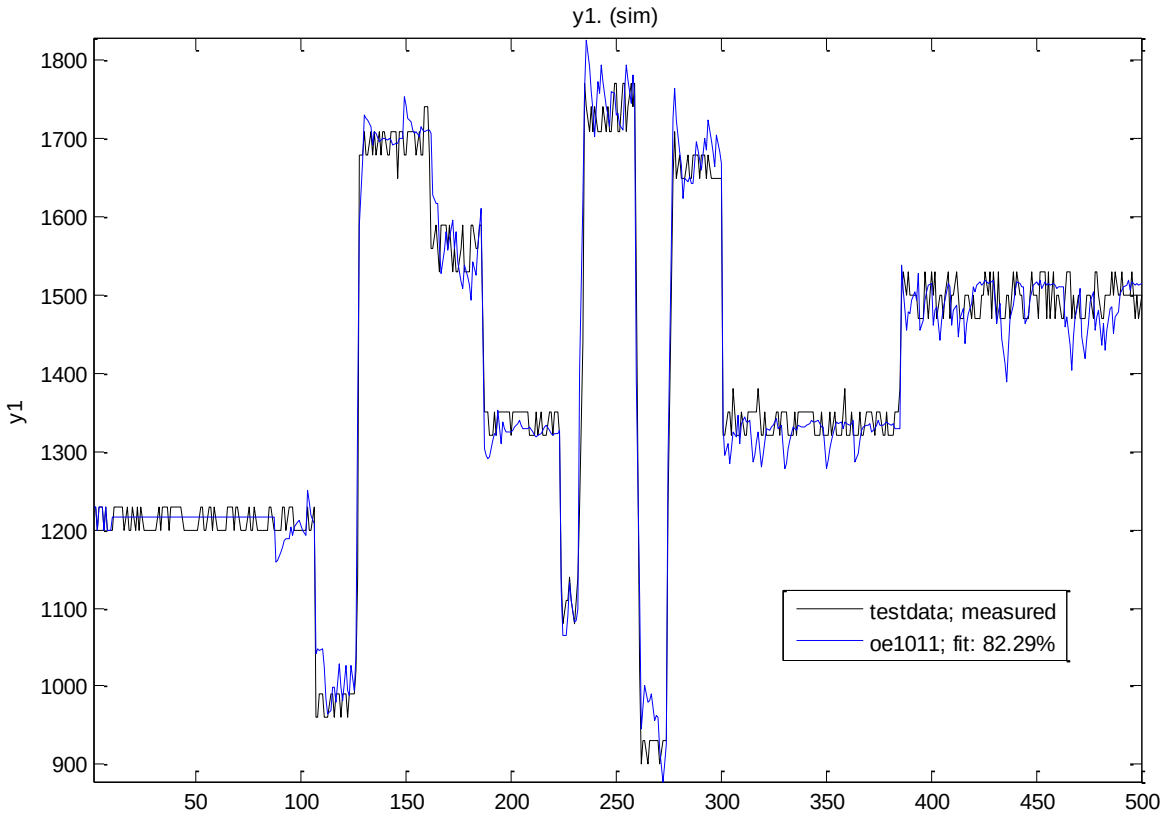


Σχήμα 4.33 – Αποτελέσματα μοντελοποίηση OE na=1,nb=10, no delay

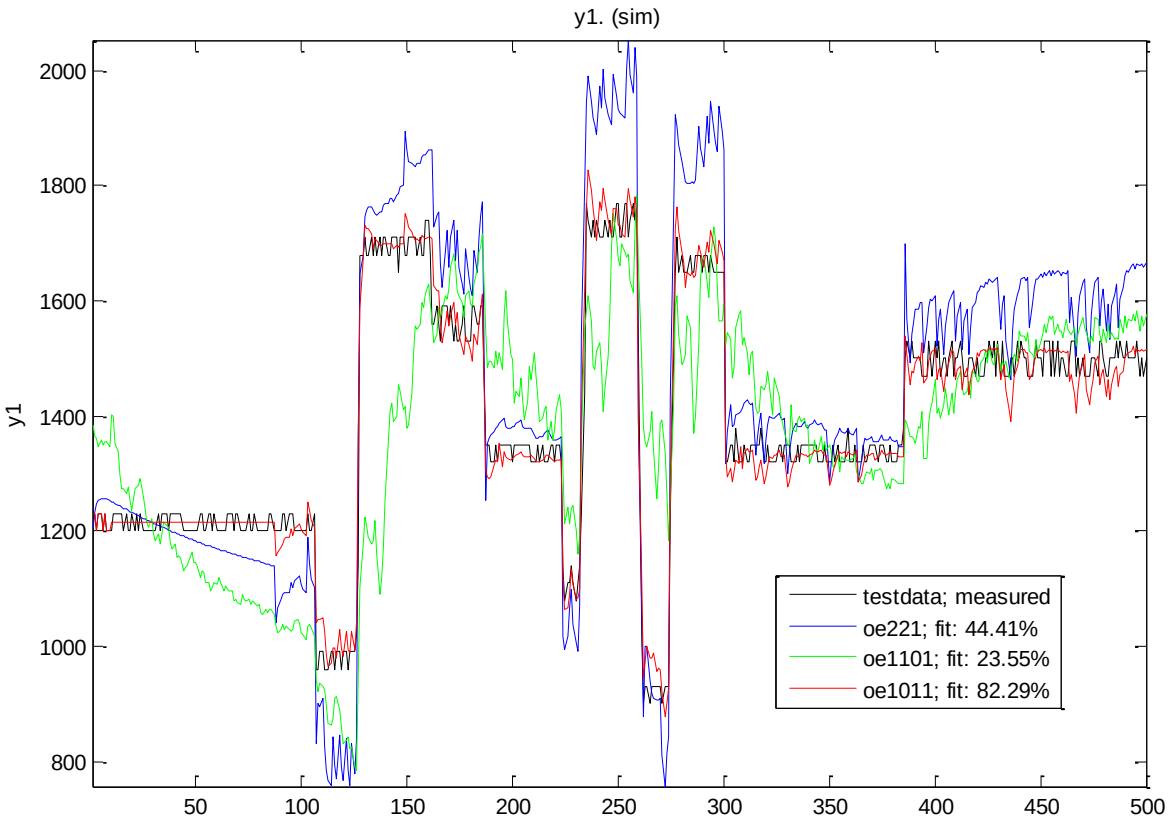
Για **na = 10** (αριθμός πόλων), **nb = 1** (αριθμός μηδενικών) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.34, δίνοντας του το testing set ως είσοδο.

Στο γράφημα του σχήματος 4.34 βλέπουμε πως έχουμε μια πολύ καλύτερη μοντελοποίηση, αφού η εκτίμηση είναι κατά 82.29% όμοια. Στο σχήμα 4.35 βλέπουμε την σύγκριση των μοντέλων OE, στο οποίο φαίνεται ξεκάθαρα πως το μοντέλο OE 10 1 1 αποτελεί το πιο κοντινό στο πραγματικό, σύμφωνα με το testing set.

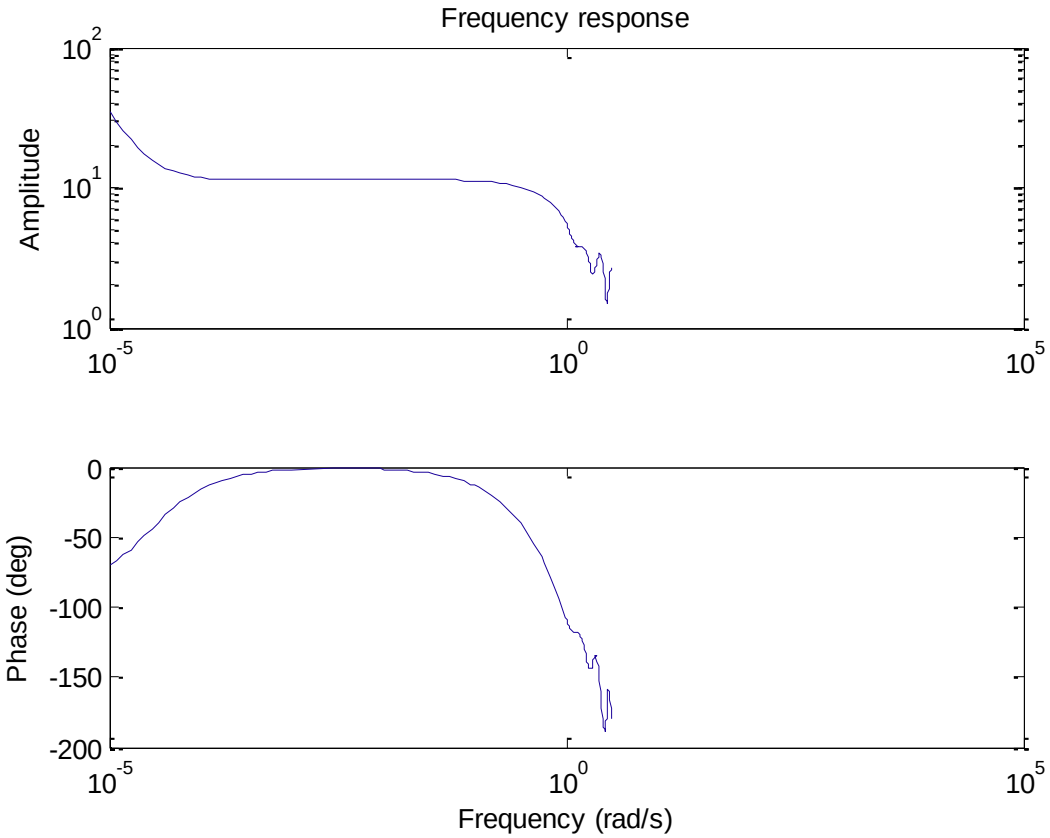
Έτσι θα μπορούσαμε να πούμε πως η καλύτερη επιλογή μοντέλου OE θα ήταν αυτή του 10-1-1, του οποίου η απόκριση συχνότητας φαίνεται στο σχήμα 4.36 και οι πόλοι – μηδενικά του στο σχήμα 4.37.



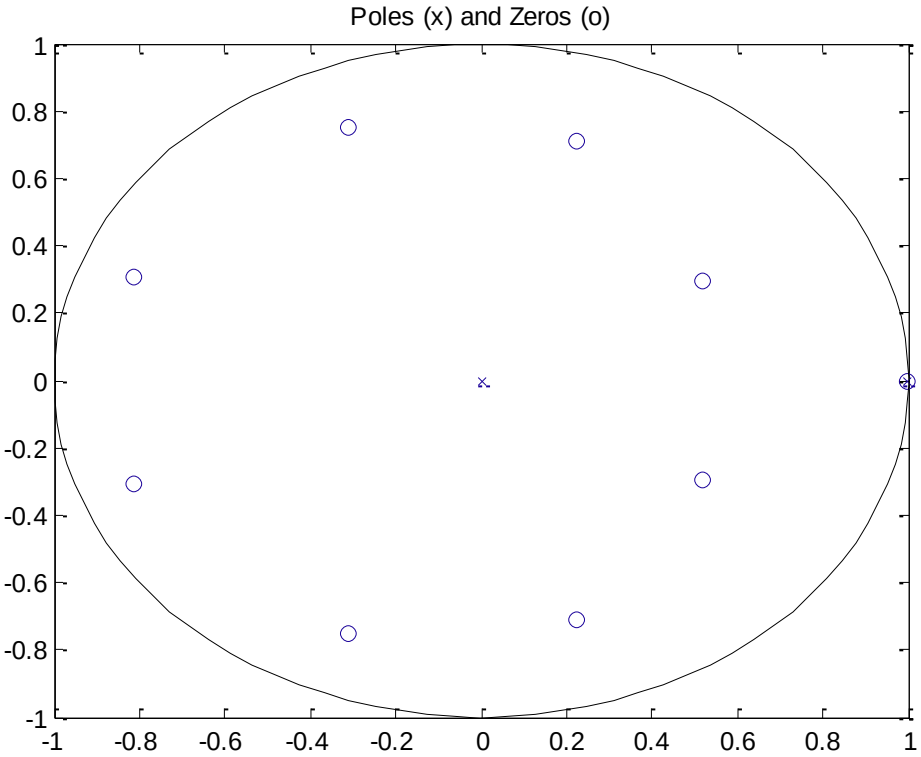
Σχήμα 4.34 – Αποτελέσματα μοντελοποίηση OE na=10,nb=1, no delay



Σχήμα 4.35 – Σύγκριση μοντέλων ΟΕ



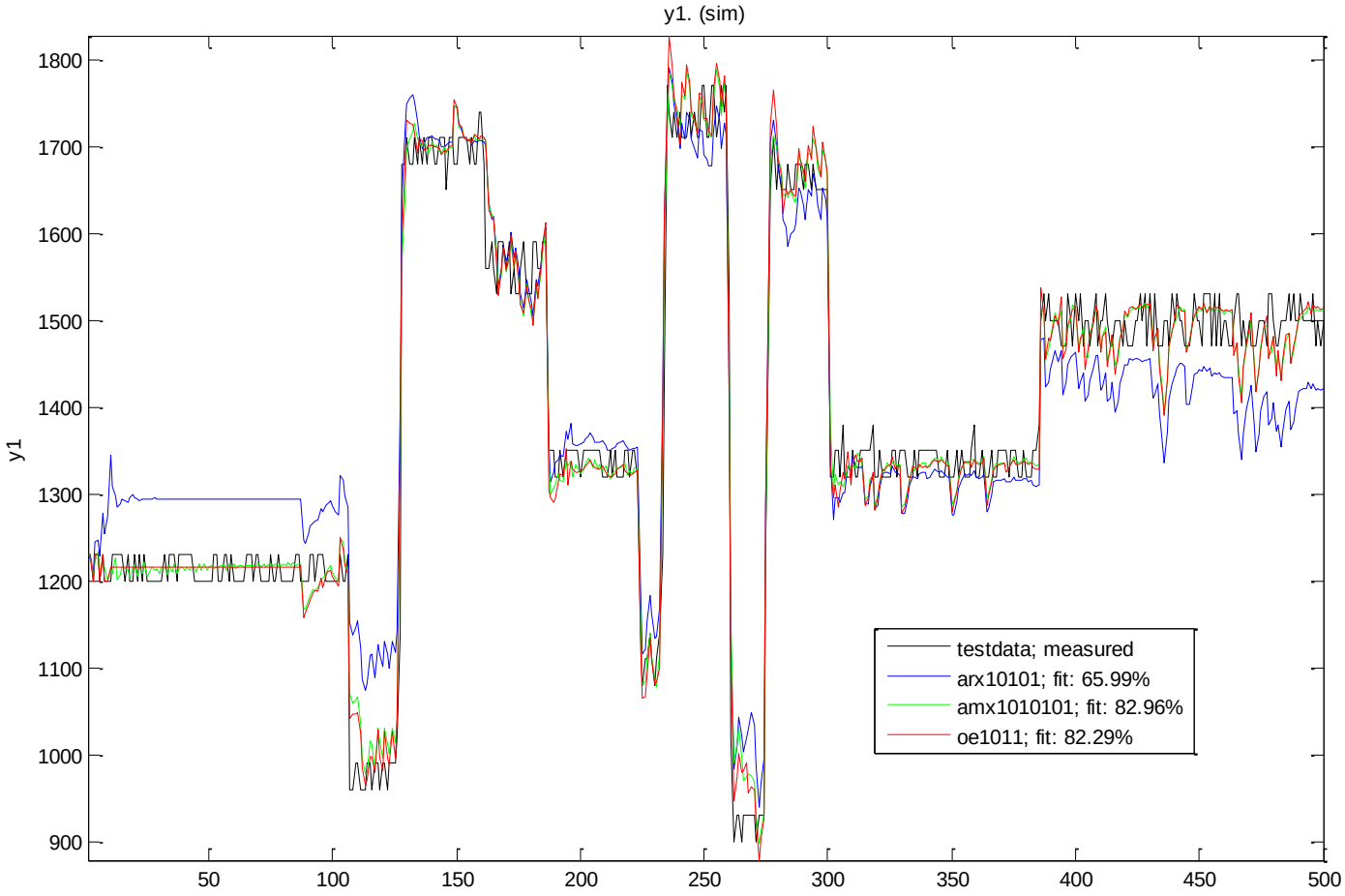
Σχήμα 4.36 – απόκριση συχνότητας μοντέλου OE 10 1 1



Σχήμα 4.37 – Πόλοι και μηδενικά μοντέλου OE 10 1 1 (z Transform)

4.6.4 *Επιλογή καλύτερου γραμμικού μοντέλου.*

Τα γραμμικά μοντέλα που μελετήθηκαν είναι τα μοντέλα ARX, ARMAX και OE. Από κάθε κατηγορία έγινε η επιλογή του καλύτερου μοντέλου σύμφωνα με το testing set. Έτσι σε αυτό το σημείο θα συγκρίνουμε τα τρία μοντέλα που επιλέγηκαν, δηλαδή το μοντέλο ARX 10 10 1, το μοντέλο ARMAX 10 10 10 1 και το μοντέλο OE 10 1 1. Στο σχήμα 4.38 μπορούμε να δούμε την σύγκριση των μοντέλων αυτών μέσω των αποτελεσμάτων τους στο testing set δεδομένων.

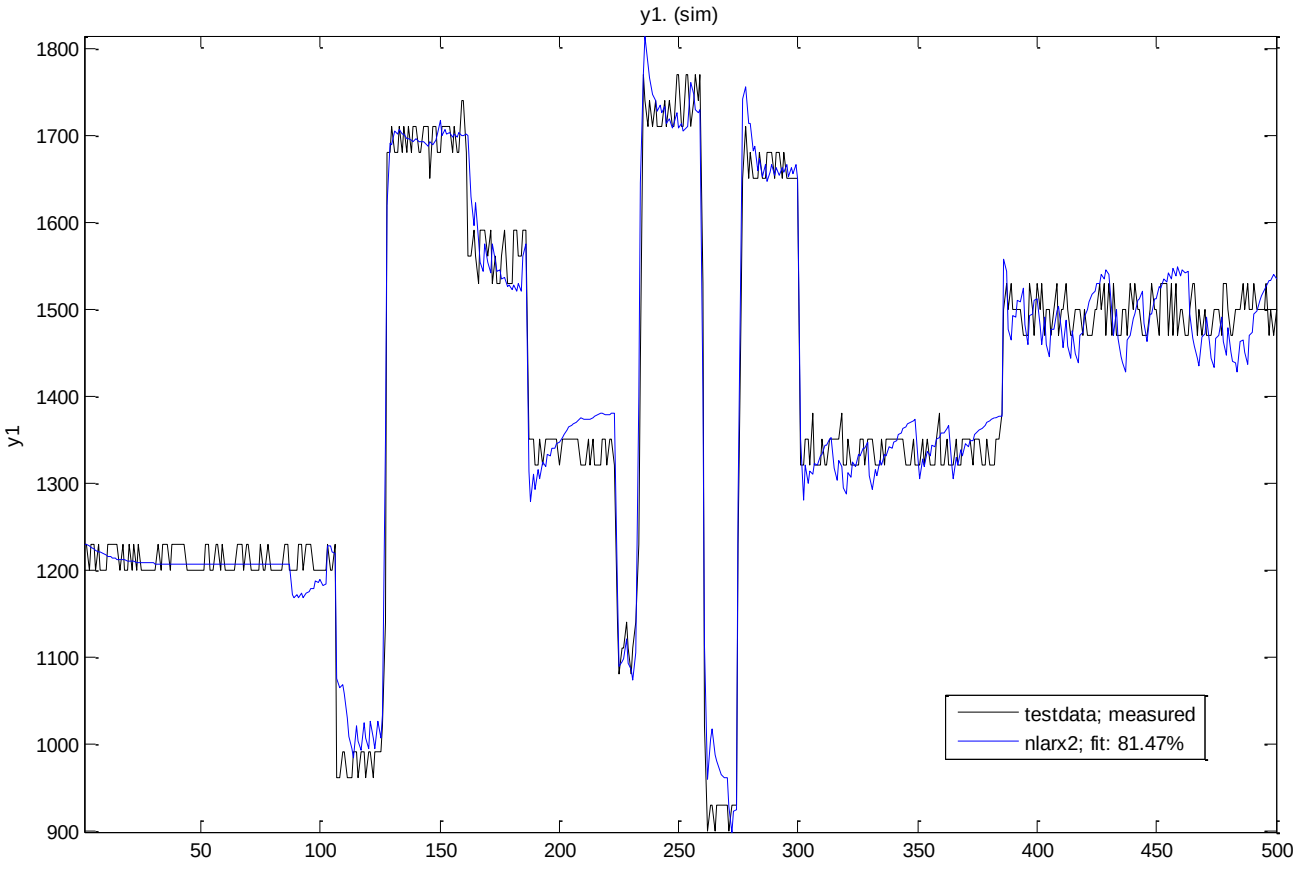


Σχήμα 4.38 – Σύγκριση γραμμικών μοντέλων

Έτσι θα μπορούσαμε να πούμε πως η καλύτερη επιλογή **γραμμικού** μοντέλου θα ήταν αυτή του ARMAX 10-10-10-1, το οποίο βρίσκεται πιο κοντά στην απόκριση του πραγματικού συστήματος, fit 82.96.

4.6.5 *Μη γραμμικό μοντέλο, non-linear ARX:*

Για **na = 2** (αριθμός προηγούμενων καταστάσεων εξόδου), **nb = 2** (αριθμός προηγούμενων καταστάσεων εισόδου)[4] το μη γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.32, δίνοντας του το testing set ως είσοδο.



Σχήμα 4.39 – Αποτελέσματα μη γραμμικής μοντελοποίησης ARX na=2,nb=2, no delay

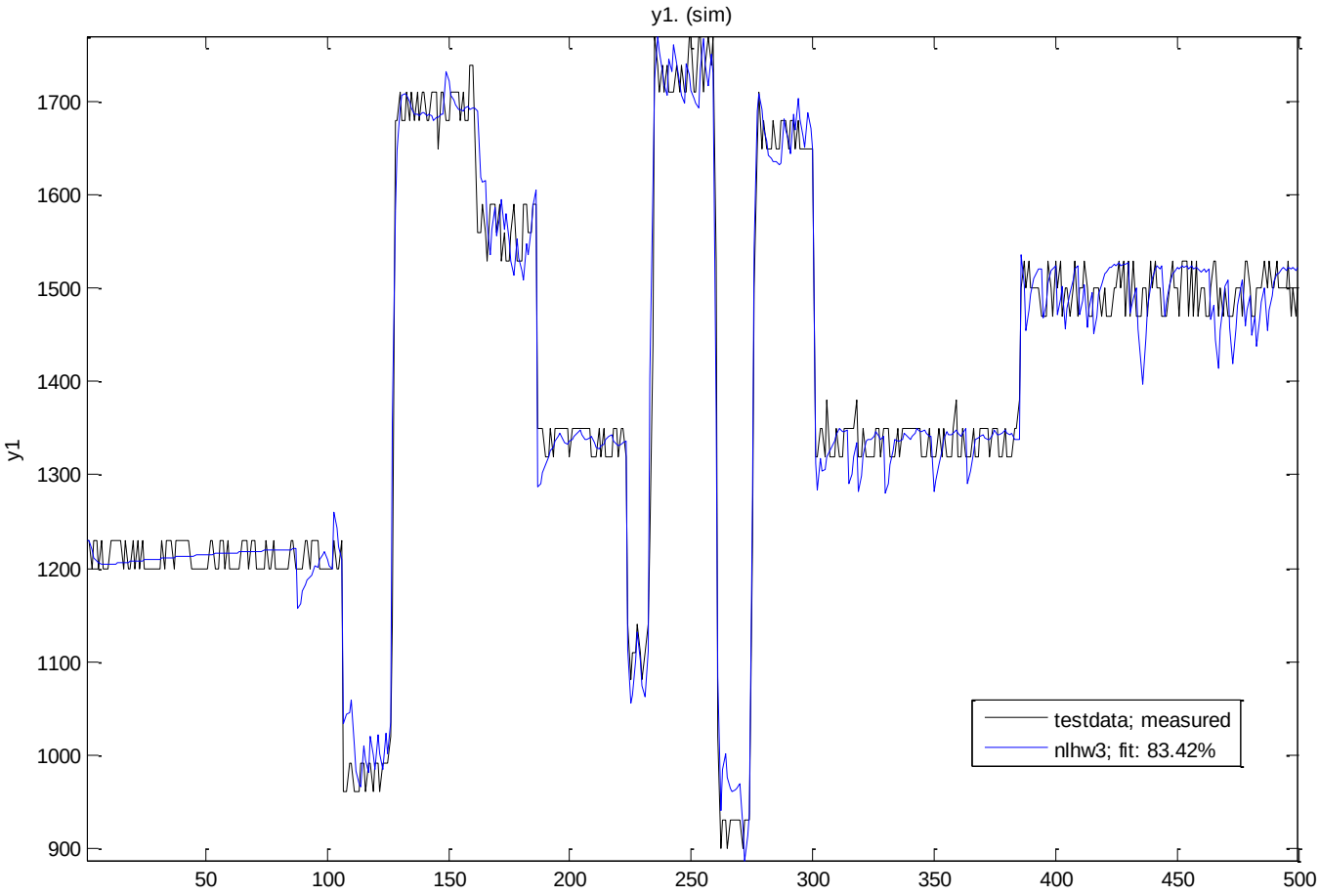
Στο σχήμα 4.39 φαίνεται πως το μη γραμμικό μοντέλο προσεγγίζει πολύ καλύτερά (81.49% ομοιότητα) το πραγματικό σύστημα από το αντίστοιχο γραμμικό ARX (γράφημα σχήματος 4.23) το οποίο είχε ομοιότητα μόλις 65.99%.

Θα πρέπει να σημειωθεί πως αλλάζοντας τις παραμέτρους na, nb και nk η εκτίμηση του μοντέλου γινόταν χειρότερη και γι' αυτό έγινε η επιλογή των 2 προηγούμενων καταστάσεων για την είσοδο και για την έξοδο.

[4] Οι παράμετροι na, nb όπως και όλα τα πιθανά μη γραμμικά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

4.6.6 Μη γραμμικό μοντέλο, Hammerstein -Wiener:

Για **na = 2** (αριθμός προηγούμενων καταστάσεων εξόδου), **nb = 2** (αριθμός προηγούμενων καταστάσεων εισόδου) το μη γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 4.40, δίνοντας του το testing set ως είσοδο.



Σχήμα 4.40 – Αποτελέσματα μη γραμμικής μοντελοποίησης Hammerstein – Wiener na=2,nb=2

Στο σχήμα 4.40 φαίνεται πως το μη γραμμικό μοντέλο Hammerstein - Wiener προσεγγίζει ακόμα καλύτερά (83.42% ομοιότητα) το πραγματικό σύστημα από το μη γραμμικό ARX (γράφημα σχήματος 4.39).

4.6.7 Τελική επιλογή θεωρητικού μοντέλου συστήματος ανεμιστήρα

Στον πίνακα 4.1 βλέπουμε τα αποτελέσματα της ομοιότητας του κάθε θεωρητικού μοντέλου με το πραγματικό για το testing set δεδομένων.

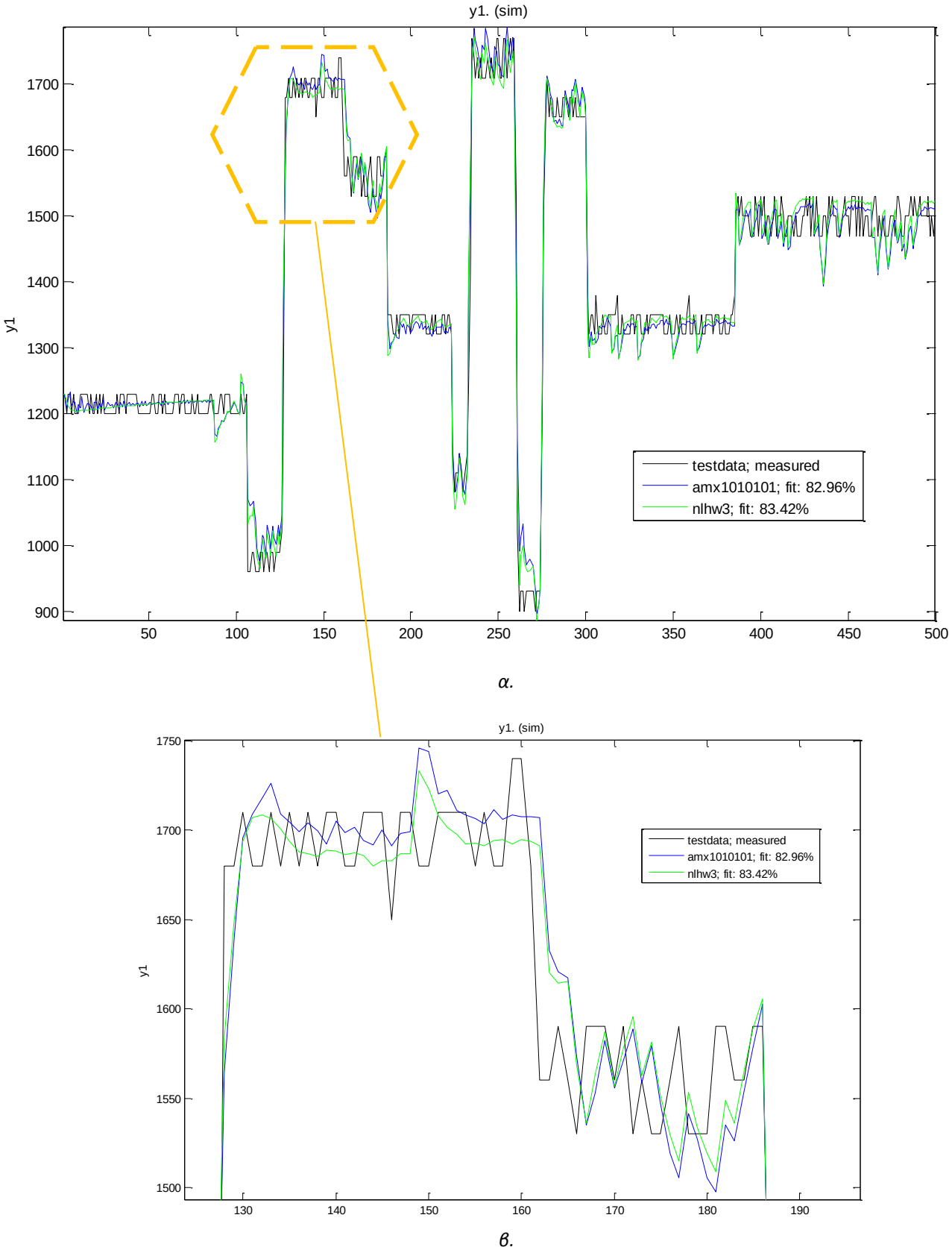
Πίνακας 4.1 – Πίνακες σύγκρισης μοντέλων συστήματος ανεμιστήρα.

Είδος	Μοντέλο	Παράμετροι	Ομοιότητα με το πραγματικό [%]
Γραμμικό	ARX	n_a=1, n_b=1, n_k=1	5.463
		n_a=4, n_b=4, n_k=1	36.23
		n_a=10, n_b=10, n_k=1	65.99 (καλύτερο ARX)
	ARMAX	n_a=2, n_b=2, nc=2, n_k=1	81.61
		n_a=10, n_b=10, nc=10, n_k=1	82.96 (καλύτερο γραμμικό)
	OE	n_a=2, n_b=2, n_k=1	44.41
		n_a=1, n_b=10, n_k=1	23.55
		n_a=10, n_b=1, n_k=1	82.29 (καλύτερο OE)
Μη γραμμικό	ARX	n_a=2, n_b=2, n_k=1	81.47
	Ham-Wien	n_a=2, n_b=2, n_k=1	83.42 (καλύτερο)

Έτσι για λόγους προσομοίωσης θα επιλέγαμε το μη γραμμικό μοντέλο που εξήχθη από την μοντελοποίηση τύπου Hammerstein – Wiener, το οποίο έχει τα εξής χαρακτηριστικά όπως φαίνονται εντός του Matlab:

```
  IDNLHW model with 1 output and 1 input
  Input name: u1
  Output name: y1
  Linear transfer function corresponding to the orders nb = 2, nf = 3,
nk = 1
  Input nonlinearity estimator: pwlinear with 2 units
  Output nonlinearity estimator: pwlinear with 2 units
  Loss function: 1927.5037
  Sampling interval:  1
  Estimated by PEM
```

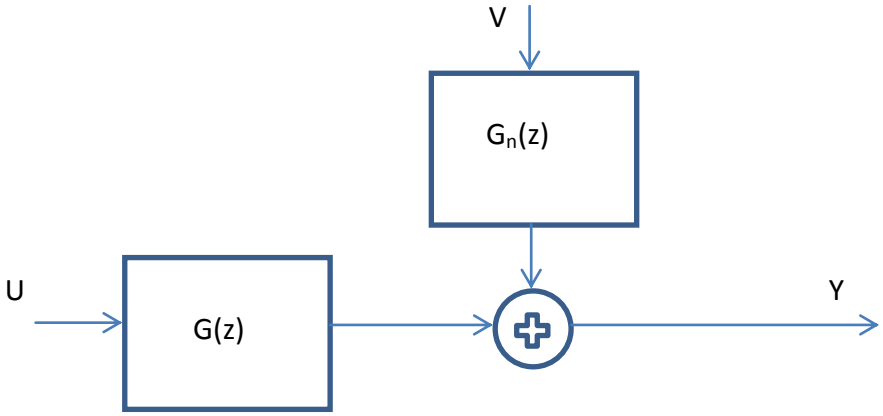
Στο σχήμα 4.41 φαίνεται η σύγκριση του καλύτερου γραμμικού μοντέλου (ARMAX 10 10 10 1) σε σχέση με το καλύτερο μη-γραμμικό (Ham- Wiener 2 2 1).



Σχήμα 4.41 – Σύγκριση καλύτερου γραμμικού και μη γραμμικού μοντέλου.

Επειδή στην παρούσα εργασία σκοπός μας είναι να εξαχθεί η συνάρτηση μεταφοράς του μοντέλου, επιλέξαμε ένα γραμμικό μοντέλο για την παρακάτω έρευνα. Αρχικά έγινε η επιλογή του ARMAX 10 10 10 1, αφού είχε τις καλύτερες αποδόσεις. Αναγκαζόμαστε να συνεχίσουμε με γραμμικό μοντέλο επειδή τα παρακάτω εργαλεία που θα χρησιμοποιηθούν όπως το SISOtool του Matlab, μπορούν να χειριστούν μόνο γραμμικά μοντέλα.

Το σύστημα ARMAX εκπροσωπείται από δύο γραμμικά υποσυστήματα τα οποία είναι συνδεδεμένα κατά τον τρόπο που φαίνεται στο σχήμα 4.42.



Σχήμα 4.42 – Το μοντέλο ARMAX.

Έτσι η συνάρτηση μεταφοράς για το μοντέλο ARMAX 10 10 10 1 που επιλέγεται είναι αυτή με 10 πόλους, 10 μηδενικά και δέκα συντελεστές θορύβου, και είναι η εξής:

$$Y = G(z)U + Gn(z)V \tag{4.32}$$

Όπου:

$$G(z) = \frac{N(z)}{D(z)} \tag{4.33}$$

Όπου:

$$N(z) = 3.52z^9 + 5.411z^8 + 3.304z^7 + 1.381z^6 - 1.707z^5 - 4.453z^4 - 3.126z^3 - 1.973z^2 - 1.973z - 0.4443 \tag{4.34}$$

$$D(z) = z^{10} + 0.6231z^9 - 0.1733z^8 - 0.1081z^7 - 0.6434z^6 - 0.7739z^5 + 0.07234z^4 - 0.002869z^3 - 0.2357z^2 + 0.2446z - 0.005297 \quad (4.35)$$

και

$$Gn(z) = \frac{Nn(z)}{Dn(z)} \quad (4.36)$$

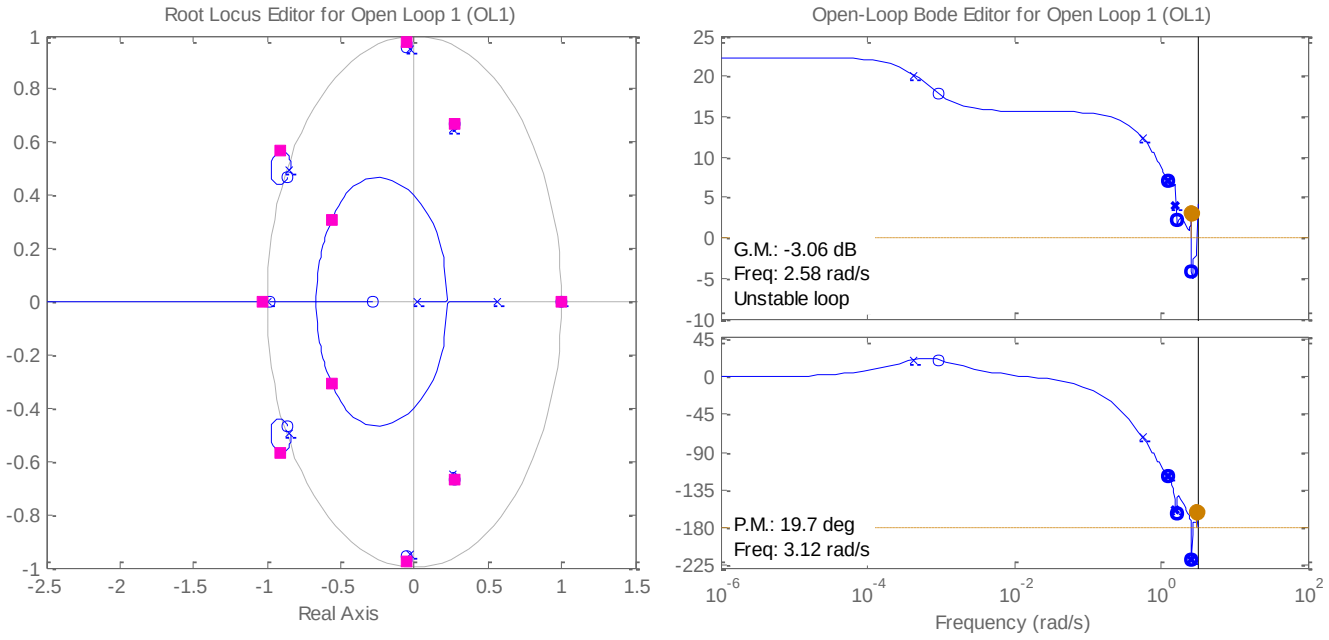
Όπου:

$$Nn(z) = 39.33z^{10} + 44.92z^9 + 19.38z^8 + 5.245z^7 - 22.6z^6 - 37.98z^5 - 16.93z^4 - 11.83z^3 - 15.9z^2 + 0.0244z - 2.901 \quad (4.37)$$

$$Dn(z) = z^{10} + 0.6231z^9 - 0.1733z^8 - 0.1081z^7 - 0.6434z^6 - 0.7739z^5 + 0.07234z^4 - 0.002869z^3 - 0.2357z^2 + 0.2446z - 0.005297 \quad (4.38)$$

(Σχέσεις 4.32 –4.38: Συνάρτηση μεταφοράς μοντέλου ARMAX 10 10 10 1)

Στο σχήμα 4.43 φαίνεται ο γεωμετρικός τόπος ριζών του ανοιχτού βρόχου του συστήματος του μοντέλου, όπως και το διάγραμμα Bode.



α. β.

Σχήμα 4.43 – α. Γεωμετρικός τόπος ριζών μοντέλου, β. Διάγραμμα Bode (ARMAX 10 10 10 1)

Παρόλα αυτά λόγο της φύσης και του μεγέθους του μοντέλου ARMAX 10 10 10 1 ήταν αδύνατο να γίνει το autotuning στο sisotool (που θα δούμε παρακάτω), όπως και για τα μοντέλα OE. Έτσι τελικά επιλέχθηκε στην παρακάτω ενότητα το σύστημα να προσομοιώνεται μέσω του μοντέλου ARX 10 10 1 του οποίου η συνάρτηση μεταφορά είναι η παρακάτω:

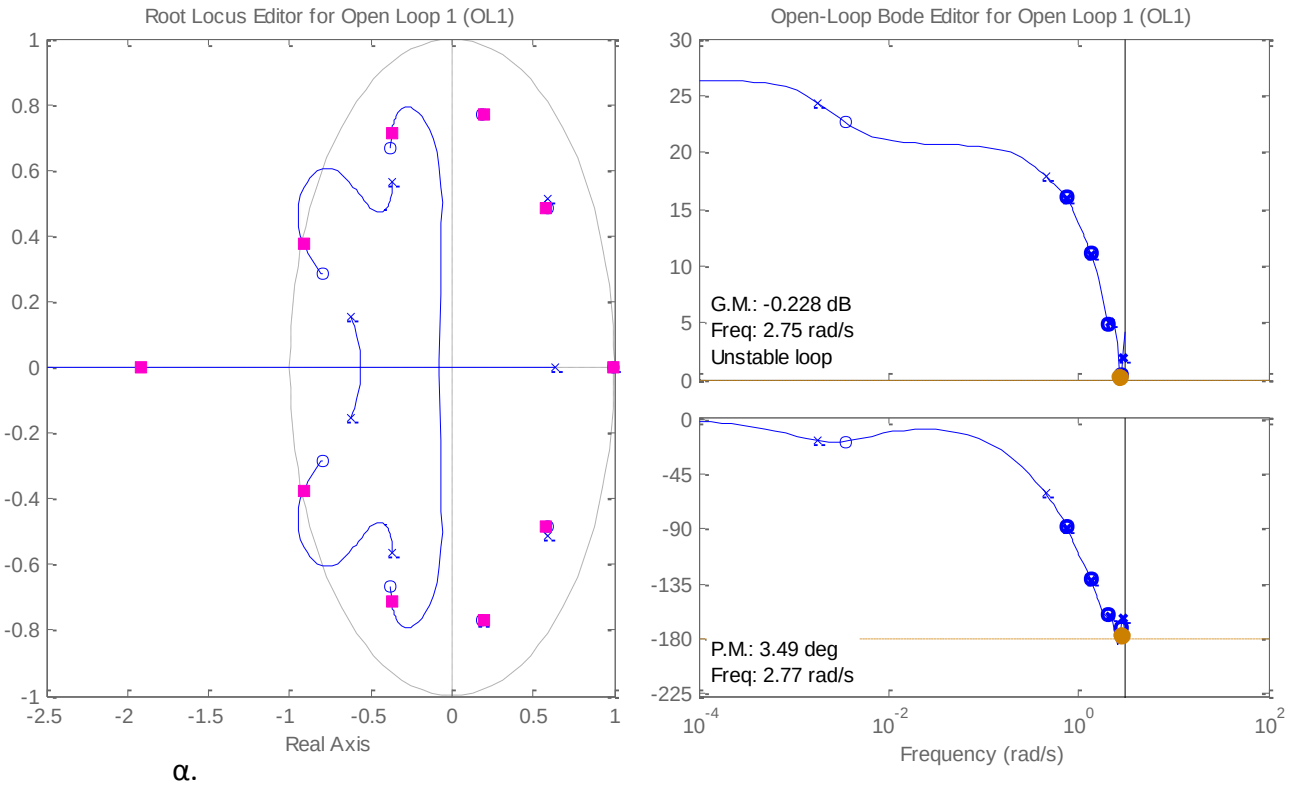
$$G(z)=\frac{N(z)}{D(z)} \tag{4.39}$$

Όπου:

$$N(z)=3.142z^{9}-0.6156z^{8}-0.8859z^{7}+0.01404z^{6}-0.3957z^{5} \\ -0.4478z^{4}-0.1552z^{3}-0.1941z^{2}+0.06822z-0.4863 \tag{4.40}$$

$$D(z)=z^{10}-1.195z^{9}+0.3043z^{8}+0.1058z^{7}-0.214z^{6}+0.0149z^{5}+ \\ 0.01502z^{4}+0.02517z^{3}-0.07907z^{2}-0.02145z+0.04643 \tag{4.41}$$

(Σχέσεις 4.39 –4.41: συνάρτηση μεταφοράς μοντέλου ARX 10 10 1)



α.

β.

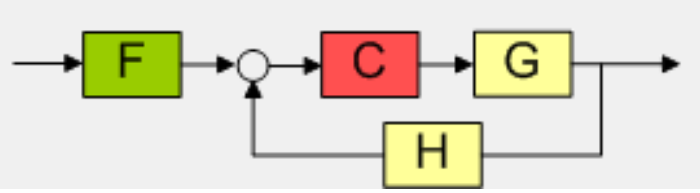
Σχήμα 4.44 – α. Γεωμετρικός τόπος ριζών μοντέλου, β. Διάγραμμα Bode (ARX 10 10 1)

4.7 Υπολογισμός PID μέσω του SISOtool

Στην συνέχεια γίνεται ο αυτόματος εντοπισμός κατάλληλου ελεγκτή PID, σύμφωνα με το γραμμικό μοντέλο που εξήχθη από την προηγούμενη ενότητα, που να ελέγχει το σύστημα όσο το πιθανόν βέλτιστα έτσι ώστε να υλοποιείται το διάγραμμα βαθμίδων που φαίνεται στο σχήμα 3.3.

Στην πραγματικότητα προσομοιώνουμε το άγνωστο σύστημα με το μοντέλο από την προηγούμενη ενότητα για να υλοποιήσουμε auto tuning του ελεγκτή PID που θα προηγείται από αυτό. Το auto tuning υλοποιήθηκε μέσω του εργαλείου Sisotool του Matlab.

Αρχικά εισήχθη στο sisotool η συνάρτηση μεταφοράς του συστήματος στην θέση του συστήματος G, στο σχήμα 4.45.



Σχήμα 4.45 – Εισαγωγή του μοντέλου στο sisotool

Πριν από το ξεκίνημα του auto tuning τα συστήματα ορίζονται ως εξής:

Πίνακας 4.2 – Αρχικοποίηση συστημάτων

System	Data
G	< arx10101 >
H	1
C	1
F	1

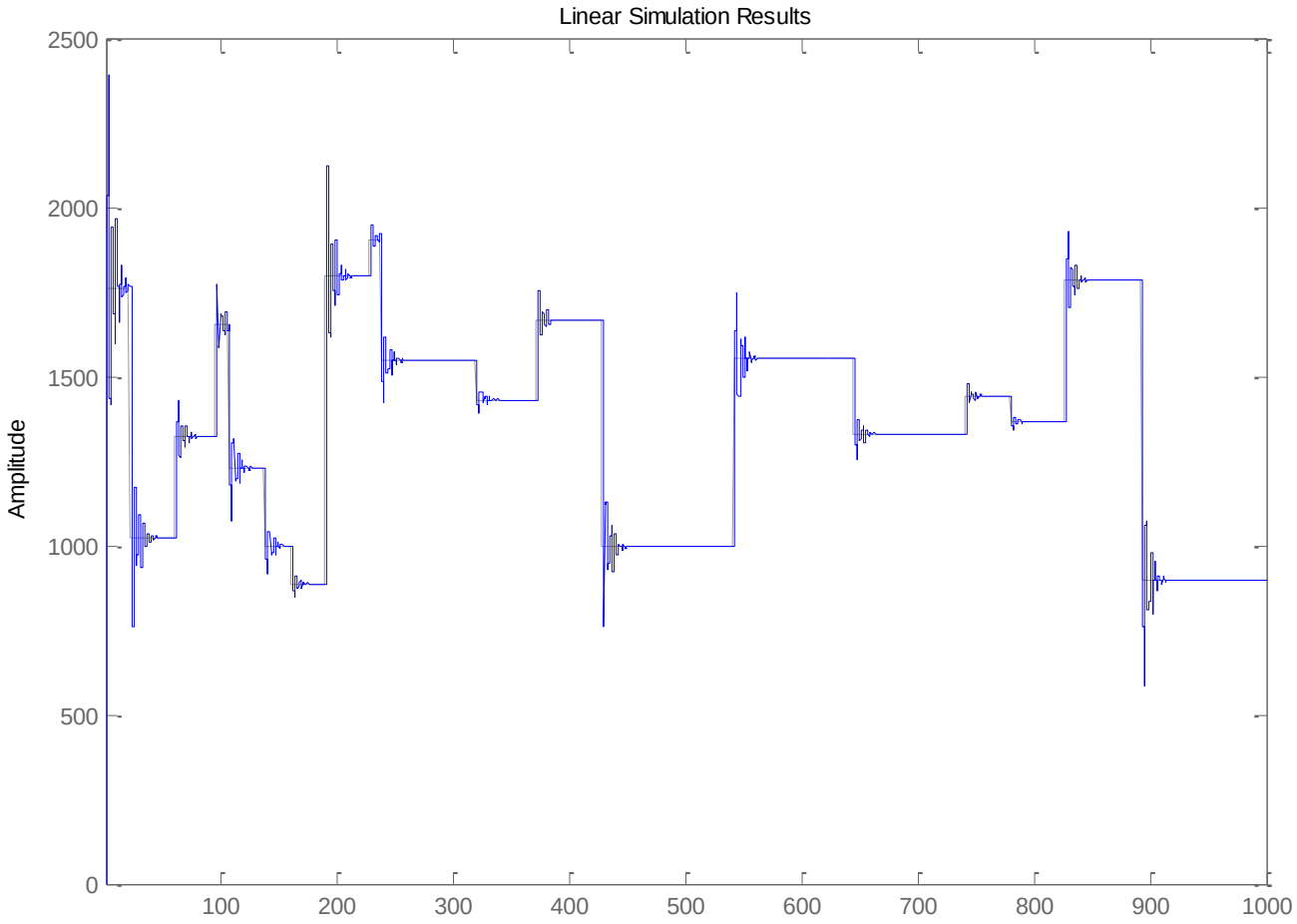
Ο ελεγκτής PID είναι στην πράξη το υποσύστημα C για το οποίο θα γίνει η αναζήτηση των καλύτερων συντελεστών Kp, Ki και Kd. Οι μέθοδοι του αυτόματου tuning που επιλέγονται είναι οι εξής:

- Κλασσικές μέθοδοι:
  - Μέσω απόκρισης συχνότητας – Ziegler - Nichols
  - Approximate MIGO – απόκριση συχνότητας
- Robust response time:
  - Αυτόματη (balanced performance and robustness)

Έτσι προκύπτουν τα εξής αποτελέσματα:

4.7.1 Μέσω απόκρισης συχνότητας – Ziegler – Nichols

Kp = 0.368, Ki = 0.122 Kd = 0

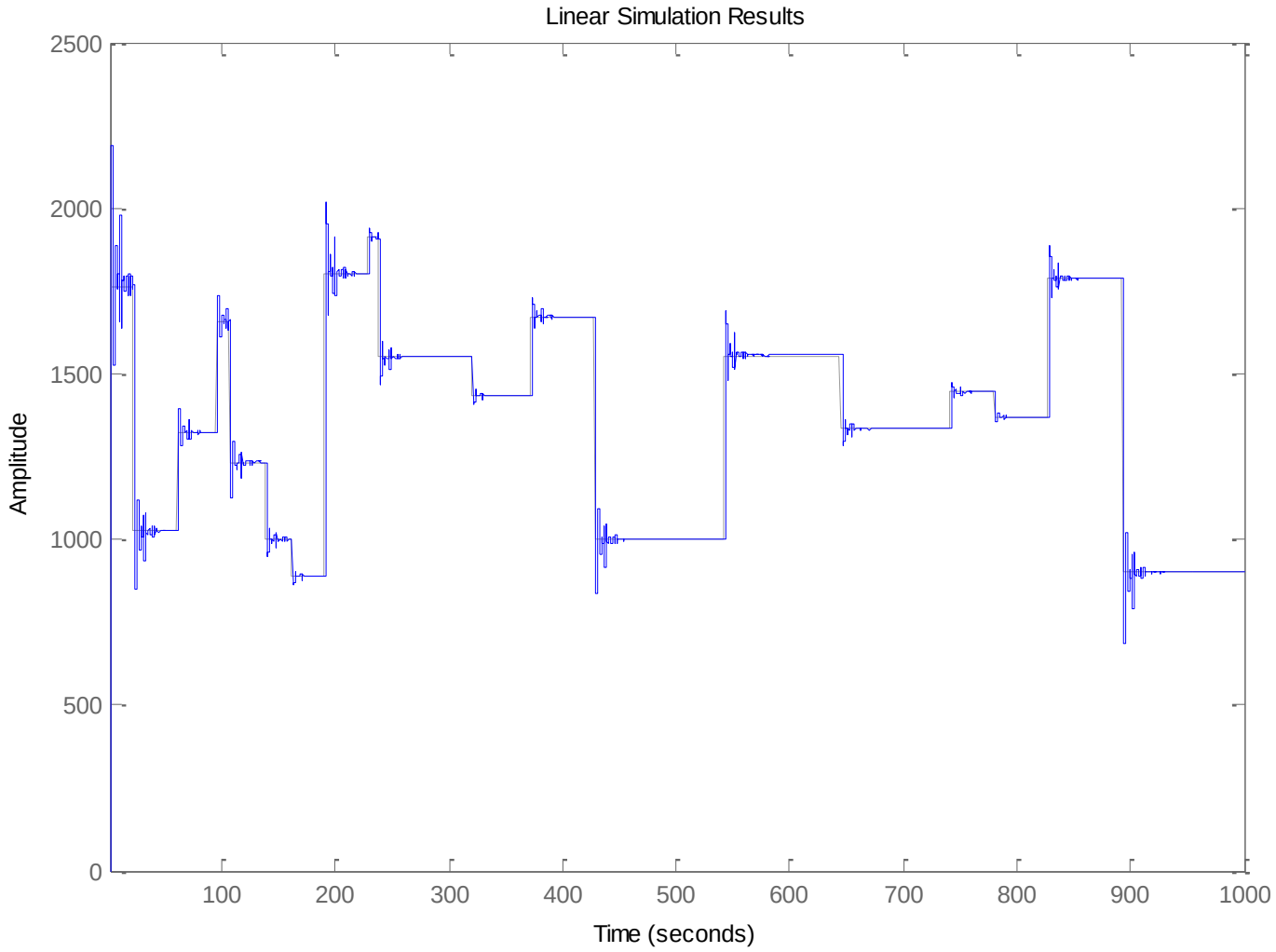


Σχήμα 4.46 – Απόκριση στον χρόνο του κλειστού βρόχου. Autotuning -> Freq. Ziegler - Nichols

MAE (Mean Absolute Error) = 21.6978 RPM
Ποσοστό υπερύψωσης: Mp = 36.1%

4.7.2 Approximate MIGO – απόκριση συχνότητας

Kp = 0.326, Ki = 0.141, Kd = 0.0695 (καλύτερα αποτελέσματα)

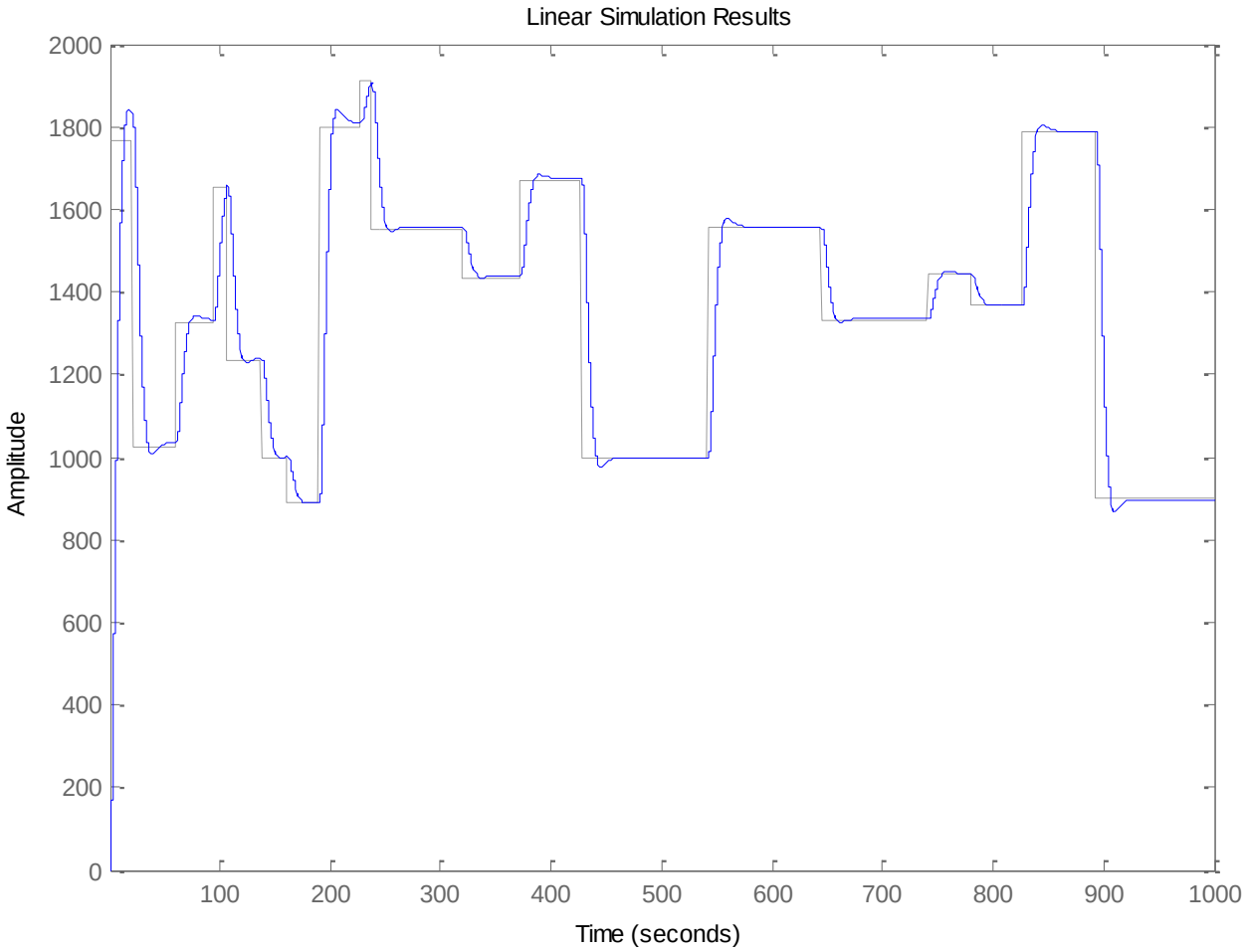


Σχήμα 4.47 – Απόκριση στον χρόνο του κλειστού βρόχου. Autotuning -> Approximate MIGO

MAE (Mean Absolute Error) = 17.7 RPM
Ποσοστό υπερύψωσης: Mp = 24.1 %

4.7.3 Αυτόματη (balanced performance and robustness)

Kp = 0.0153, Ki = 0.0153, Kd = 0.00382



Σχήμα 4.48 – Απόκριση στον χρόνο του κλειστού βρόχου. Autotuning -> Robust Automated

MAE (Mean Absolute Error) = 59.3 RPM
Ποσοστό υπερύψωσης: Mp = 2.03 %

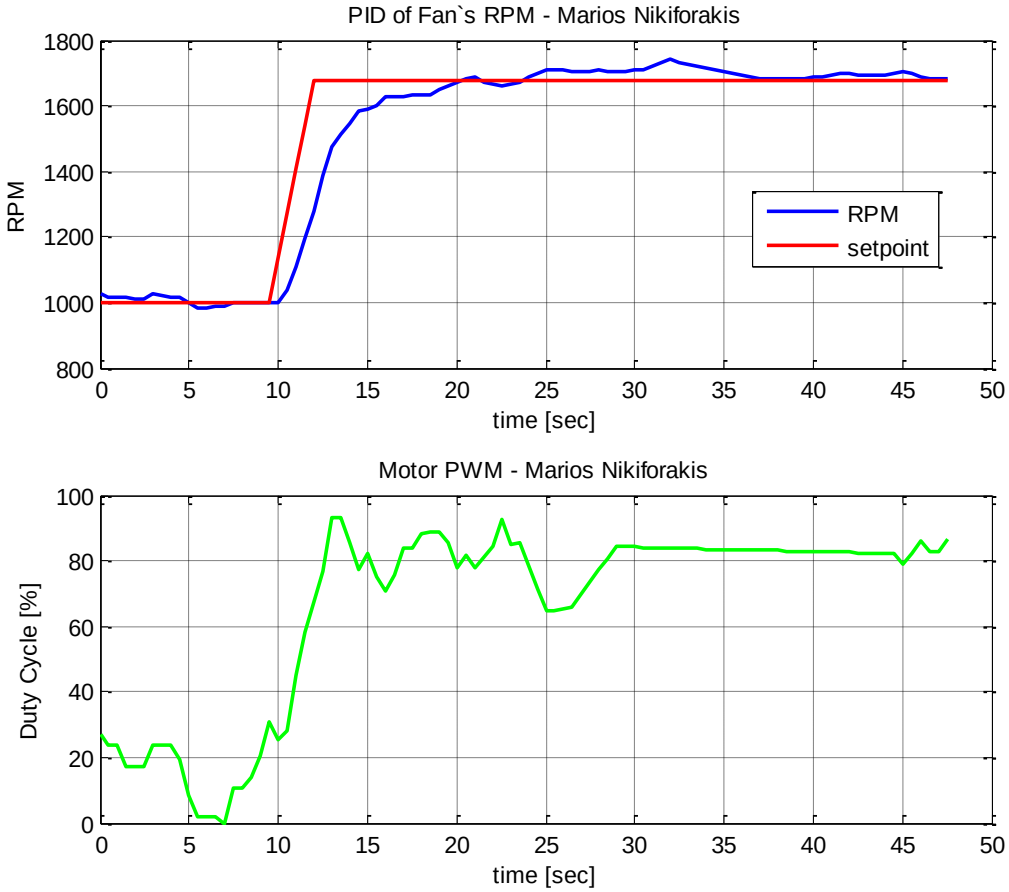
Από τα σχήματα 4.46, 47 και 48 συμπεραίνουμε πως οι πρώτες δύο προσεγγίσεις μοιάζουν αρκετά και ως προς του συντελεστές αλλά και ως προς τα αποτελέσματα, ενώ η τρίτη η οποία έγινε με τελείως διαφορετική μεθοδολογία έχει και τελείως διαφορετικό αποτέλεσμα.

Έτσι, αν και το σφάλμα είναι αρκετά μεγαλύτερο στην τελευταία περίπτωση, δεν έχουμε καμία υπερύψωση, γεγονός το οποίο να είναι περισσότερο επιθυμητό από το να έχουμε κάποιο σφάλμα. Έτσι σε εκείνη την περίπτωση θα είχαμε επιλογή των τρίτων παραμέτρων για τον ελεγκτή PID. Παρόλα αυτά το καλύτερο tuning φαίνεται να έχει γίνει στην περίπτωση του autotuning με την μέθοδο **"Approximate MIGO – απόκριση συχνότητας".**

4.8 Πειραματικά και θεωρητικά αποτελέσματα

Οι συντελεστές Kp, Ki και Kd που προέκυψαν από το autotuning μέσω του θεωρητικού μοντέλου ήταν kp=0.36, ki=0.14 και kd=0.06. Οι παράμετροι αυτοί αν και διαφέρουν από αυτές που εκτιμήσαμε με το manual tuning παραπάνω, δείχνουν πως σαν μεγέθη το kp πρέπει να είναι μεγαλύτερο του ki και kd, το ki μεγαλύτερο από το kd και το kd αρκετά μικρό.

Εισάγοντας τους συντελεστές που εξήχθησαν από το autotuning του θεωρητικού μοντέλου στην πειραματική κατασκευή, έχουμε την απόκριση που φαίνεται στο σχήμα 4.49. Θα πρέπει να σημειωθεί πως σε αυτό το σημείο γίνεται η μετατροπή των συντελεστών κατάλληλα λόγο της διαφορετικής δειγματοληψίας του GUI του Matlab σε σχέση με την δειγματοληψία του PID του Arduino. Η δειγματοληψία του GUI είναι 0.5s, ενώ του PID του Arduino είναι 0.2s.



Σχήμα 4.49 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=0.36, Ki=0.14(0.2/0.5)= 0.05, Kd=0.06*(0.5/0.2)= 0.15*

Τα μεγέθη των παραμέτρων είναι σε επίπεδο κάτω του 1 και έχουν κοινά χαρακτηριστικά, τόσο για το auto tuning όσο και για το manual tuning.

Παρακάτω φαίνονται οι αναλογικές παράμετροι των PID οι οποίες υπολογίζονται με βάση τις συχνότητες δειγματοληψίας του κάθε συστήματος. Έτσι για τον υπολογισμό των Ki και Kd (αναλογικές) έχουμε τις εξής σχέσεις:

$$Ki_{contARD} = ki_{DiscreteArd} \cdot Ts_{Arduino} \tag{4.42}$$

$$Kd_{contARD} = \frac{kd_{DiscreteArd}}{Ts_{Arduino}} \tag{4.43}$$

Και

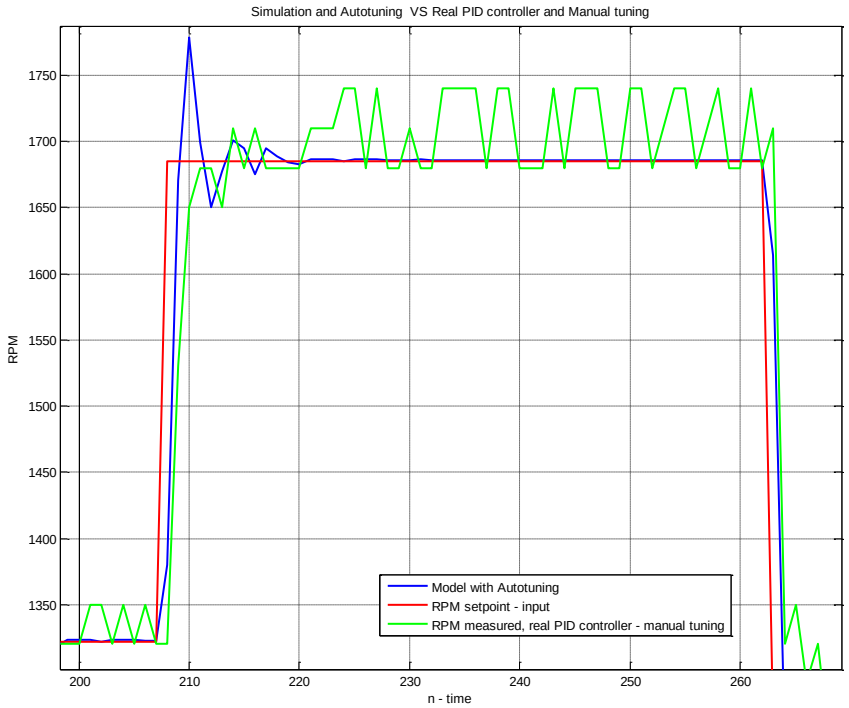
$$Ki_{contGUI} = ki_{DiscreteGUI} \cdot Ts_{GUI} \tag{4.44}$$

$$Kd_{contGUI} = \frac{kd_{DiscreteGUI}}{Ts_{GUI}} \tag{4.45}$$

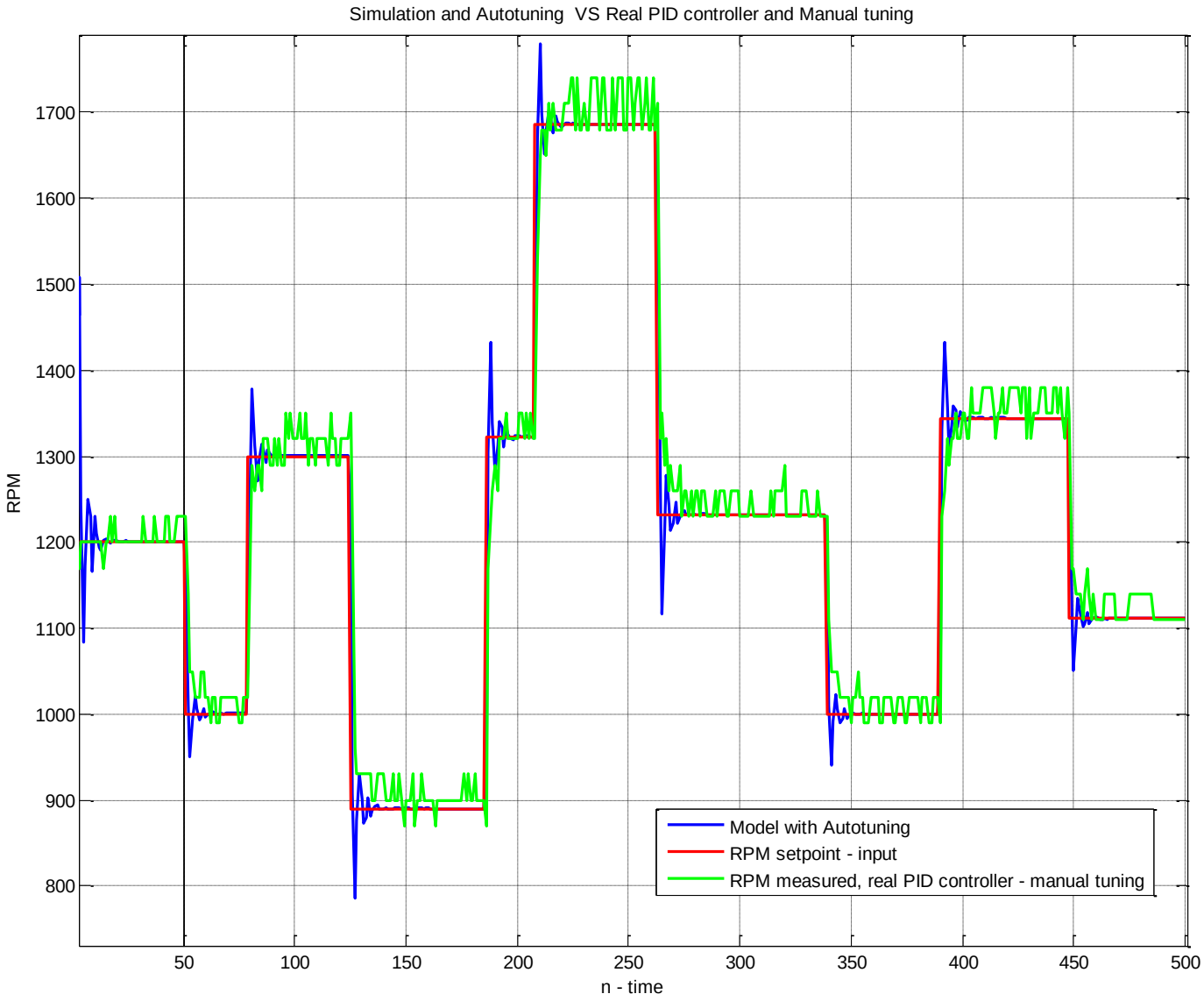
Πίνακας 4.3 – Σύγκριση Auto tuning και Manual Tuning

Tuning	Σύστημα	Kp	Ki * Ts	Kd / Ts
Auto	Θεωρητικό μοντέλο – SISOtool	**0.36**	0.14*0.5 = **0.2**	0.06/0.5 = **0.12**
Manual	Ανεμιστήρας - Arduino PID	**0.67**	0.64*0.2 = **0.12**	0.01/0.2 = **0.05**

Σε αυτή την φάση έγινε ένα γενικό πείραμα μιας ολοκληρωμένης διαδικασίας ελέγχου στροφών ανεμιστήρα αρχικά μέσω της πειραματικής κατασκευής και των συντελεστών kp, ki και kd που εξήχθησαν από το manual tuning και στην συνέχεια για τις ίδιες εισόδους έγινε η προσομοίωση με το θεωρητικό μοντέλο και τους συντελεστές που προέκυψαν από το sisotool του Matlab. Στα σχήματα 4.50 και 4.51 φαίνεται αυτή η σύγκριση.



Σχήμα 4.50 – Σύγκριση μαθηματικού μοντέλου και autotuning με πραγματικό ελεγκτή και manual tuning, σε βηματική είσοδο.



Σχήμα 4.51– Σύγκριση μαθηματικού μοντέλου και autotuning με πραγματικό ελεγκτή και manual tuning σε συνολικό πείραμα 2 λεπτών.

4.9 Συμπεράσματα

Για την εκτίμηση του μοντέλου συμπεραίνουμε πως το άγνωστο σύστημα του ανεμιστήρα μοντελοποιείται καλύτερα μέσω μη γραμμικού μοντέλου, όπως φαίνεται στον πίνακα 4.1. Παρόλα αυτά τα μη γραμμικά μοντέλα είναι δύσκολα για θεωρητική ανάλυση, όσο και για την σύνδεση τους με άλλα συστήματα μέσω Matlab. Η λογική του sisotool του Matlab στηρίζεται στην συνάρτηση μεταφοράς του συστήματος, πράγμα που δεν είναι εύκολο για τα μη γραμμικά συστήματα.

Ως μοντέλο του συστήματος τελικά επιλέγεται το ARMAX 10 10 10 1 για την γενικότερη ανάλυση, ενώ για το auto tuning χρησιμοποιείται το μοντέλο ARX 10 10 1 αφού είναι το κοντινότερο στο βέλτιστο που μπορεί να επεξεργαστεί το sisotool του Matlab.

Η μοντελοποίηση του συστήματος έγινε με επιτυχία, αφού τα ποσοστά ομοιότητας που εξήχθησαν ήταν πάνω από **80%**. Στο σχήμα 4.41 παρατηρούμε τα γραφήματα του καλύτερου μη γραμμικού και γραμμικού μοντέλου. Φαίνεται πως το μη γραμμικό μοντέλο συγκλίνει περισσότερο προς το πραγματικό σύστημα.

Θα πρέπει να σημειωθεί πως καθώς αυξανόταν οι πόλοι και τα μηδενικά του μοντέλου, γινόταν καλύτερη προσέγγιση, αλλά ταυτόχρονα αυξανόταν η πολυπλοκότητα του μοντέλου. Έτσι η τομή ομοιότητας – πολυπλοκότητας ήταν οι δέκα πόλοι και μηδενικά.

Σε όλα τα παραπάνω μοντέλα που εξήχθησαν υπήρχε ευστάθεια αφού μέσω των γραφημάτων των πόλων και μηδενικών τους μπορούμε να δούμε πως τόσο οι πόλοι, όσο και τα μηδενικά των συστημάτων βρίσκονται εντός του μοναδιαίου κύκλου.

Από τα γραφήματα BODE των καλύτερων φίλτρων μπορούμε να συμπεράνουμε πως το σύστημα μοιάζει περισσότερο σε χαμηλοπερατό φίλτρο, αφού για χαμηλές τιμές συχνότητας έχει μεγαλύτερο μέτρο. Στην περίπτωση του OE μοντέλου φαίνεται να υπάρχει μια ζωνοπερατή περιοχή για το σύστημα, αλλά και αυτή τη φορά σε χαμηλές σχετικά συχνότητες.

Από την σύγκριση του autotuning του θεωρητικού μοντέλου με το manual tuning του πραγματικού συστήματος συμπεραίνουμε πως οι τιμές που εξήχθησαν από το auto tuning ήταν αρκετά όμοιες με αυτές από το manual tuning, αφού οι συντελεστές ήταν σε ίδιες τάξεις μεγέθους και ίδιας μορφής (μικρότεροι από 1 και kp>ki>kd). Από το παραπάνω συμπεραίνουμε πως το μοντέλο που επιλέγηκε ήταν ικανό να μας δώσει μια σωστή εκτίμηση για την μελέτη του συστήματος.

Εισάγοντας τις παραμέτρους που υπολογίστηκαν από το auto tuning στο πραγματικό σύστημα η απόκριση ήταν ικανοποιητική. Εδώ θα πρέπει να σημειωθεί πως υπήρχαν διαφορές στην απόκριση του πραγματικού συστήματος από αυτή που υπολογίστηκε με το θεωρητικό μοντέλο. Το παραπάνω μπορεί να συνέβη πρώτων λόγο της διαφορετικής συχνότητας δειγματοληψίας ελέγχου αν και έγινε η κατάλληλη μετατροπή των συντελεστών (Tsard=0.2s, Tsgui=0.5s) και δεύτερων λόγο της μοντελοποίησης.

Στα σχήματα 4.50 και 4.51 βλέπουμε την σύγκριση του βέλτιστου θεωρητικού ελεγκτή PID με τον βέλτιστο πραγματικό ελεγκτή. Παρατηρούμε πως υπάρχει διαφορά ως προς την υπερύψωση αφού το autotuning θεώρησε πως η υπερύψωση περίπου 12% είναι ικανοποιητική. Στην περίπτωση του πραγματικού ελεγκτή δε παρατηρείται υπερύψωση αλλά ταλάντωση η οποία μπορεί να οφείλεται στον κβαντισμό λόγο των πράξεων εντός του Arduino και στο σφάλμα μετρήσεως των παλμών για την μέτρηση των στροφών του ανεμιστήρα. Αυτό γιατί φαίνεται πως η τιμή των στροφών παίρνει συγκεκριμένα επίπεδα και όχι συνεχείς τιμές.

Σε όλη την διαδικασία του manual tuning παρατηρούμε πως αυξάνοντας τον συντελεστή kp αυξάνει η ταχύτητα του συστήματος, αυξάνοντας τον ki αυξάνοντας το σφάλμα θέσης του ελεγκτή και αυξάνοντας τον kd μειώνεται η ταλάντωση. Παρόλα αυτά ο κάθε συντελεστής επηρεάζει αρνητικά τους υπόλοιπους και είναι επιθυμητό να βρεθεί η τομή που θα δώσει την βέλτιστη λύση, κατά το manual tuning.

Οι τελικές συναρτήσεις που προκύπτουν για το PID είναι οι εξής:

Για το Auto tuning:

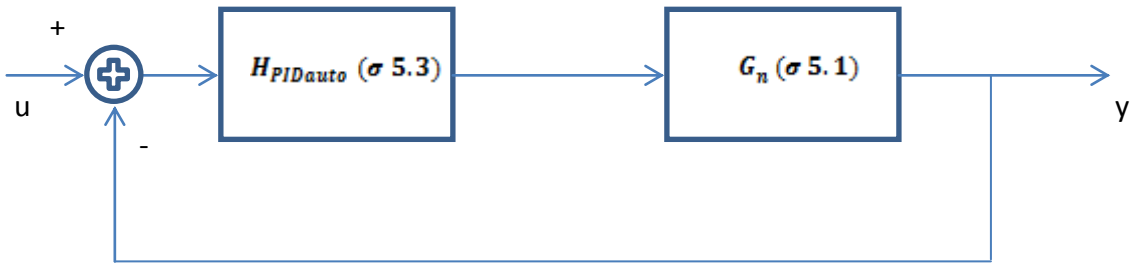
$$H_{PIDauto} = 0.36 + 0.14 \cdot \frac{0.5}{z-1} + 0.06 \cdot \frac{z-1}{0.5} = 0.36 + \frac{0.07}{z-1} + 0.12(z-1) \qquad (4.46)$$

Για το Manual tuning:

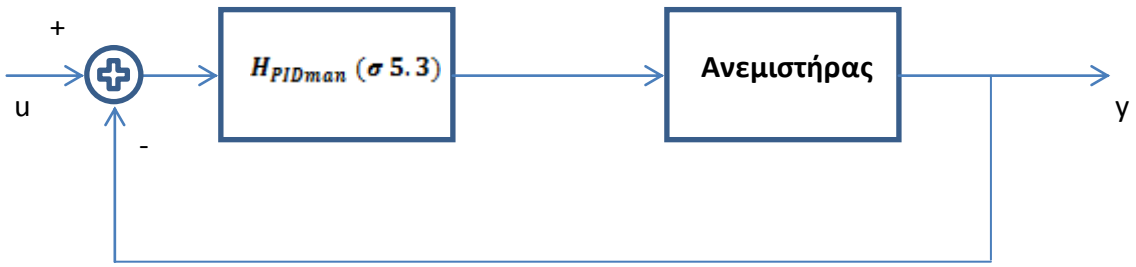
$$H_{PIDman} = 0.67 + 0.64 \cdot \frac{0.2}{z-1} + 0.01 \cdot \frac{z-1}{0.2} = \\ = 0.67 + \frac{0.128}{z-1} + 0.05(z-1) \qquad (4.47)$$

(Σχέσεις 4.46 – 4.47: συναρτήσεις μεταφοράς των ελεγκτών PID)

Άρα τα τελικά βέλτιστα για κάθε περίπτωση διαγράμματα βαθμίδων είναι αυτά που φαίνονται στα σχήματα 4.52 και 4.53.



Σχήμα 4.52 – Διάγραμμα βαθμίδων βέλτιστου θεωρητικού μοντέλου ελέγχου στροφών.

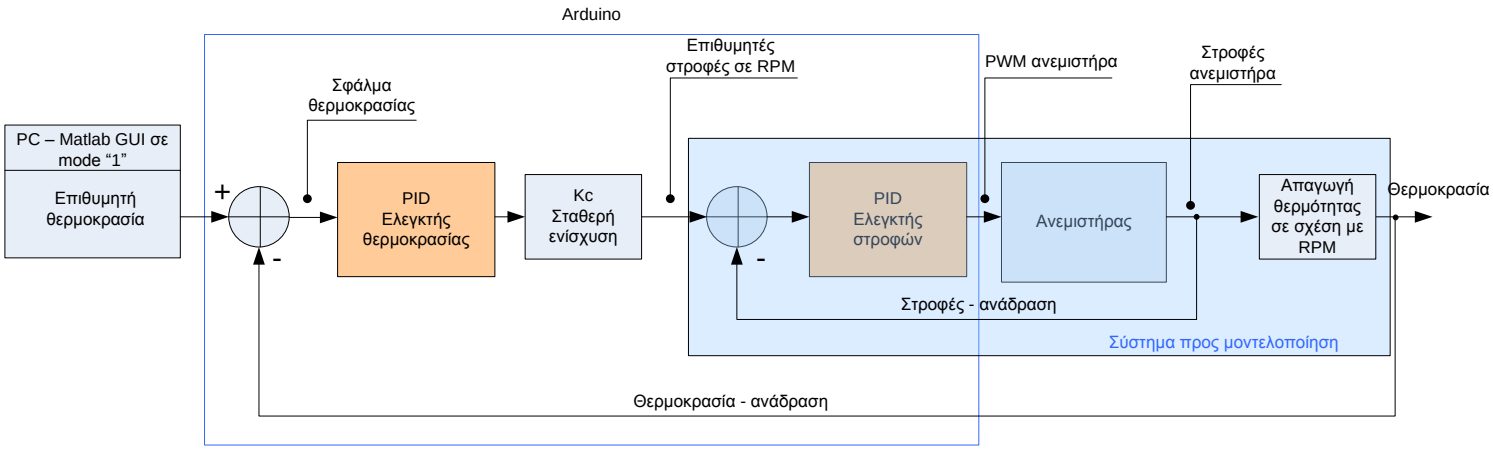


Σχήμα 4.53 – Διάγραμμα βαθμίδων βέλτιστου πραγματικού συστήματος ελέγχου στροφών.

ΚΕΦΑΛΑΙΟ 5
Έλεγχος θερμοκρασίας

5.1 Εισαγωγή

Ο έλεγχος θερμοκρασίας γίνεται μέσω ενός ακόμα ελεγκτή PID ο οποίος και αυτός υλοποιείται εντός του Arduino. Το σύστημα πάνω στο οποίο πρέπει να γίνει ο έλεγχος, αυτή την φορά, είναι το σύστημα ελέγχου στροφών. Έτσι υλοποιούμε ένα νέο σύστημα PID το οποίο θα ελέγχει το σύστημα ελέγχου στροφών, δηλαδή θα δίνει την επιθυμητή τιμή στροφών. Το διάγραμμα βαθμίδων του σχήματος 3.2 αποτελεί την δομή ελέγχου που υλοποιείται, ενώ μια πιο αναλυτική μορφή αποτελεί το σχήμα 5.1.



Σχήμα 5.1 – Διάγραμμα βαθμίδων έλεγχου θερμοκρασίας

Θα πρέπει να σημειωθεί πως για τον έλεγχο της συγκεκριμένης λειτουργίας από τον χρήστη (ξεχωριστά) θα πρέπει το συνολικό σύστημα να βρίσκεται στο mode "1". Έτσι θα εκτελείται ο έλεγχος στροφών του κινητήρα, όπως και εξωτερικά ο έλεγχος θερμοκρασίας.

Στο κεφάλαιο αυτό γίνεται αρχικά μια μοντελοποίηση του συστήματος που πρέπει να ελεγχθεί μέσω του νέου ελεγκτή PID που εισήχθη, μέσω εργαλείων του Matlab. Στην συνέχεια επιλέγεται η καλύτερη μοντελοποίηση, και γίνεται το auto-tuning του συστήματος μέσω εργαλείων του Matlab. Τέλος γίνεται η σύγκριση του θεωρητικού μοντέλου σε σχέση με το πραγματικό στο οποίο έχει προηγηθεί manual tuning.

Η μοντελοποίηση του συστήματος θα μελετηθεί μέσω των τύπων που έγινε και η μοντελοποίηση στο κεφάλαιο 4, δηλαδή:

- Γραμμικό παραμετρικό μοντέλο (Linear parametric model)
  - ARX
  - ARMAX
  - OE
- Μη – γραμμικό μοντέλο (non-Linear model)
  - ARX
  - Hammerstein – Wiener
- Διαδικαστικό μοντέλο (process model)

Οι παραπάνω τύποι μοντελοποίησης αναλύονται στο Κεφάλαιο 2.

5.2 Χειροκίνητη ρύθμιση του ελεγκτή – Manual tuning

Το manual tuning του πειραματικού ελέγχου έγινε ακλουθώντας πάλι τους βασικούς κανόνες της παραγράφου 4.5, οι οποίοι ορίζουν τον τρόπο αλλαγής των παραμέτρων για το σωστό tuning. Οι κανόνες και η διαδικασία είναι (για άλλη μια φορά) τα εξής:

1. Αν η έξοδος δεν αλλάζει αρκετά ή δεν έχουμε την επιθυμητή υπερύψωση ή ταχύτητα απόκρισης, τότε αυξάνουμε το αναλογικό κέρδος kp κατά 50%.
2. Αν η εσωτερική μεταβλητή RPM setpoint έχει σταθερή ταλάντωση ή γενικότερα ταλάντωση και η υπερύψωση ξεπερνά το 25%, τότε μειώνουμε το αναλογικό κέρδος kp κατά 50% και το ολοκληρωτικό κέρδος ki κατά 50%.
3. Αν η ταλάντωση της εσωτερικής μεταβλητής παραμένει, όπως και η υπερύψωση, τότε μειώνουμε το αναλογικό κέρδος kp κατά 20% και το ολοκληρωτικό ki κατά 50%.
4. Αν συμβαίνουν πάνω από 3 κορυφώσεις κατά την αλλαγή της επιθυμητής τιμής, τότε μειώνουμε το ολοκληρωτικό κέρδος ki κατά 30% και αυξάνουμε το διαφορικό κέρδος kd κατά 50%.
5. Αν η τιμή της εξόδου παραμένει για μεγάλο χρονικό διάστημα πιο κάτω από την επιθυμητή τιμή (long tail scenario), τότε αυξάνουμε το ολοκληρωτικό κέρδος ki κατά 100%.

6. Αν η τιμή της εξόδου παραμένει για μεγάλο χρονικό διάστημα πιο κάτω από την επιθυμητή τιμή (long tail scenario), χωρίς να υπάρχει ιδιαίτερη ταλάντωση, τότε αυξάνουμε το ολοκληρωτικό κέρδος ki κατά 100% και μειώνουμε το διαφορικό kd κατά 100%.

Στην συνέχεια επαναλαμβάνεται η διαδικασία από την αρχή μέχρι να πετύχουμε το επιθυμητό αποτέλεσμα.

Βήμα 1:

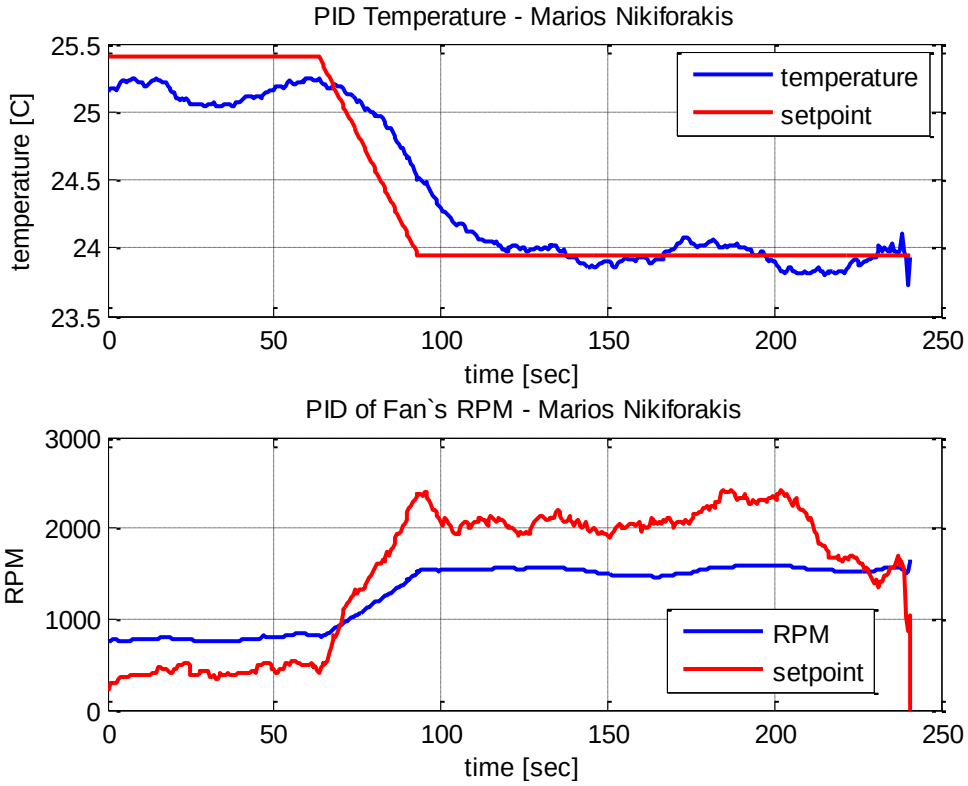
Kp = 800, Ki =50, Kd = 50

Τα αποτελέσματα που προέκυψαν φαίνονται στο σχήμα 5.2. Εύκολα διακρίνεται πως υπάρχει μικρή ταλάντωση. Έτσι στο επόμενο βήμα, θεωρώντας πως ενεργοποιείται ο κανόνας 4, μειώνεται το Ki κατά 30% και αυξάνεται το kd κατά 50%. Έτσι έχουμε:

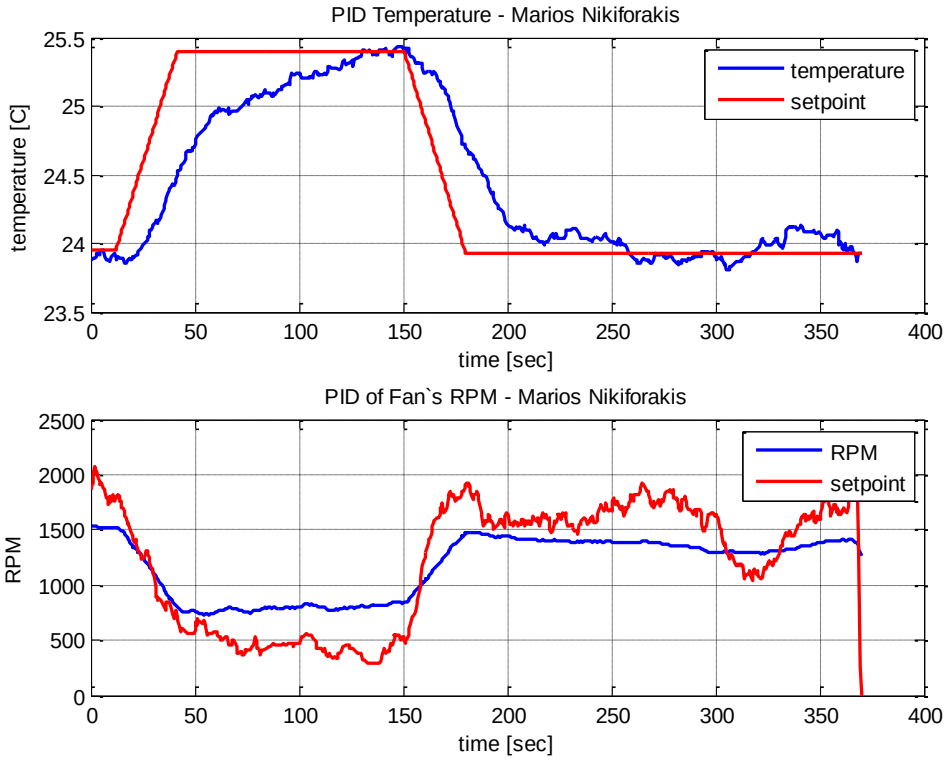
Βήμα 2:

Kp = 800, Ki =30, Kd = 70 (καλύτερα αποτελέσματα)

Τα αποτελέσματα που προέκυψαν φαίνονται στο σχήμα 6.25. Εύκολα διακρίνεται πως υπάρχει ακόμα μια μικρή ταλάντωση. Παρόλα αυτά φαίνεται να έχει αυξηθεί ο χρόνος long tail και έτσι υποθέτουμε πως ικανοποιείται ο κανόνας 5. Έτσι αυξάνεται το ki κατά 100%



Σχήμα 5.2 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=800, Ki=50, Kd=50



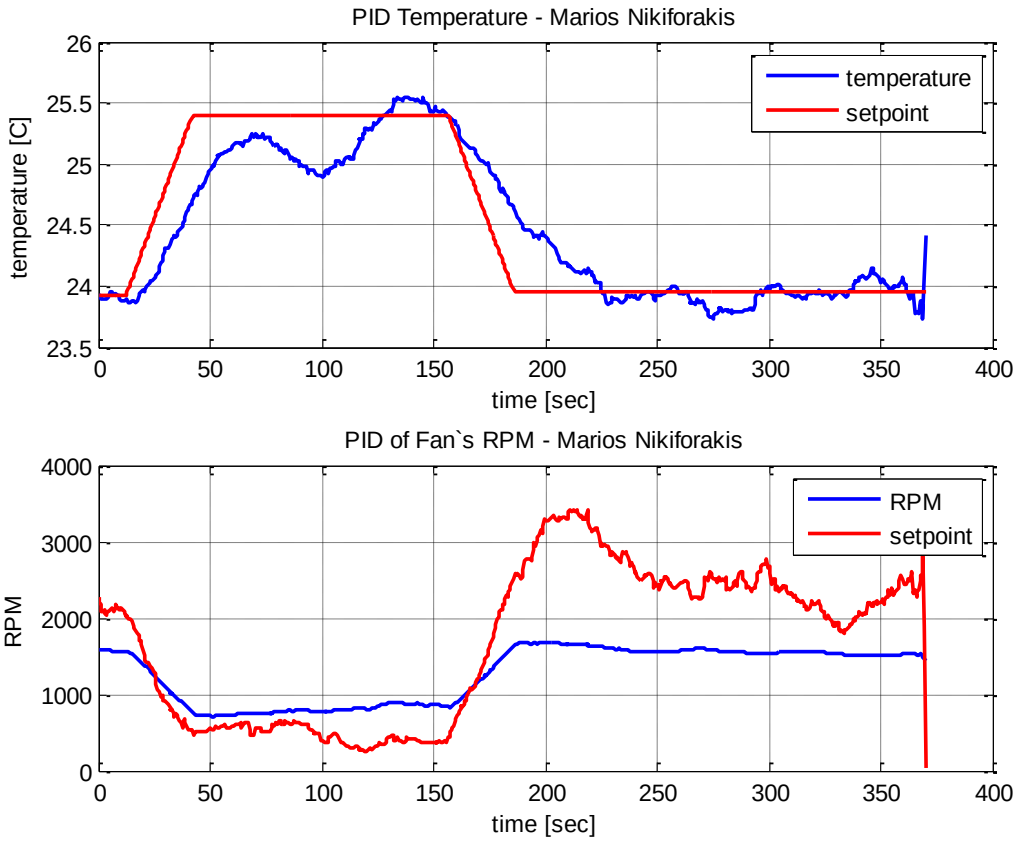
Σχήμα 5.3 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=800, Ki=30, Kd=70

Έτσι έχουμε:

Βήμα 3:

Kp = 800, Ki =60, Kd = 70

Στο σχήμα 5.3 βλέπουμε την απόκριση του βήματος 3, στο οποίο βλέπουμε πως το σύστημα έχει γίνει ελαφρώς πιο αργό. Επίσης παρατηρούμε πως η εσωτερική μεταβλητή (RPM setpoint) έχει βγει εκτός ορίων, έχοντας σαν αποτέλεσμα τον μη λειτουργικό έλεγχο και εισάγοντας παραπάνω μη – γραμμικότητες.



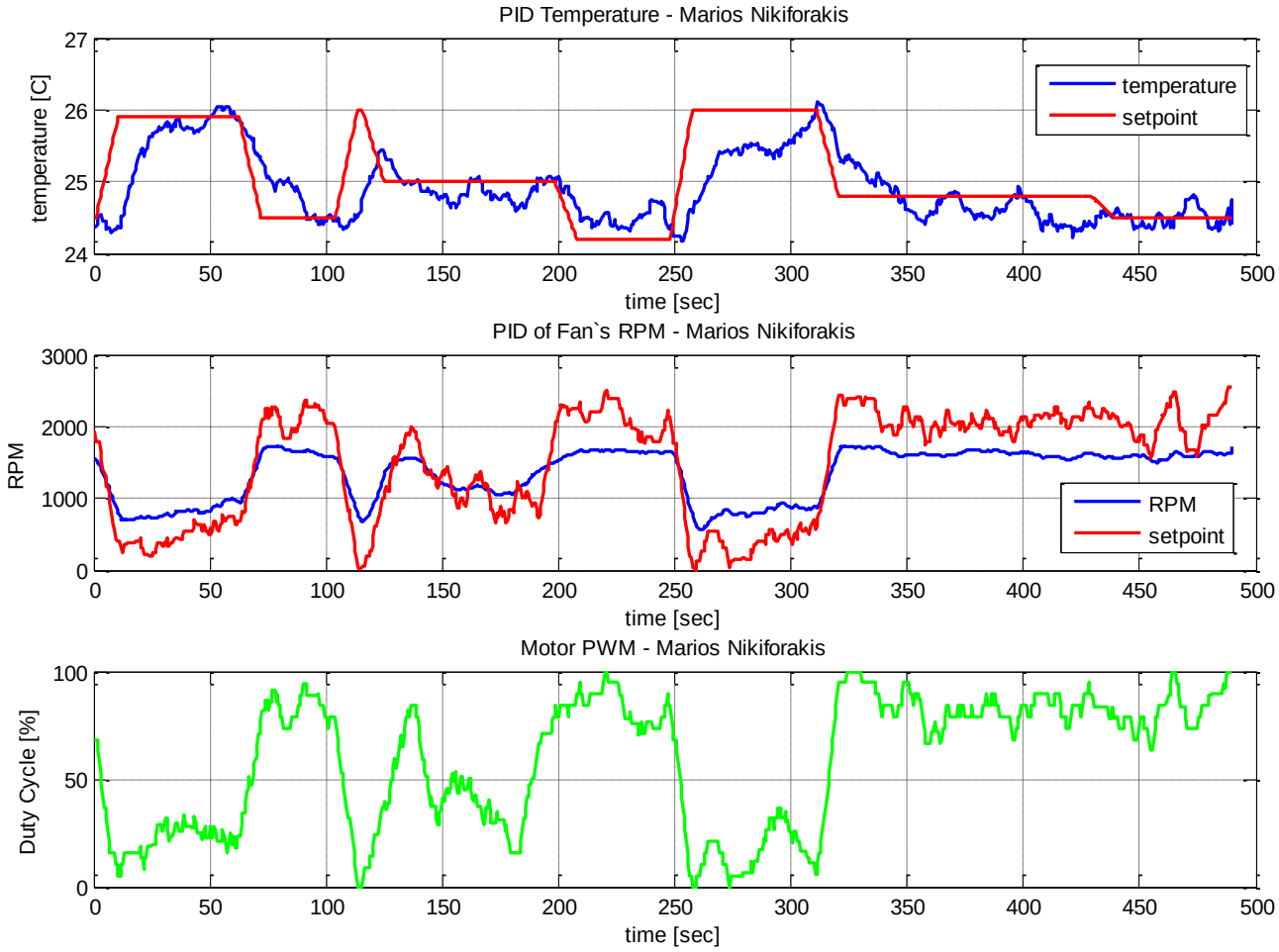
Σχήμα 5.4 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=800, Ki=60, Kd=70

Έτσι, μπορούμε να πούμε πως ο ελεγκτής κατά τον οποίο δε βγαίνει οριακά εκτός ορίων η εσωτερική μεταβλητή (RPM setpoint) αλλά έχει και την γρηγορότερη απόκριση, είναι αυτός με τις παρακάτω παραμέτρους (αυξάνοντας ελαφρώς το ki):

Kp = 800, Ki =50, Kd = 70

5.3 Εκτίμηση - Μοντελοποίηση συστήματος ψύξης

Για την μοντελοποίηση του συστήματος ψύξης χρειαζόμαστε να ακολουθήσουμε την μέθοδο που αναλύεται παρακάτω. Στην πραγματικότητα θέλουμε να εξάγουμε πληροφορία για το σύστημα που είναι σε αυτό το στάδιο το σύστημα ελέγχου στροφών και το σύστημα απαγωγής θερμότητας. Έτσι θέλουμε να βρούμε το μαθηματικό μοντέλο που εκφράζει την σχέση της εντολής σε επιθυμητές στροφές που δίνουμε προς το σύστημα, με την θερμοκρασία στον αισθητήρα.



Σχήμα 5.5 – Σετ εκπαίδευσης για την μοντελοποίηση. Διαγράμματα από μετρήσεις πειράματος ελέγχου θερμοκρασίας. Mode '1'.

Αρχικά καταγράφουμε τις τιμές της θερμοκρασίας δίνοντας συγκεκριμένες τιμές επιθυμητών στροφών. Οι μετρήσεις δε θα είναι χρονικά ανεξάρτητες, αφού όλα τα πραγματικά συστήματα έχουν κάποια μη πεπερασμένου χρόνου απόκριση. Το πείραμα είναι μεγάλου χρονικού διαστήματος και περιέχει αρκετές περιπτώσεις εναλλαγής των μεγεθών. Το set δεδομένων αυτό αποτελεί, όπως θα δούμε και παρακάτω το set

εκμάθησης της μοντελοποίησης. Στο σχήμα 5.5 βλέπουμε τα γραφήματα που αποτελούν αυτό το set δεδομένων.

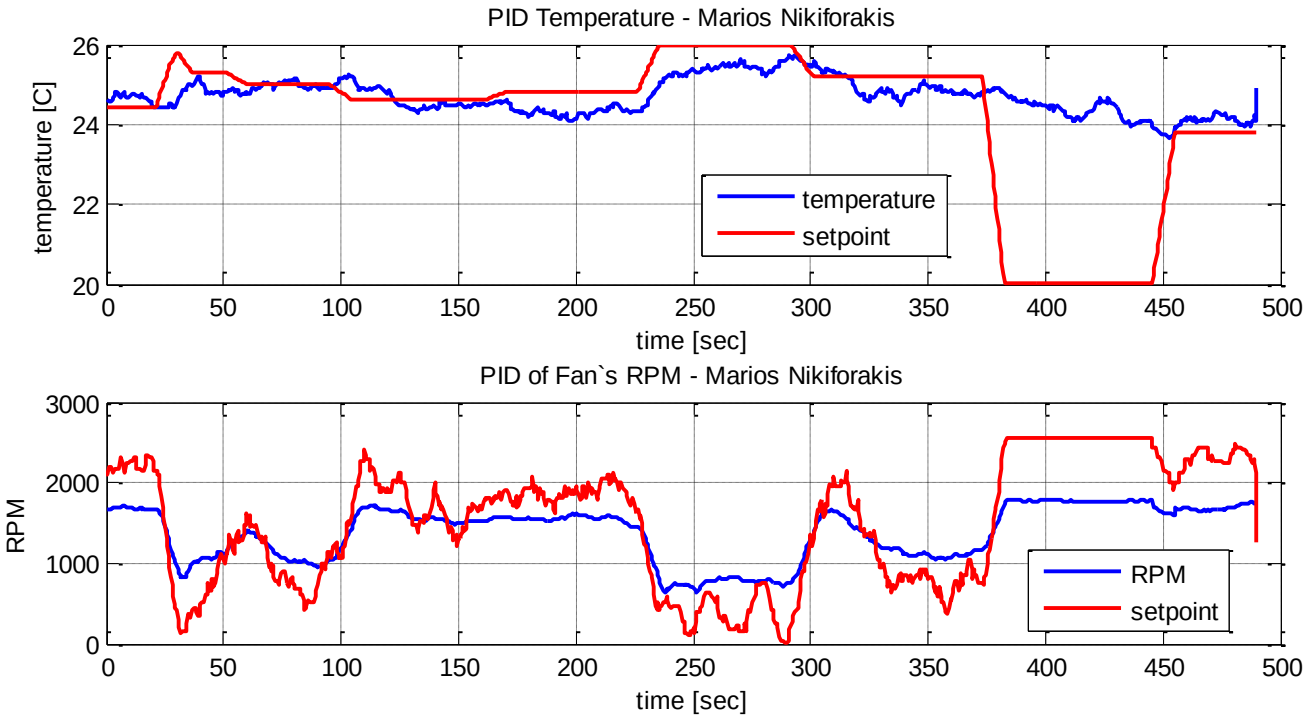
Θα πρέπει να σημειωθεί πως για το σετ δεδομένων εκπαίδευσης χρειαζόμαστε μόνο τιμές επιθυμητών στροφών και τιμές θερμοκρασίας. Έτσι χρησιμοποιούμε μετρήσεις που έγιναν με το mode λειτουργίας '0' δίνοντας τιμές των επιθυμητών στροφών και λαμβάνοντας μετρήσεις για την θερμοκρασίας. Οι παράμετροι που έχουν εισαχθεί στον ελεγκτή PID στροφών kp, ki και kd είναι αυτοί που εξήχθησαν από το manual tuning του κεφαλαίου 5, δηλαδή οι βέλτιστοι.

Από το σχήμα 5.5 μπορούμε να βγάλουμε το συμπέρασμα πως οι τιμές που παίρνουμε είναι καθώς το σύστημα βρίσκεται σε mode '1', δηλαδή λειτουργούν και οι δύο ελεγκτές, έτσι ώστε να συμπεριληφθούν και οι περιπτώσεις που το setpoint των στροφών βγαίνει εκτός ορίου του ανεμιστήρα. Τα δεδομένα είναι συγχρονισμένα και θα πρέπει να σημειωθεί πως έχουν φιλτραριστεί μέσω του φίλτρου Moving Average που αναλύεται στο Κεφάλαιο 3. Ο συνολικός χρόνος λήψης των μετρήσεων είναι 8.3 λεπτά, ικανοποιητικό χρονικό διάστημα για την μελέτη της απόκρισης του συστήματος (αργές μεταβολές στην θερμοκρασία).

Οι μετρήσεις έγιναν σε περιβάλλον οπού ήταν πλήρως αναμμένη η λυχνία (μέγιστη θέρμανση.)

Ως testing set χρησιμοποιήθηκε μέτρηση που έγινε σε άλλο χρονικό διάστημα. Το testing set που χρησιμοποιήθηκε φαίνεται στο σχήμα 5.6. Για το testing set χρησιμοποιήθηκαν τιμές που πάρθηκαν ενώ το σύστημα βρισκόταν σε mode λειτουργίας '1', δηλαδή λειτουργούσαν και οι δύο ελεγκτές και κάναμε έλεγχο μέσω επιθυμητής θερμοκρασίας.

Στην συνέχεια εισάγουμε τα δεδομένα που καταγράφθηκαν από το πείραμα στο "identification toolbox" του Matlab.



Σχήμα 5.6 – Testing set για την μοντελοποίηση.

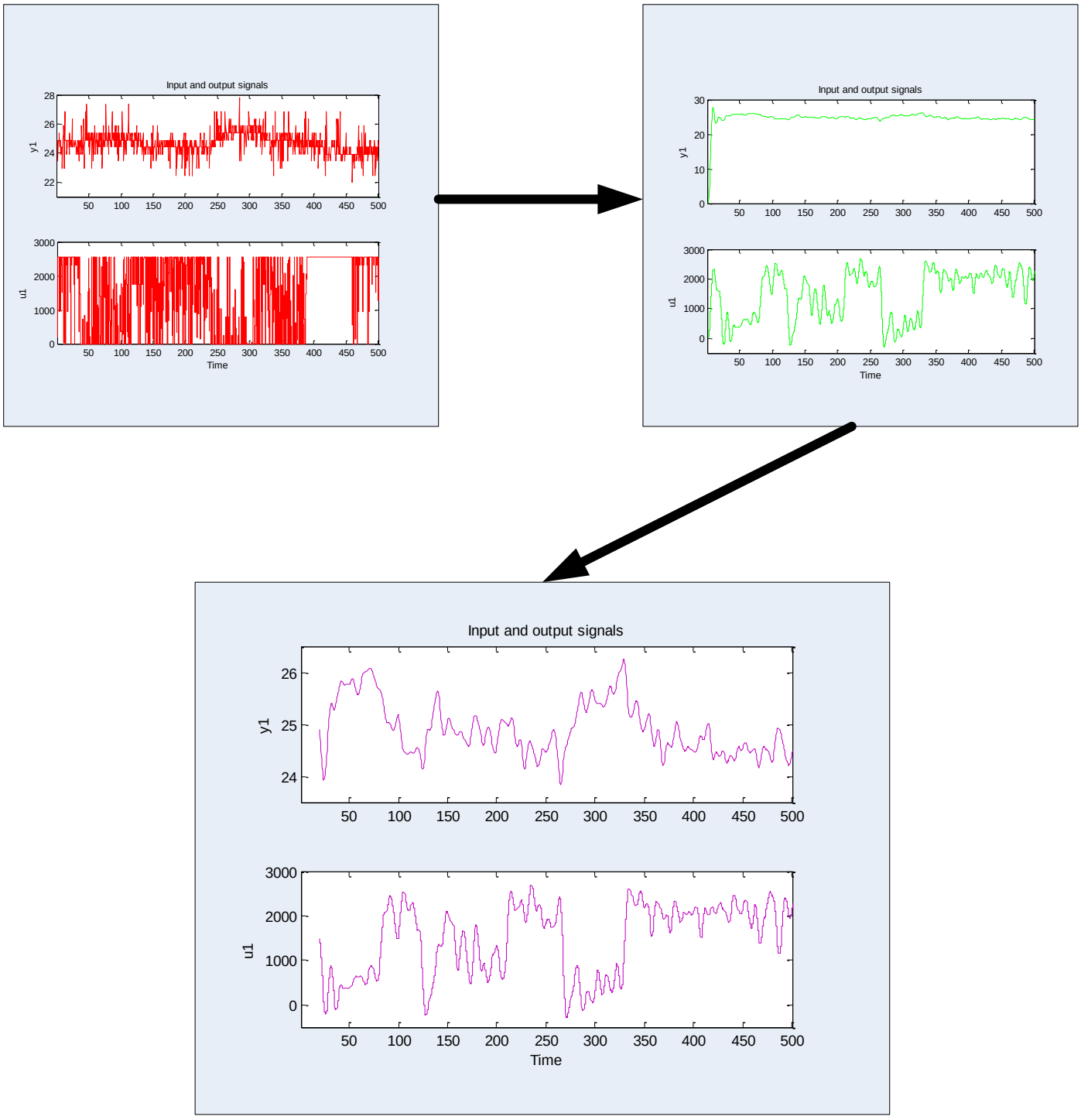
Τα δεδομένα εισόδου για την εκπαίδευση και το testing της μοντελοποίησης φιλτράρονται αρχικά κατάλληλα.

Αρχικά εισέρχονται σε ένα χαμηλοπερατό φίλτρο. Το φίλτρο αυτό αφήνει να παιρνούν κυκλικές συχνότητες από 0.01 rad/s μέχρι 0.7 rad/s. Έτσι από το πάνω αριστερά σήμα του σχήματος 5.7 περνάμε στο σήμα που φαίνεται πάνω δεξιά στο ίδιο σχήμα.

Παρόλα αυτά στο πάνω δεξιά σχήμα φαίνεται στην αρχή του σήματος εξόδου μια παράξενη απότομη μεταβολή η οποία προέκυψε έπειτα από το φιλτράρισμα. Έτσι στην συνέχεια αποκόπτονται τα πρώτα 20 δείγματα από τα σήματα έτσι ώστε τελικά να προκύψει το σήμα που φαίνεται στο κάτω μέρος του σχήματος 5.7.

Ο θόρυβος που προέκυψε και ήταν αναγκαίος να αποκοπεί ήταν σε πιο μεγάλο βαθμό θόρυβος κβαντισμού του σήματος του αισθητήρα όπως φαίνεται στις αρχικές μετρήσεις του πάνω αριστερά γραφήματος του σχήματος 5.7

Ο θόρυβος που προέκυψε και ήταν αναγκαίος να αποκοπεί ήταν σε πιο μεγάλο βαθμό θόρυβος κβαντισμού του σήματος του αισθητήρα όπως φαίνεται στις αρχικές μετρήσεις του πάνω αριστερά γραφήματος του σχήματος 5.7

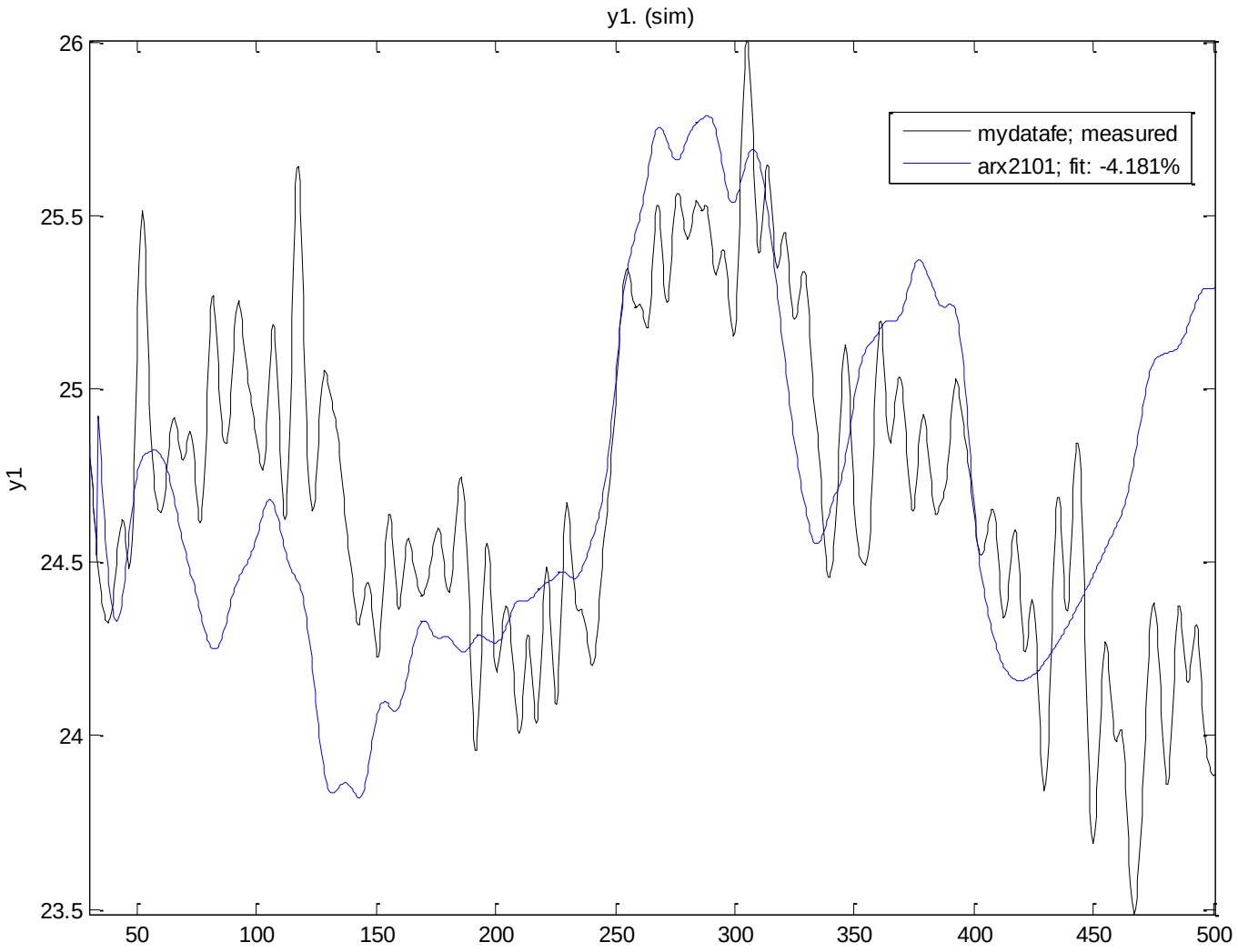


Σχήμα 5.7 – Φιλτράρισμα των σημάτων. Ενδεικτική διαδικασία.

Έτσι εισάγοντας στο 'identification toolbox' σαν είσοδο τις πραγματικές φιλτραρισμένες τιμές των επιθυμητών στροφών και σαν έξοδο τις πραγματικές τιμές θερμοκρασίας (training set σχήμα 5.5), εξάγονται τα εξής μοντέλα:

5.3.1 *Γραμμικό μοντέλο, ARX:*

Για **na = 2** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών)[5] και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.8, δίνοντας του το testing set ως είσοδο.



Σχήμα 5.8 – Αποτελέσματα μοντελοποίηση ARX na=2,nb=10, no delay

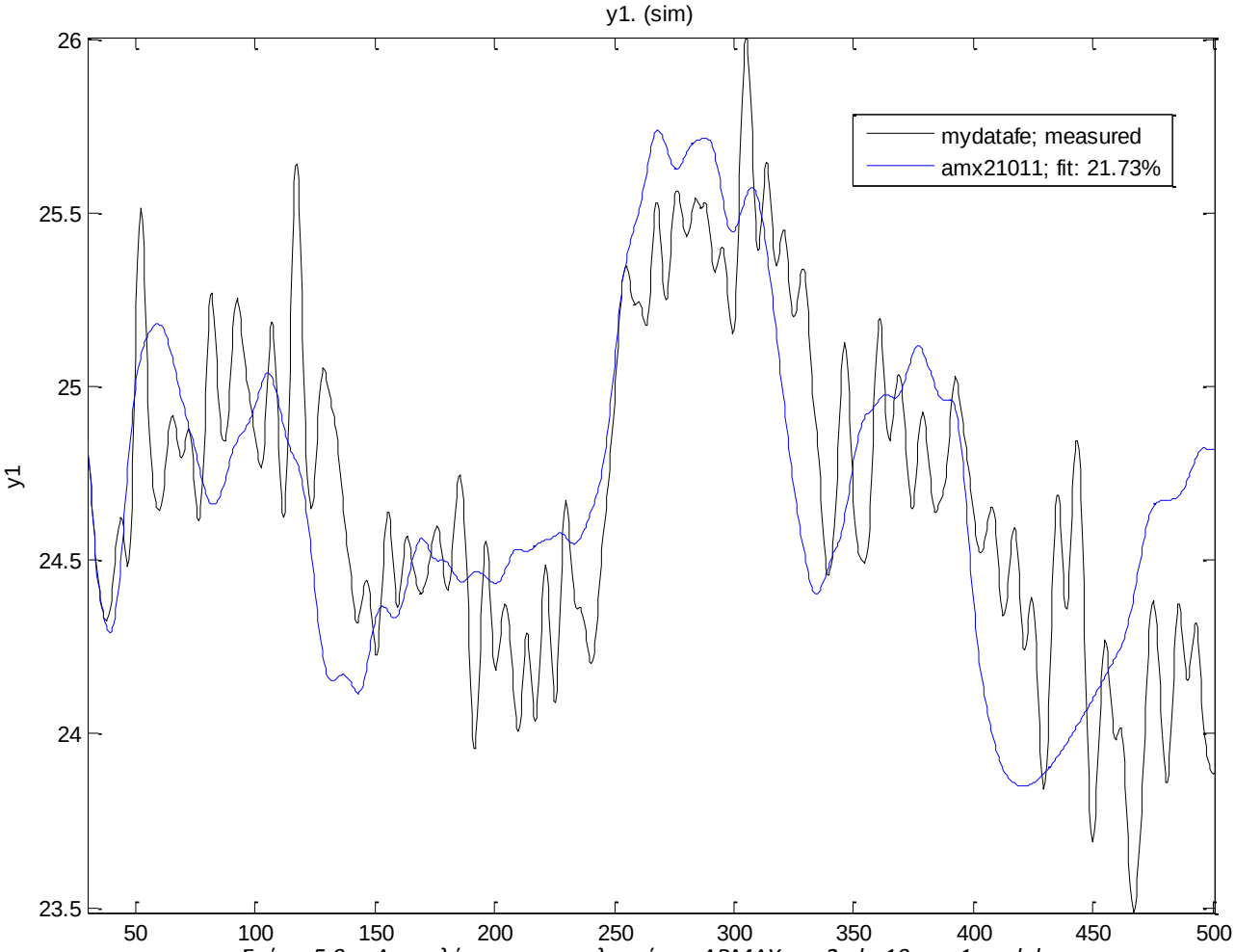
[5] Οι παράμετροι na, nb, nk όπως και όλα τα πιθανά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

Όπως φαίνεται αποτελεί μια κακή μοντελοποίηση, αφού έχουμε fit: -4.181% ομοιότητα των πειραματικών δεδομένων με αυτά που εξήχθησαν από το θεωρητικό μοντέλο.

Θα πρέπει να σημειωθεί πως αλλάζοντας τις παραμέτρους της μοντελοποίησης το αποτέλεσμα ήταν χειρότερο, αφού οι παράμετροι αυτές δίνουν το καλύτερο αποτέλεσμα για μοντελοποίηση τύπου ARX.

5.3.2 Γραμμικό μοντέλο, ARMAX:

Για **na = 2** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών), **nc = 1** (αριθμός συντελεστών στοχαστικού θορύβου) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.9, δίνοντας του το testing set ως είσοδο.



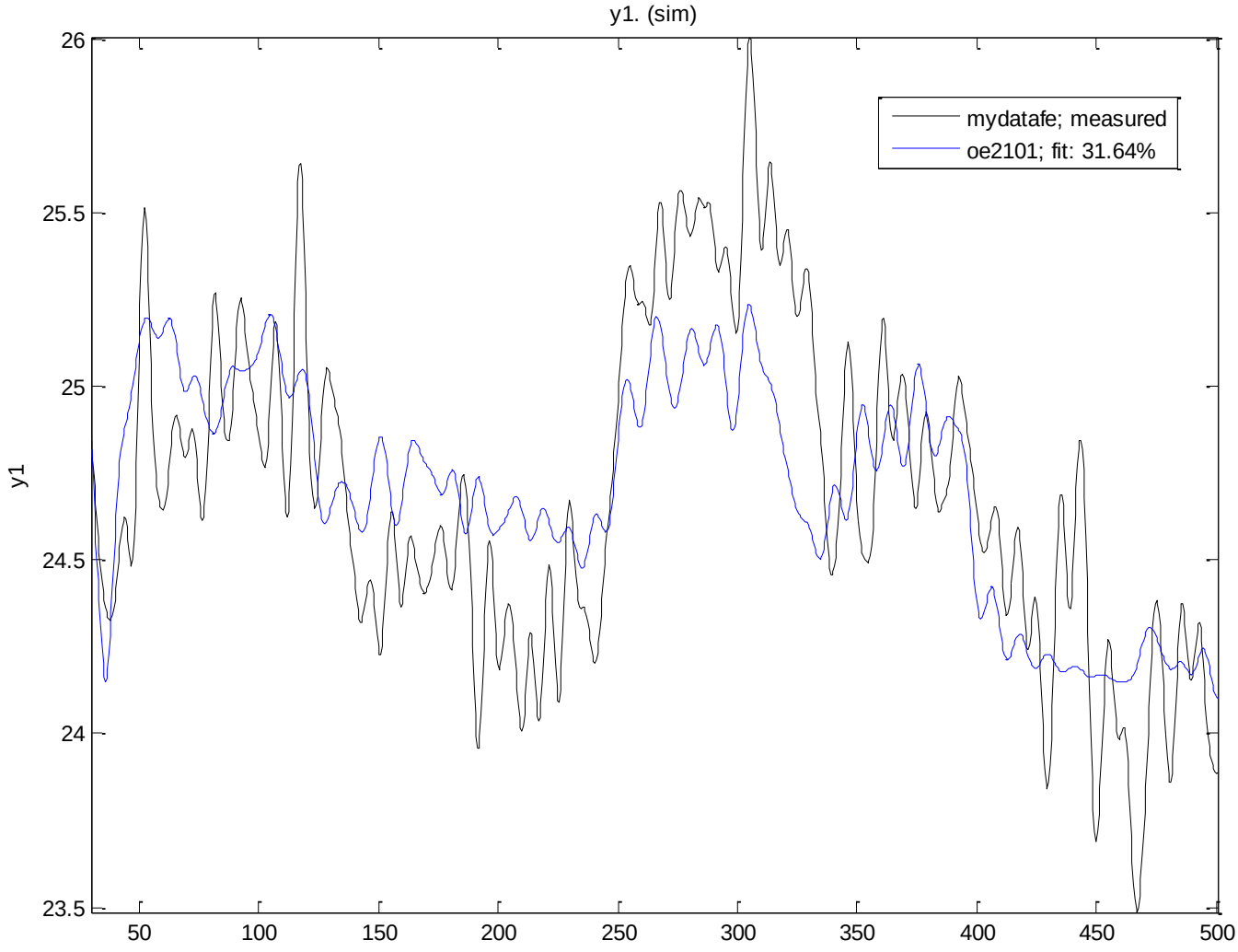
Σχήμα 5.9 – Αποτελέσματα μοντελοποίηση ARMAX na=2,nb=10, nc=1 no delay

Στο διάγραμμα του σχήματος 5.9 βλέπουμε πως έχουμε ένα μοντέλο του οποίου τα αποτελέσματα μοιάζουν σε μεγαλύτερο βαθμό από αυτό της μοντελοποίησης ARX, κατά 21.73% με τα αποτελέσματα του πειράματος.

Θα πρέπει να σημειωθεί πως αλλάζοντας τις παραμέτρους της μοντελοποίησης το αποτέλεσμα ήταν χειρότερο, αφού οι παράμετροι αυτές δίνουν το καλύτερο αποτέλεσμα για μοντελοποίηση τύπου ARMAX.

5.3.3 Γραμμικό μοντέλο, OE:

Για **na = 2** (αριθμός πόλων), **nb = 10** (αριθμός μηδενικών) και χωρίς καθυστέρηση, το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.10, δίνοντας του το testing set ως είσοδο.

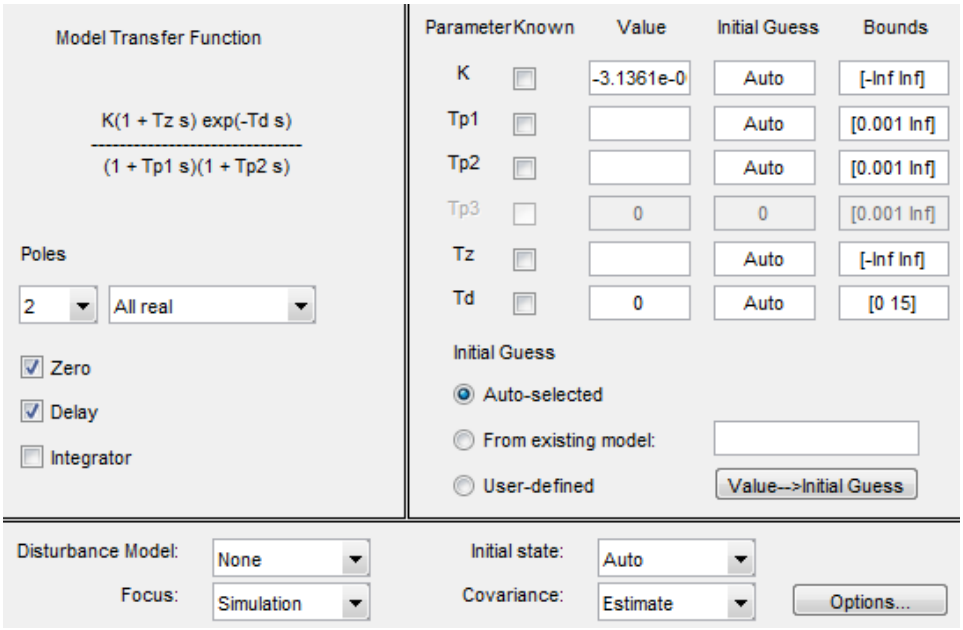


Σχήμα 5.10 – Αποτελέσματα μοντελοποίηση OE na=1,nb=10, no delay

Φαίνεται πως αποτελεί μια αρκετά καλύτερη, από τα προηγούμενα, μοντελοποίηση, αφού έχουμε fit: 31.64% ομοιότητα με τα πειραματικά δεδομένα. Παρόλα αυτά αποτελεί μια κακή σχετικά μοντελοποίηση.

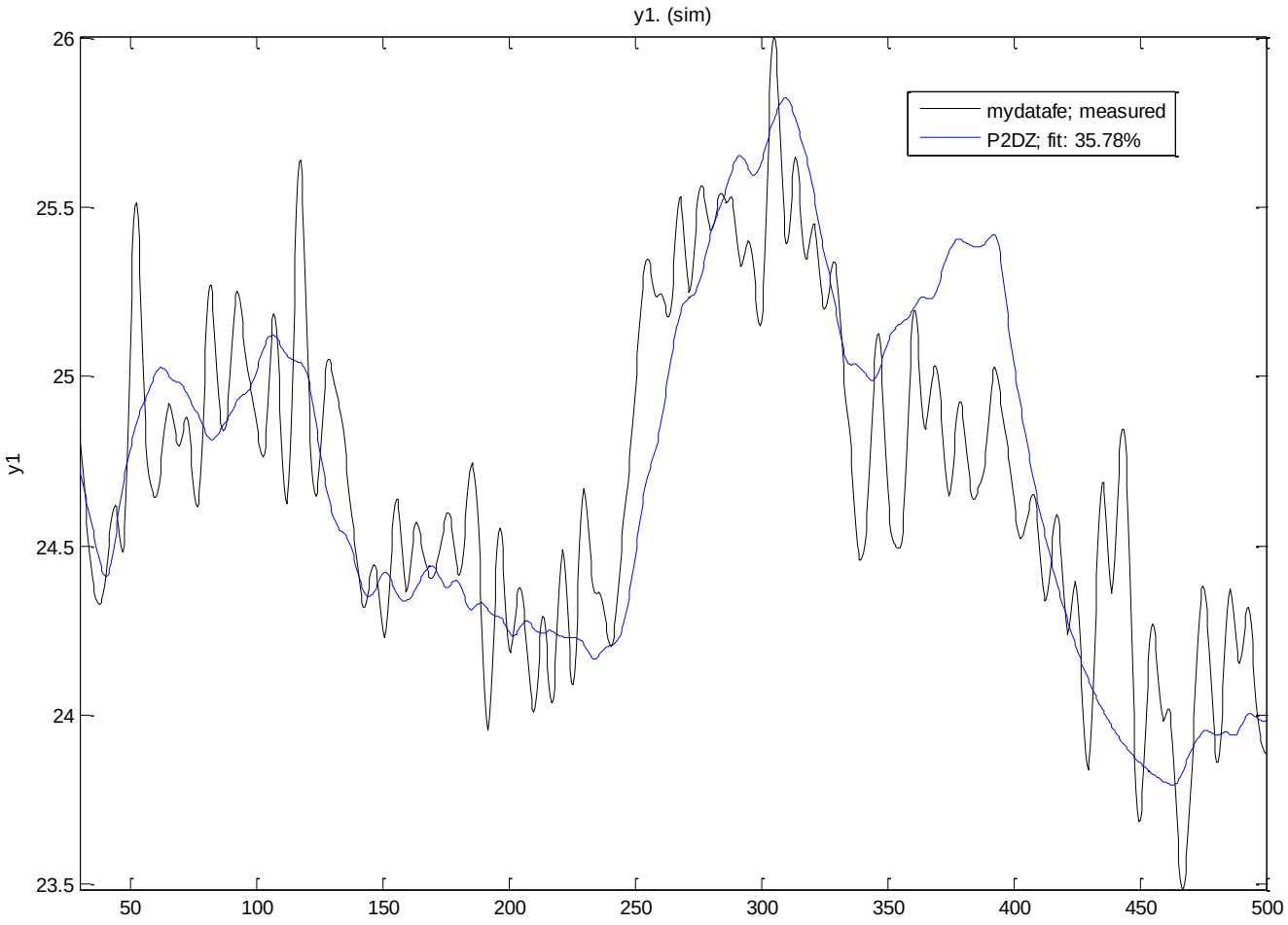
5.3.4 *Διαδικαστικό μοντέλο, Process Model :*

Για αριθμό πόλων na = 2, για καθυστέρηση και μηδενικά, το μοντέλο που προσεγγίζεται είναι αυτό που φαίνεται στο σχήμα 5.11. Επίσης στο σχήμα αυτό μπορούμε να δούμε και τον τύπο της συνάρτησης μεταφοράς που θα προκύψει από το μοντέλο.



Σχήμα 5.11 – Μοντελοποίηση μέσω process model. Na=2, delay + zero

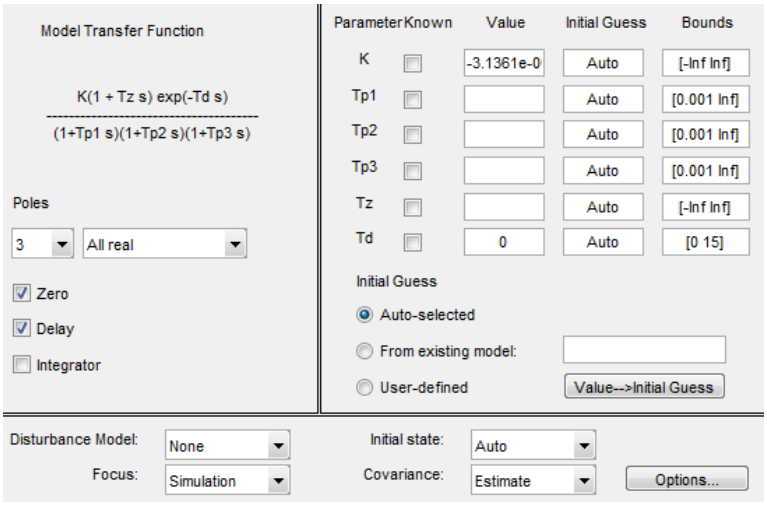
Το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.12, δίνοντας του το testing set ως είσοδο.



Σχήμα 5.12 – Αποτελέσματα μοντελοποίησης Process na=2, zero + delay

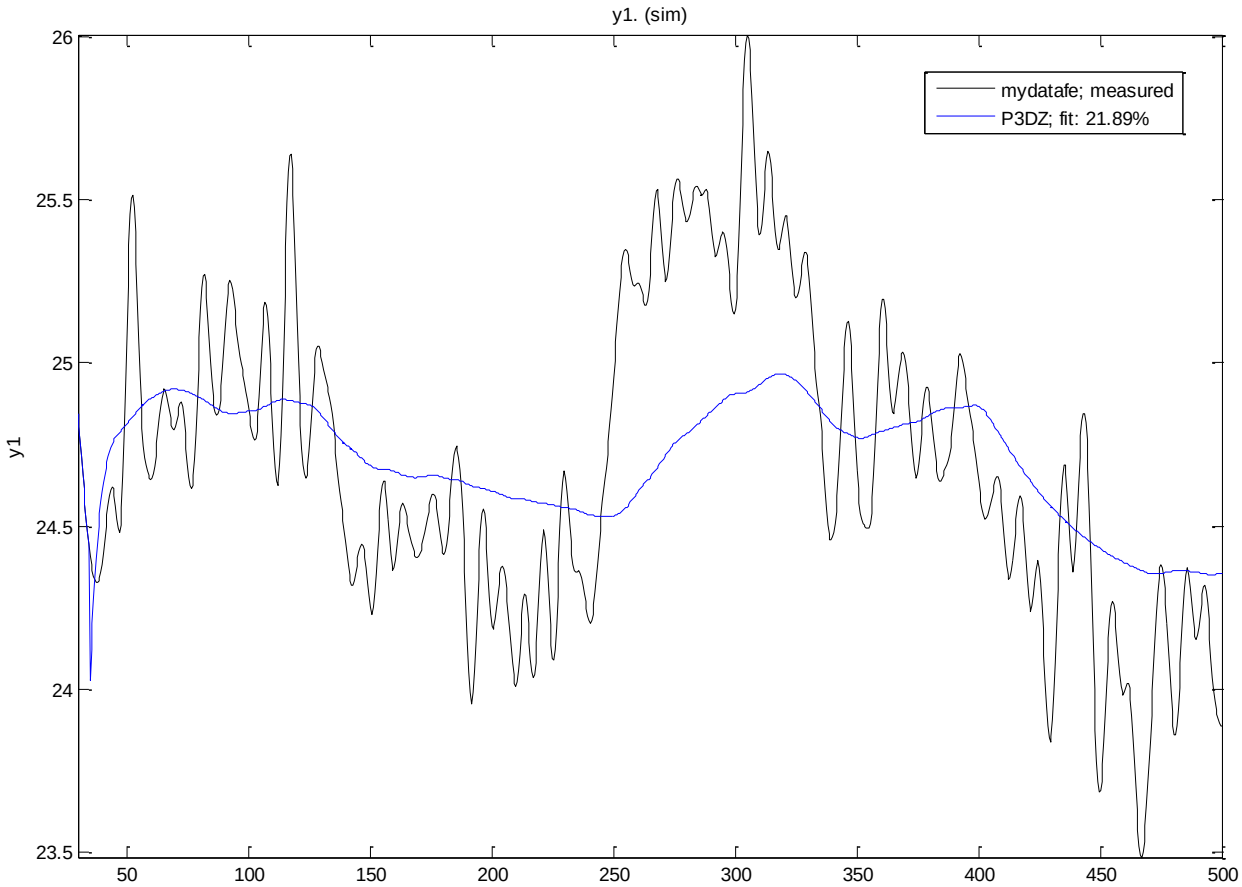
Φαίνεται πως αποτελεί μια καλύτερη, από τα προηγούμενα, μοντελοποίηση, αφού έχουμε fit: 35.78% ομοιότητα με τα πειραματικά δεδομένα.

Για αριθμό πόλων **na = 3**, για καθυστέρηση και μηδενικά, το μοντέλο που προσεγγίζεται είναι αυτό που φαίνεται στο σχήμα 5.13. Επίσης στο σχήμα αυτό μπορούμε να δούμε και τον τύπο της συνάρτησης μεταφοράς που θα προκύψει από το μοντέλο.



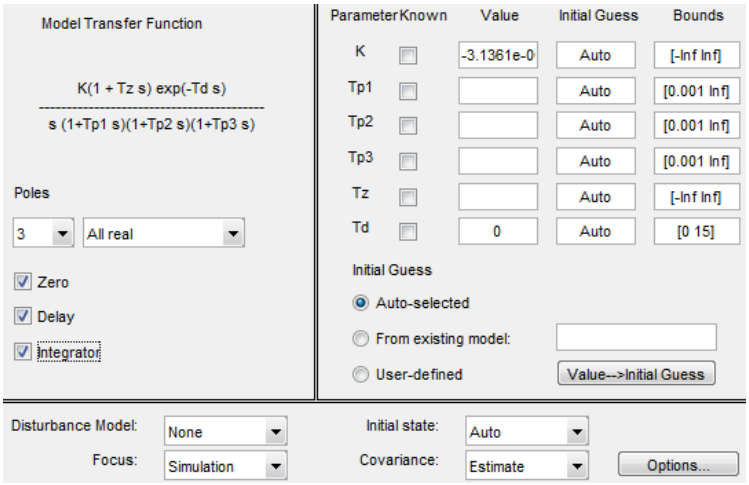
Σχήμα 5.13 – Μοντελοποίηση μέσω process model. Na=3, delay + zero

Το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.14, δίνοντας του το testing set ως είσοδο. Στο σχήμα αυτό φαίνεται πως η μοντελοποίηση είναι αρκετά χειρότερη από το προηγούμενο μοντέλο τύπου process, αφού ο βαθμός ομοιότητας είναι 21.89%.



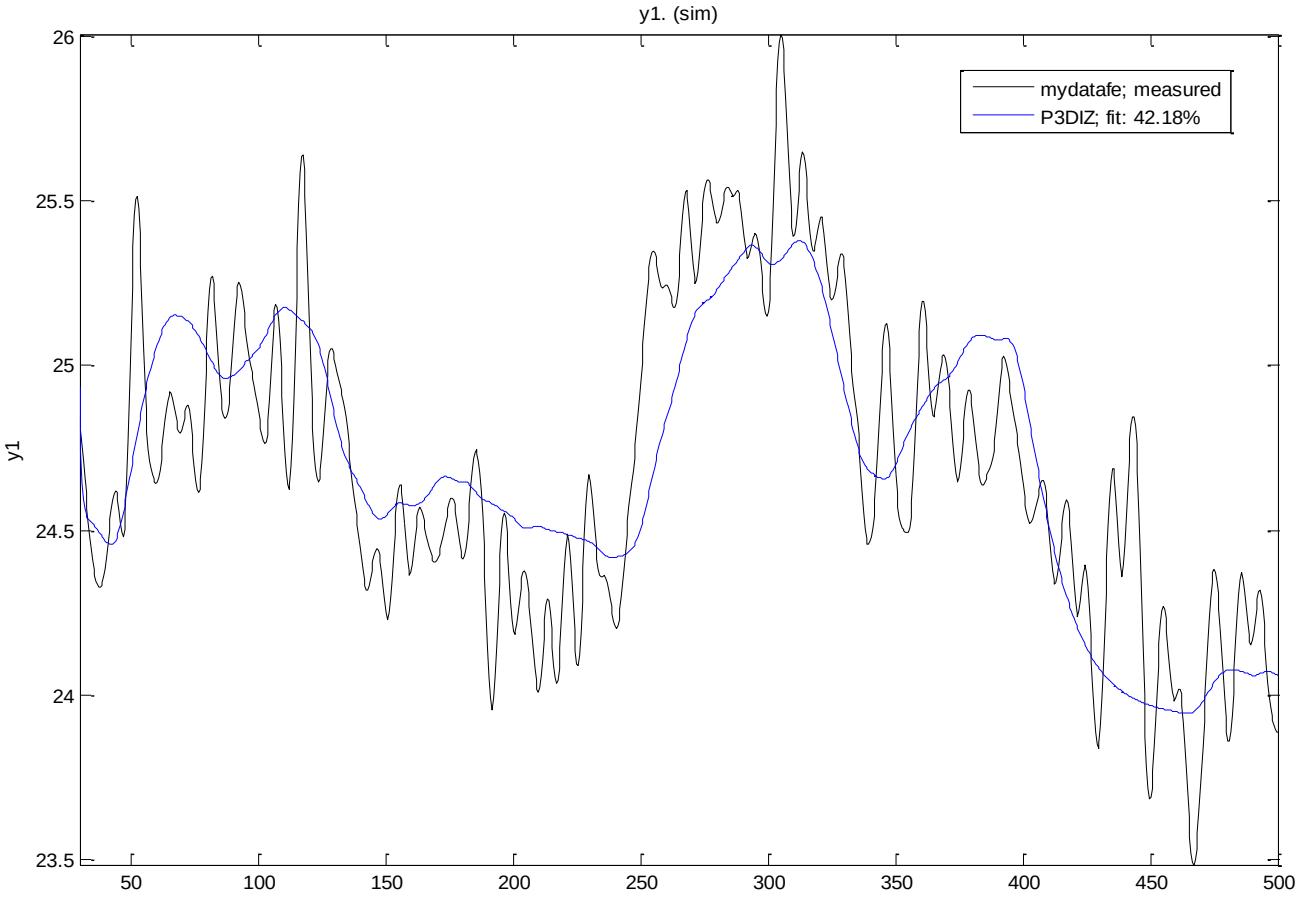
Σχήμα 5.14 – Αποτελέσματα μοντελοποίησης Process na=3, zero + delay

Για αριθμό πόλων **na = 3**, για καθυστέρηση, μηδενικά, αλλά αυτή τη φορά και ολοκληρωτή το μοντέλο που προσεγγίζεται είναι αυτό που φαίνεται στο σχήμα 5.15.



Σχήμα 5.15 – Μοντελοποίηση μέσω process model. Na=3, delay + zero + Integrator

Το γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.16, δίνοντας του το testing set ως είσοδο.

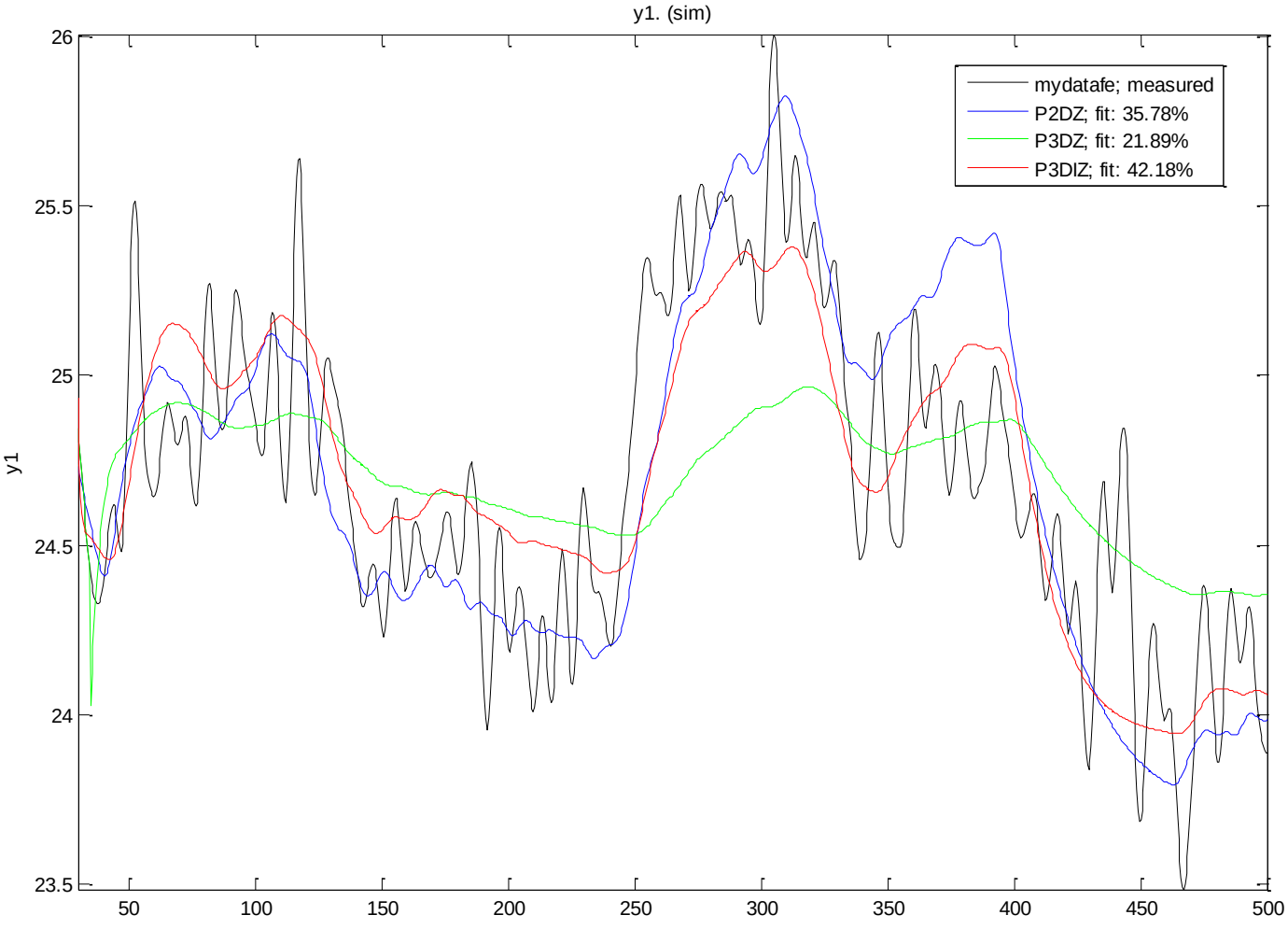


Σχήμα 5.16 – Αποτελέσματα μοντελοποίησης Process na=3, zero + delay+ Integrator

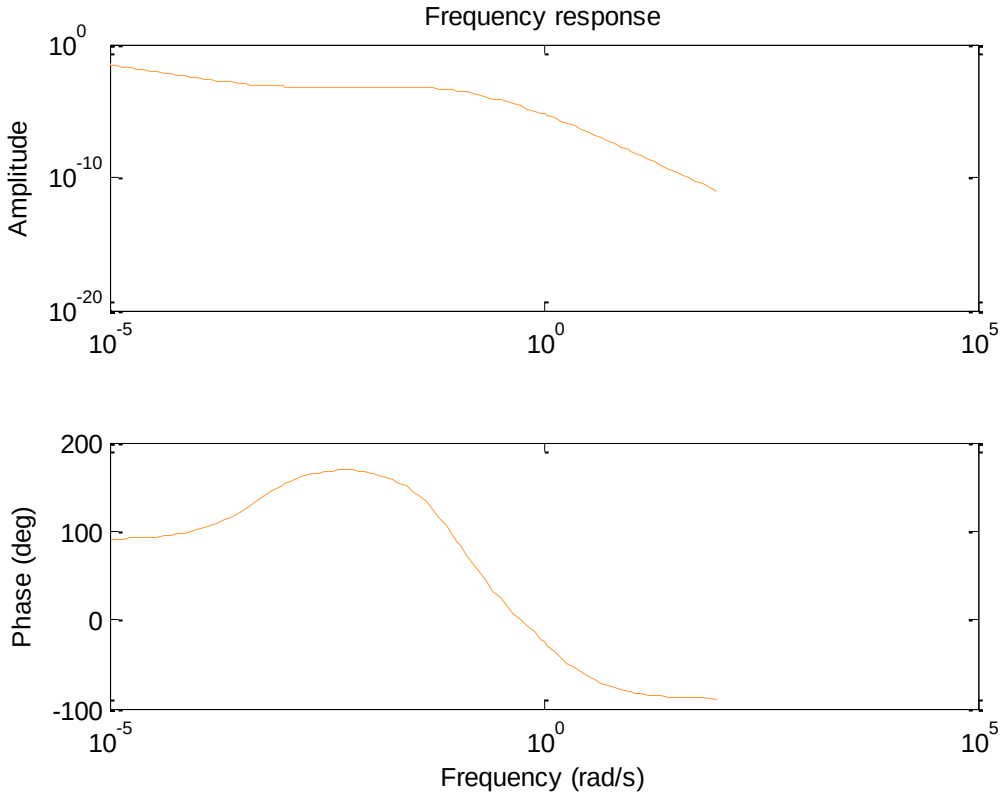
Στο σχήμα 5.16 φαίνεται πως η μοντελοποίηση είναι αρκετά καλή, σχετικά με την δυσκολία προσέγγισης του μοντέλου, αφού ο βαθμός ομοιότητας είναι 42.18%.

Στο σχήμα 5.17 βλέπουμε την σύγκριση των μοντέλων Process, στο οποίο φαίνεται ξεκάθαρα πως το μοντέλο Process na=3 (+ zero + Integrator) αποτελεί το πιο κοντινό στο πραγματικό, σύμφωνα με το testing set.

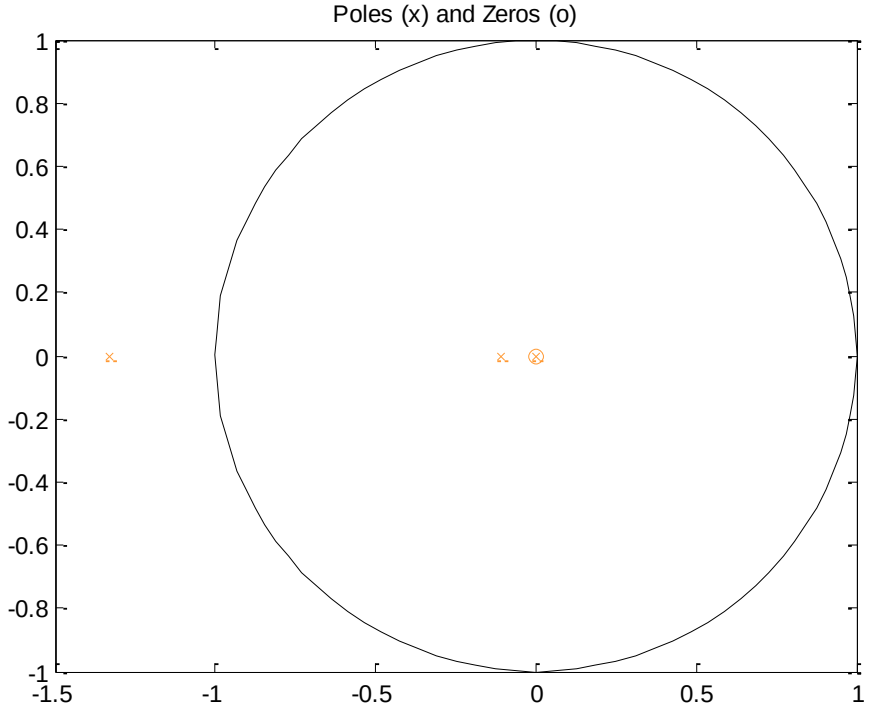
Έτσι θα μπορούσαμε να πούμε πως η καλύτερη επιλογή μοντέλου Process θα ήταν αυτή του P3DIZ, του οποίου η απόκριση συχνότητας φαίνεται στο σχήμα 5.18 και οι πόλοι – μηδενικά του στο σχήμα 5.19.



Σχήμα 5.17 – Σύγκριση μοντέλων Process



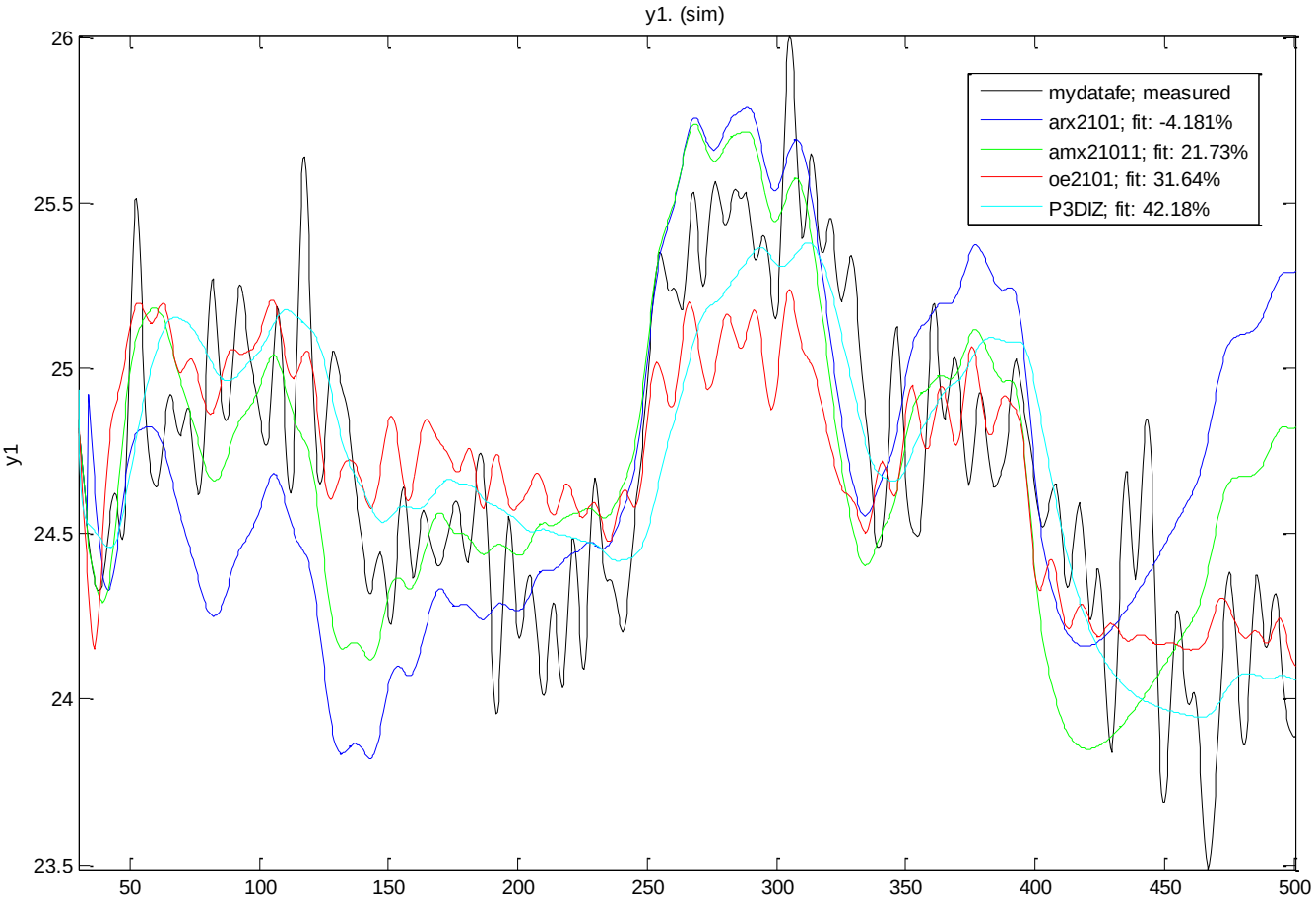
Σχήμα 5.18 – Απόκριση συχνότητας του μοντέλου Process na=3, delay, zero και Integrator



Σχήμα 5.19 –Πόλοι και μηδενικά του μοντέλου Process na=3, delay, zero και Integrator

5.3.5 *Επιλογή καλύτερου γραμμικού μοντέλου.*

Τα γραμμικά μοντέλα που μελετήθηκαν είναι τα μοντέλα ARX, ARMAX, OE και Process. Από κάθε κατηγορία έγινε η επιλογή του καλύτερου μοντέλου σύμφωνα με το testing set. Έτσι σε αυτό το σημείο θα συγκρίνουμε τα τέσσερα μοντέλα που επιλέγηκαν, δηλαδή το μοντέλο ARX 2 10 1, το μοντέλο ARMAX 2 10 1 1, το μοντέλο OE 2 10 1 και το μοντέλο Process na=3, delay, zero και Integrator. Στο σχήμα 5.20 μπορούμε να δούμε την σύγκριση των μοντέλων αυτών μέσω των αποτελεσμάτων τους στο testing set δεδομένων.

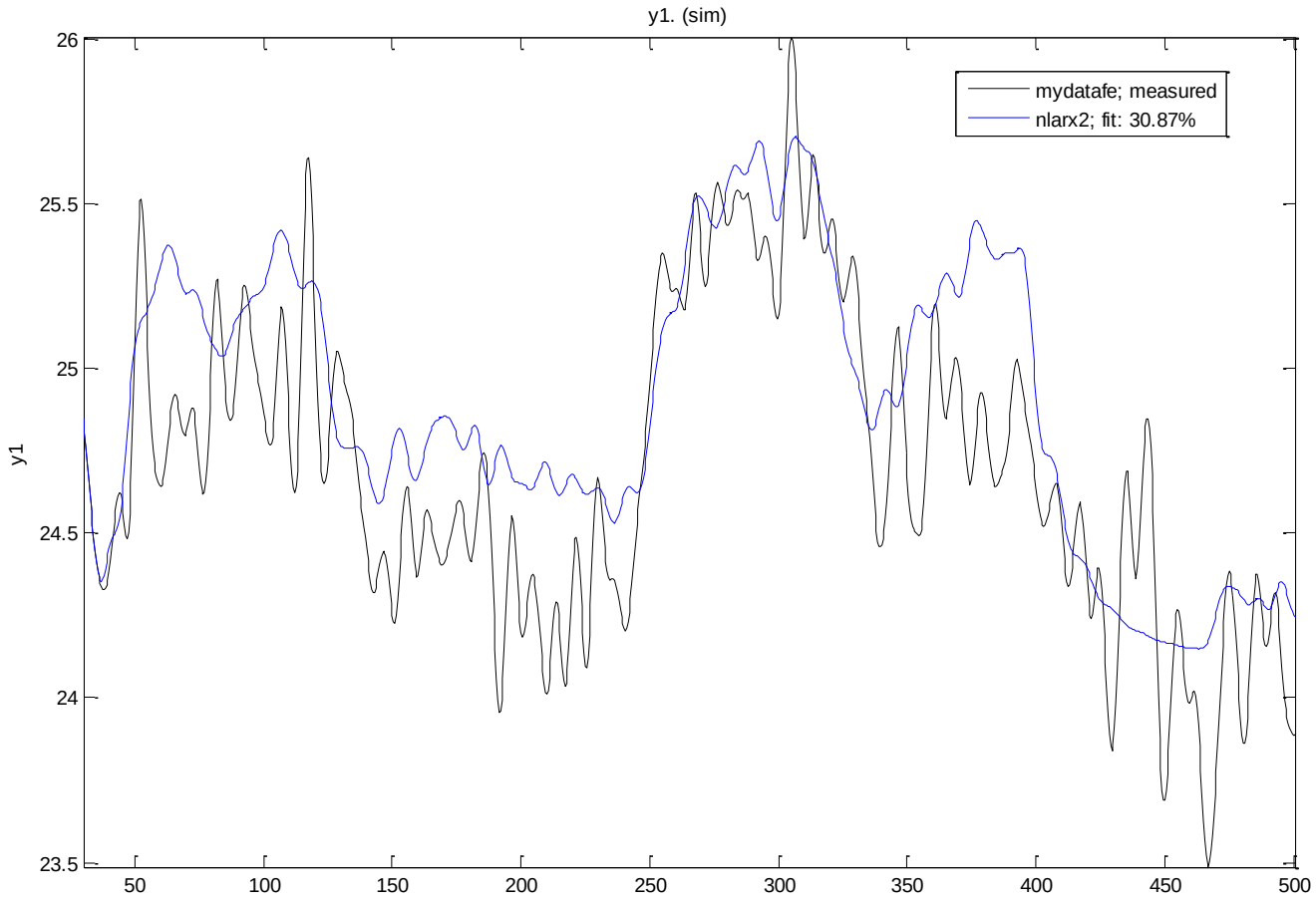


Σχήμα 5.20 – Σύγκριση γραμμικών μοντέλων

Έτσι θα μπορούσαμε να πούμε πως η καλύτερη επιλογή γραμμικού μοντέλου θα ήταν αυτή του Process na=3, delay, zero και Integrator, το οποίο βρίσκεται πιο κοντά στην απόκριση του πραγματικού συστήματος, fit 42.18%.

5.3.6 *Μη γραμμικό μοντέλο, non-linear ARX:*

Για **na = 10** (αριθμός προηγούμενων καταστάσεων εξόδου), **nb = 2** (αριθμός προηγούμενων καταστάσεων εισόδου)[6] το μη γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.21, δίνοντας του το testing set ως είσοδο.



Σχήμα 5.21 – Αποτελέσματα μη γραμμικής μοντελοποίησης ARX na=10,nb=2, no delay

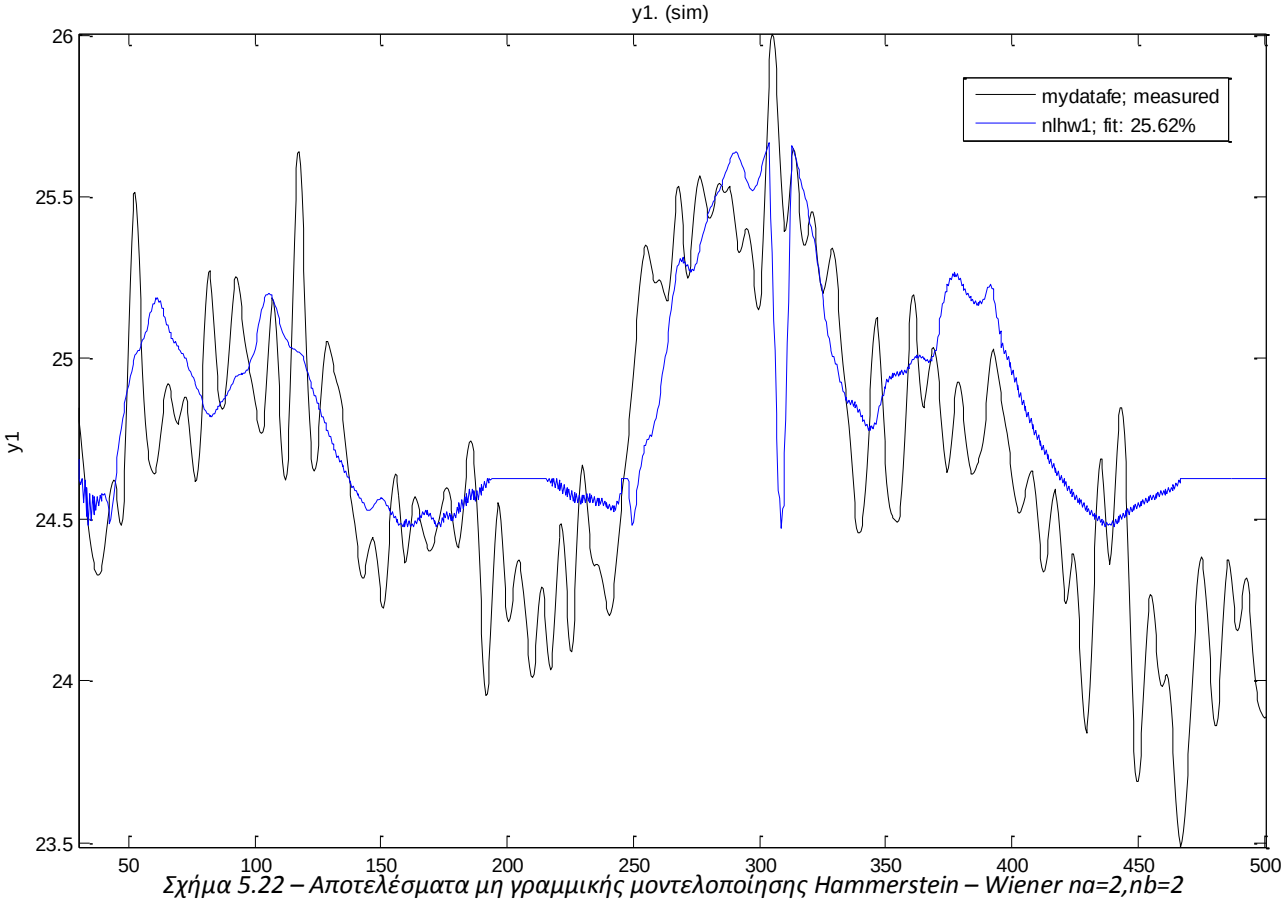
Στο σχήμα 5.21 φαίνεται πως το μη γραμμικό μοντέλο προσεγγίζει όπως και τα γραμμικά μοντέλα (περίπου 30.87% ομοιότητα) το πραγματικό σύστημα.

Θα πρέπει να σημειωθεί πως αλλάζοντας τις παραμέτρους na, nb και nk η εκτίμηση του μοντέλου γινόταν χειρότερη και γι' αυτό έγινε η επιλογή των 2 προηγούμενων παραμέτρων για την είσοδο και για την έξοδο.

[6] Οι παράμετροι na, nb όπως και όλα τα πιθανά μη γραμμικά μοντέλα, αναπτύσσονται στην παράγραφο 2 στην υποενότητα που εξηγούνται τα μοντέλα του Identification toolbox του Matlab

5.3.7 Μη γραμμικό μοντέλο, Hammerstein -Wiener:

Για **na = 2** (αριθμός προηγούμενων καταστάσεων εξόδου), **nb = 2** (αριθμός προηγούμενων καταστάσεων εισόδου) το μη γραμμικό μοντέλο που εκτιμάται έχει ως αποτέλεσμα το γράφημα του σχήματος 5.22, δίνοντας του το testing set ως είσοδο.



Σχήμα 5.22 – Αποτελέσματα μη γραμμικής μοντελοποίησης Hammerstein – Wiener na=2,nb=2

Στο σχήμα 5.22 φαίνεται πως το μη γραμμικό μοντέλο Hammerstein - Wiener προσεγγίζει χειρότερα (25.62% ομοιότητα) το πραγματικό σύστημα από το μη γραμμικό ARX (γράφημα σχήματος 5.21).

5.3.8 *Τελική επιλογή θεωρητικού μοντέλου συστήματος ανεμιστήρα*

Στον πίνακα 5.1 βλέπουμε τα αποτελέσματα της ομοιότητας του κάθε θεωρητικού μοντέλου με το πραγματικό για το testing set δεδομένων.

Πίνακας 5.1 – Πίνακες σύγκρισης μοντέλων συστήματος ψύξης.

Είδος	Μοντέλο	Παράμετροι	Ομοιότητα με το πραγματικό [%]
Γραμμικό	ARX	n_a=2, n_b=10, n_k=1	-4.181
	ARMAX	n_a=2, n_b=10, n_c=1, n_k=1	21.73
	OE	n_a=2, n_b=10, n_k=1	31.64
	Process	n_a=2, zero, delay	35.78
		n_a=3, zero, delay	21.89
		n_a=3, zero, delay, Integrator	42.18 (καλύτερο)
Μη γραμμικό	ARX	n_a=10, n_b=2	30.87
	Ham-Wien	n_a=2, n_b=2	25.62

Έτσι για λόγους προσομοίωσης θα επιλέγαμε το γραμμικό μοντέλο που εξήχθη από την μοντελοποίηση τύπου Process. Έτσι η συνάρτηση μεταφοράς που επιλέγεται είναι αυτή με 3 πόλους, 1 μηδενικό και έναν ολοκληρωτή, και είναι η εξής:

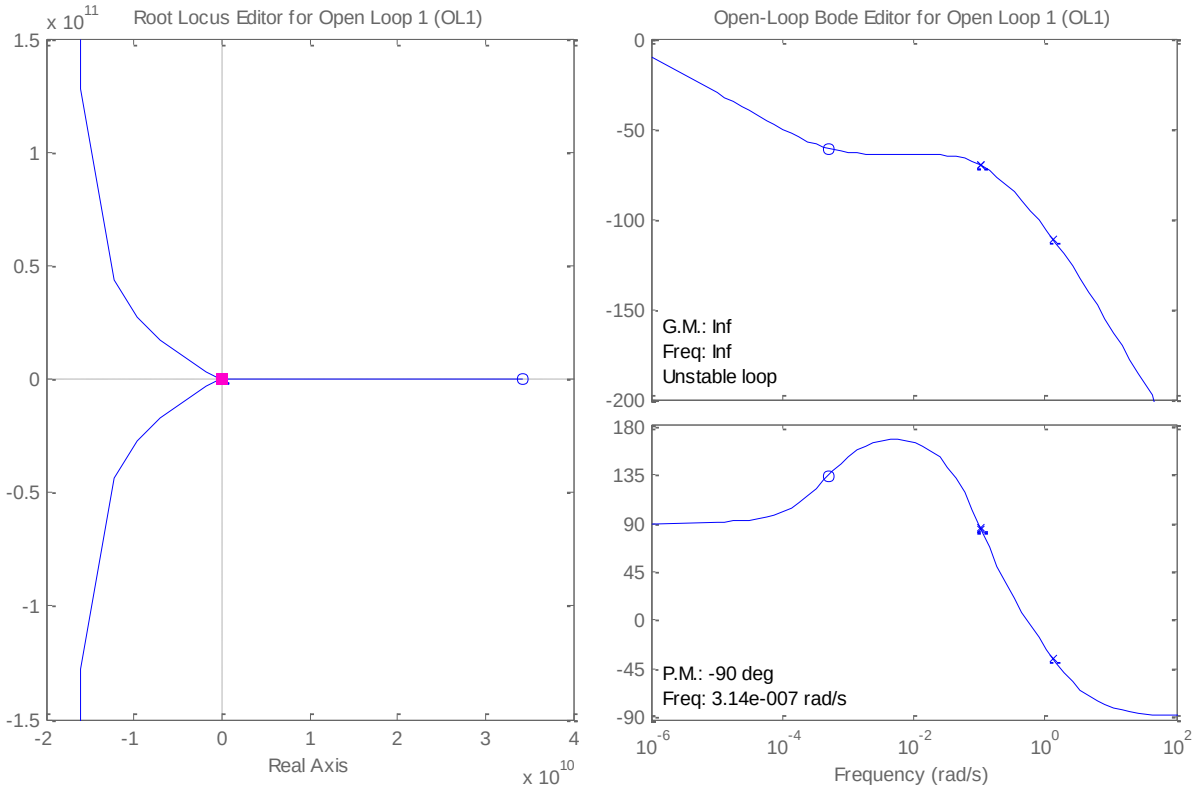
$$G(z) = k_p \frac{1+T_Z s}{s(1+T_{p1}s)(1+T_{p2}s)(1+T_{p3}s)} e^{-T_d s} \qquad (5.1)$$

(Σχέσεις 5.1 – συνάρτηση μεταφοράς μοντέλου P3 +D +I +Z)

Όπου:

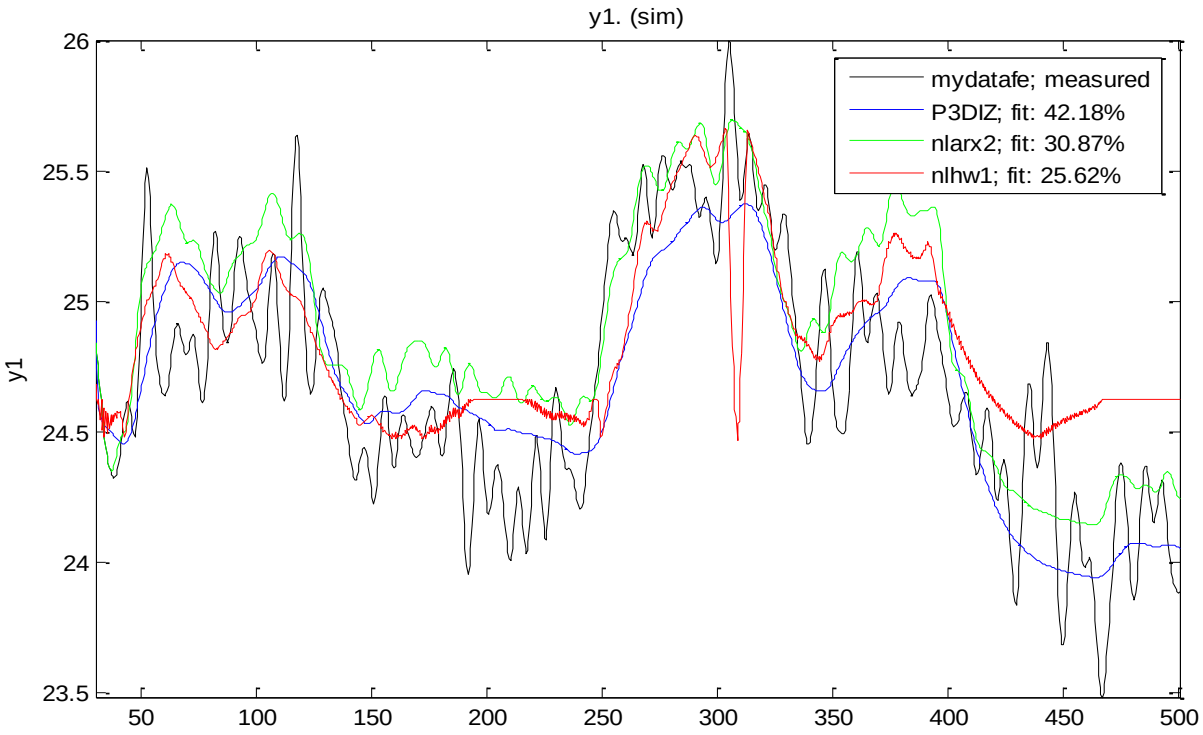
k_p = -3.1361e-007
T_{p1} = 9.426
T_{p2} = 9.2765
T_{p3} = 0.75093
T_d = 0
T_z = 1985.2

Στο σχήμα 5.23 φαίνεται ο γεωμετρικός τόπος ριζών του ανοιχτού βρόχου του συστήματος του μοντέλου, όπως και το διάγραμμα Bode.



α. β.

Σχήμα 5.23 – α. Γεωμετρικός τόπος ριζών μοντέλου, β. Διάγραμμα Bode (P3 +D +I +Z)



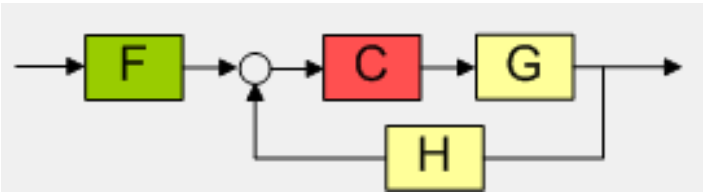
Σχήμα 5.24 – Σύγκριση καλύτερου γραμμικού με μη-γραμμικά μοντέλα

Υπολογισμός PID μέσω του SISOtool

Στην συνέχεια γίνεται ο αυτόματος εντοπισμός κατάλληλου ελεγκτή PID, σύμφωνα με το γραμμικό μοντέλο που εξήχθη από την προηγούμενη ενότητα, που να ελέγχει το σύστημα όσο το πιθανόν βέλτιστα έτσι ώστε να υλοποιείται το διάγραμμα βαθμίδων που φαίνεται στο σχήμα 3.2.

Στην πραγματικότητα προσομοιώνουμε το σύστημα ψύξης με το μοντέλο από την προηγούμενη ενότητα για να υλοποιήσουμε auto tuning του ελεγκτή PID που θα προηγείται από αυτό. Το auto tuning υλοποιήθηκε και σε αυτή την περίπτωση μέσω του εργαλείου Sisotool του Matlab.

Αρχικά εισήχθη στο sisotool η συνάρτηση μεταφοράς του συστήματος στην θέση του συστήματος G, στο σχήμα 5.25.



Σχήμα 5.25 – Εισαγωγή του μοντέλου στο sisotool

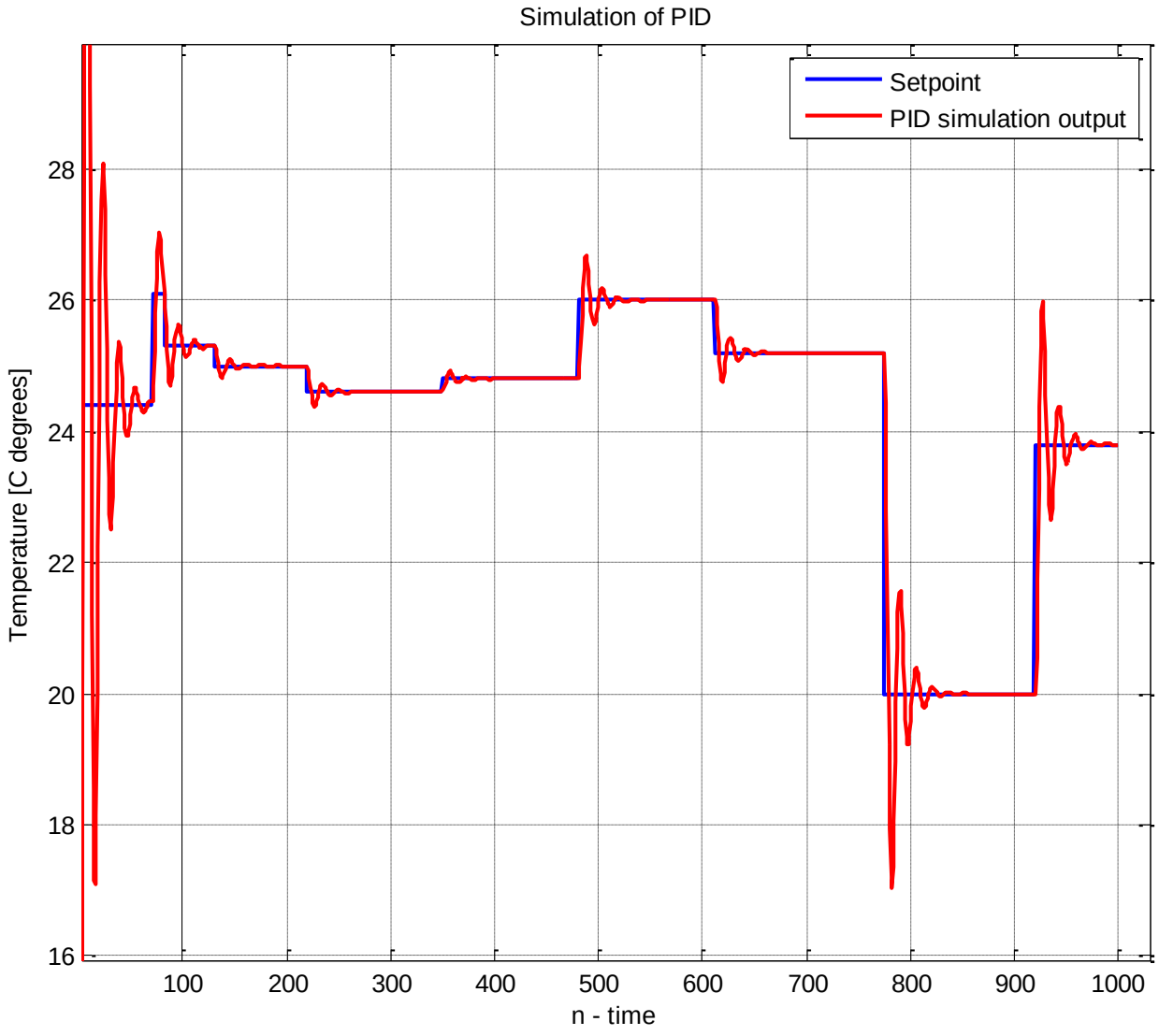
Ο ελεγκτής PID είναι στην πράξη το υποσύστημα C για το οποίο θα γίνει η αναζήτηση των καλύτερων συντελεστών Kp, Ki και Kd. Η μέθοδος του αυτόματου tuning που επιλέχθηκε είναι μέσω απόκρισης συχνότητας – Ziegler – Nichols.

Έτσι προκύπτουν τα εξής αποτελέσματα:

Kp = 27900, Ki = 4830 Kd = 40400

MAE (Mean Absolute Error) = 0.3193 ^{o}C
Ποσοστό υπερύψωσης: Mp = 58.32 %

Στο σχήμα 5.26 βλέπουμε την απόκριση του ελεγκτή PID και του μοντέλου που υπολογίστηκαν θεωρητικά.

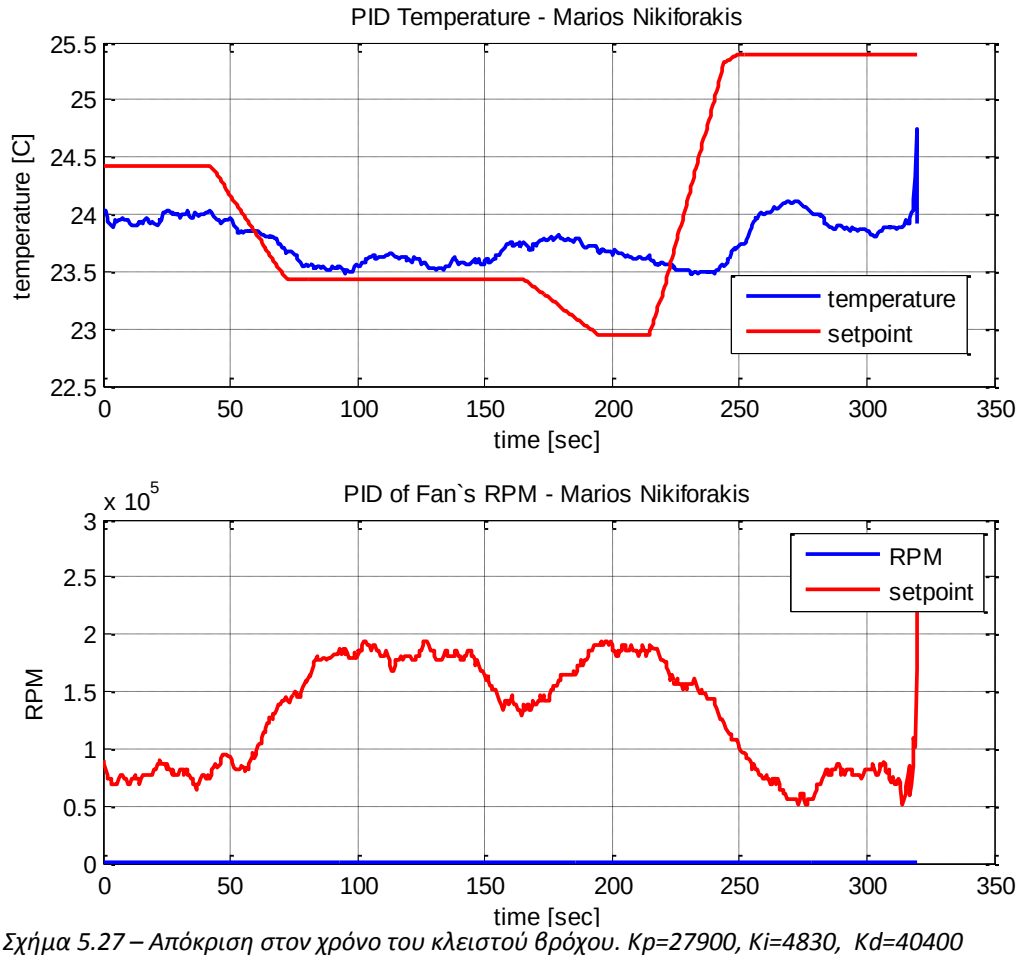


Σχήμα 5.26 – Απόκριση στον χρόνο του κλειστού βρόχου. Autotuning -> Freq. Ziegler - Nichols

5.4 Πειραματικά και θεωρητικά αποτελέσματα

Οι συντελεστές Kp, Ki και Kd που προέκυψαν από το autotuning μέσω του θεωρητικού μοντέλου ήταν kp=27900, ki=4830 και kd=40400. Οι παράμετροι αυτοί διαφέρουν αρκετά από αυτές που εκτιμήσαμε με το manual tuning παραπάνω.

Εισάγοντας τους συντελεστές που εξήχθησαν από το autotuning του θεωρητικού μοντέλου στην πειραματική κατασκευή, έχουμε την απόκριση που φαίνεται στο σχήμα 5.27. Θα πρέπει να σημειωθεί πως σε αυτό το σημείο γίνεται η μετατροπή των συντελεστών κατάλληλα λόγο της δειγματοληψίας του PID του Arduino, το Matlab έχει εξάγει αναλογικούς συντελεστές. Η δειγματοληψία PID του Arduino είναι 0.2s.



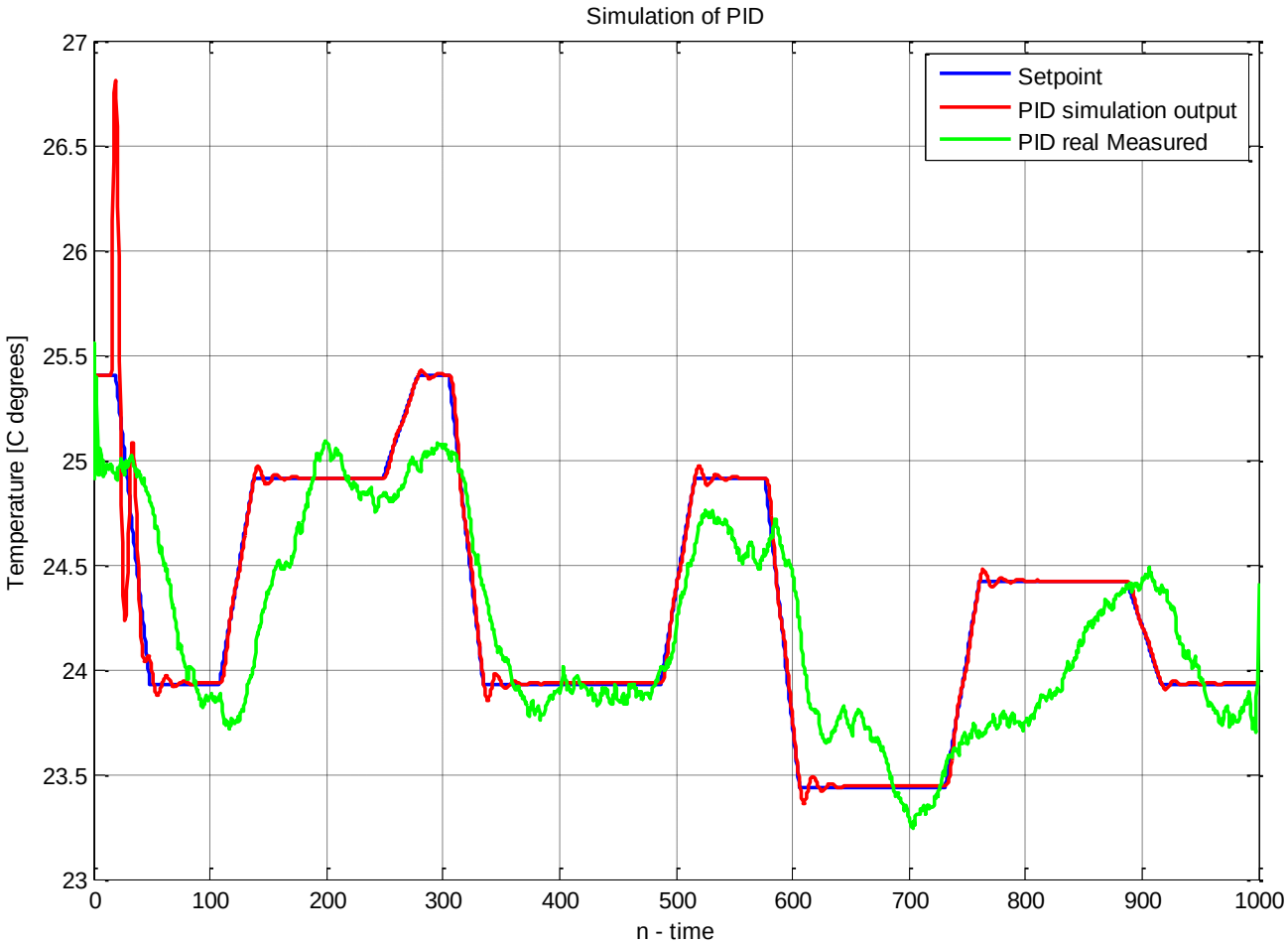
Σχήμα 5.27 – Απόκριση στον χρόνο του κλειστού βρόχου. Kp=27900, Ki=4830, Kd=40400

Στον Πίνακα 5.2 φαίνονται οι συντελεστές που υπολογίστηκαν (μέσω του θεωρητικού μοντέλου και του auto tuning) σε σχέση με αυτούς που επιλέγηκαν μέσω του manual tuning και της πειραματικής κατασκευής.

Πίνακας 5.2– Σύγκριση Auto tuning και Manual Tuning

Tuning	Σύστημα	Kp	Ki * Ts	Kd / Ts
Auto	Θεωρητικό μοντέλο – SISOtool	**27900**	4830	40400
Manual	Ανεμιστήρας - Arduino PID	**800**	50	70

Σε αυτή την φάση έγινε ένα γενικό πείραμα μιας ολοκληρωμένης διαδικασίας ελέγχου στροφών ανεμιστήρα αρχικά μέσω της πειραματικής κατασκευής και των συντελεστών kp, ki και kd που εξήχθησαν από το manual tuning και στην συνέχεια για τις ίδιες εισόδους έγινε η προσομοίωση με το θεωρητικό μοντέλο και τους συντελεστές που προέκυψαν από το sisotool του Matlab. Στο σχήμα 5.28 φαίνεται αυτή η σύγκριση.



Σχήμα 5.28 – Σύγκριση μαθηματικού μοντέλου και autotuning με πραγματικό ελεγκτή και manual tuning, σε βηματικές εισόδους.

5.5 Συμπεράσματα σύγκρισης

Για την εκτίμηση του μοντέλου συμπεραίνουμε πως το άγνωστο σύστημα ψύξης μοντελοποιείται καλύτερα μέσω γραμμικού διαδικαστικού μοντέλου, όπως φαίνεται στον πίνακα 5.1.

Ως μοντέλο του συστήματος τελικά επιλέγεται το P3 +D +I +Z (3 πόλοι, με μηδενικό, ολοκληρωτή και καθυστέρηση) για την γενικότερη ανάλυση, όσο και για το auto tuning.

Η μοντελοποίηση του συστήματος δεν είναι αρκετά ικανοποιητική, αφού τα ποσοστά ομοιότητας που εξήχθησαν ήταν σχεδόν από 40%. Στο σχήμα 5.24 παρατηρούμε την σύγκριση των μοντέλων. Φαίνεται πως το P3 +D +I +Z γραμμικό μοντέλο συγκλίνει περισσότερο προς το πραγματικό σύστημα.

Το αρκετά μεγάλο σφάλμα στην μοντελοποίηση οφείλεται στους εξής παράγοντες:

- Μικρό εύρος λειτουργίας του ανεμιστήρα (από 800 ως 1800 RPM περίπου). Το οποίο είχε σαν συνέπεια τις πολύ μικρές αλλαγές στην θερμοκρασία. Το οποίο είχε σαν συνέπεια την δύσκολη μέτρηση από τον αισθητήρα λόγο μικρών αλλαγών. Έτσι ο θόρυβος που ήταν περίπου +/- 1 LSB του Analog to Digital ήταν κοντά στα ποσοστά μετρήσεων. Έτσι έγινε η εισαγωγή φίλτρων που άλλαζαν κατά πολύ την συμπεριφορά του συστήματος.
- Υπήρχαν έντονες μη γραμμικότητες αφού δεν γινόταν να οριστούν τα όρια για τις εσωτερικές μεταβλητές ελέγχου (RPM setpoint) εντός των εργαλείων του Matlab. Έτσι πολλές φορές το μοντέλο θεωρούσε σωστό το να βγαίνει εκτός ορίων (800 – 1800 RPM).
- Η αλλαγή της θερμοκρασίας γινόταν μη γραμμικά κατά την αλλαγή της ταχύτητας του ανεμιστήρα. Η θερμοκρασία είχε ανώτατο και κατώτατο όριο (για την συγκεκριμένη εποχή και με λυχνία ενεργοποιημένη: από 23.5 ως 25.5 περίπου).
- Το υπο-μοντελοποίηση σύστημα περιέχει ήδη έναν ελεγκτή PID και είναι αυξημένη η πολυπλοκότητα του (PID ελέγχου στροφών από προηγούμενη παράγραφο)

Σε όλα τα παραπάνω μοντέλα που εξήχθησαν υπήρχε ευστάθεια αφού μέσω των γραφημάτων των πόλων και μηδενικών τους μπορούμε να δούμε πως τόσο οι πόλοι, όσο και τα μηδενικά των συστημάτων βρίσκονται εντός του μοναδιαίου κύκλου.

Από τα γραφήματα BODE του καλύτερου συστήματος (σχήμα 5.23) μπορούμε να συμπεράνουμε πως το σύστημα μοιάζει περισσότερο σε χαμηλοπερατό φίλτρο, αφού για χαμηλές τιμές συχνότητας έχει μεγαλύτερο μέτρο.

Από την σύγκριση του autotuning του θεωρητικού μοντέλου με το manual tuning του πραγματικού συστήματος συμπεραίνουμε πως οι τιμές που εξήχθησαν από το auto tuning δεν ήταν αρκετά όμοιες με αυτές από το manual tuning το οποίο οφείλεται στα εξής γεγονότα:

- Το θεωρητικό μοντέλο το οποίο δε περιείχε τα όρια της εσωτερικής μεταβλητής ελέγχου(800 – 1800 RPM περίπου) εξήγαγε τον συγκεκριμένο PID με πολύ μεγάλα κέρδη, ο οποίος έκανε το σύστημα κλειστού βρόχου πολύ γρήγορο. Παρόλα αυτά στο πραγματικό σύστημα επειδή υπάρχουν τα συγκεκριμένα όρια, όταν εισήχθησαν οι συγκεκριμένες τιμές η εσωτερική μεταβλητή των επιθυμητών στροφών του ανεμιστήρα έβγαινε αρκετά εκτός ορίων όπως φαίνεται στο σχήμα 5.4 (κάτω), όπου η τιμή των επιθυμητών στροφών έχει φτάσει τιμές τάξης μεγέθους 105.
- Η μοντελοποίηση του συστήματος ήταν γραμμική ενώ το πραγματικό σύστημα περιέχει έντονες μη γραμμικότητες, τόσο λόγω των ορίων στην έξοδο, όσο και λόγο των ορίων της εσωτερικής μεταβλητής ελέγχου (RPM setpoint).
- Πολύ μεγάλος θόρυβος στον αισθητήρα σε σχέση με τις μεταβολές που πρέπει να ελέγξουμε.

Παρόλα αυτά μπορούμε να συμπεραίνουμε πως το μοντέλο που επιλέγθηκε ήταν ικανό να μας δώσει μια αρκετά καλή εκτίμηση για την γενική και ποιοτική μελέτη του συστήματος.

Εισάγοντας τις παραμέτρους που υπολογίστηκαν από το auto tuning στο πραγματικό σύστημα η απόκριση δεν ήταν ικανοποιητική (σχήμα 5.27). Αυτό για τους λόγους που αναφέρονται παραπάνω.

Στο σχήμα 5.28 βλέπουμε την σύγκριση του βέλτιστου θεωρητικού ελεγκτή PID με τον βέλτιστο πραγματικό ελεγκτή. Παρατηρούμε πως υπάρχει διαφορά ως προς την

απόκριση αφού ο έλεγχος μέσω του auto tuning και του θεωρητικού μοντέλου είναι πολύ καλύτερος από τον πειραματικό έλεγχο. Και στις δύο περιπτώσεις παρατηρείται υπερύψωση αλλά και ταλάντωση η οποία μπορεί να οφείλεται στον κβαντισμό λόγο των πράξεων εντός του Arduino και στο σφάλμα μετρήσεως αφού όπως αναφέρεται και παραπάνω υπάρχει μεγάλος θόρυβος σε σχέση με τις μεταβολές που πρόκειται να ελεγχθούν.

Οι τελικές συναρτήσεις που προκύπτουν για το PID είναι οι εξής:

Για το Auto tuning:

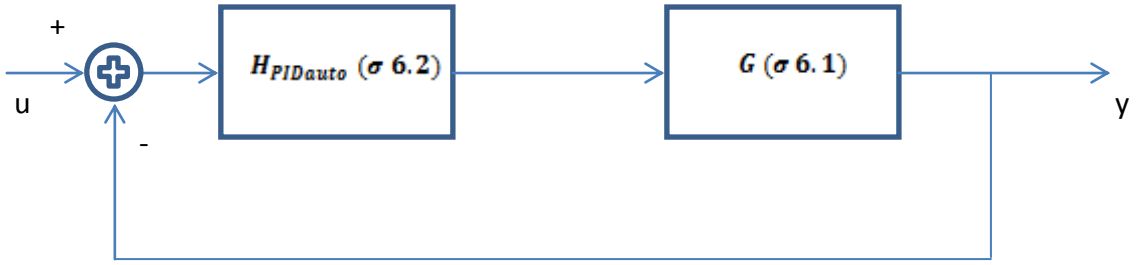
$$H_{PIDauto} = 1000(27.9 + 4.83\frac{1}{s} + 40.4 \cdot s) \quad (5.2)$$

Για το Manual tuning:

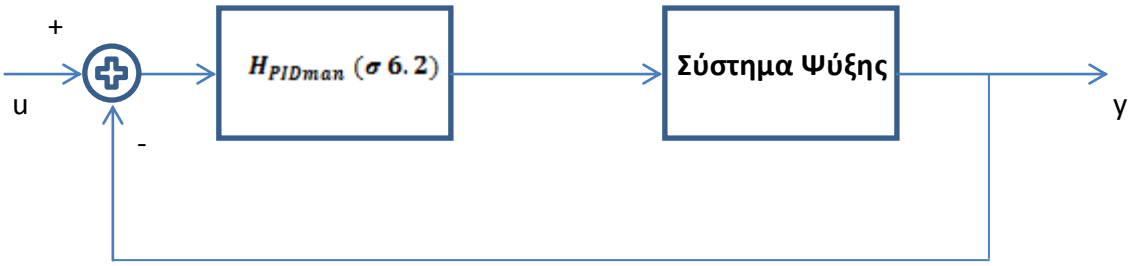
$$H_{PIDauto} = 1000(0.8 + 0.05\frac{1}{s} + 0.07 \cdot s) \quad (5.3)$$

(Σχέσεις 5.2 – 5.3: συναρτήσεις μεταφοράς των ελεγκτών PID)

Άρα τα τελικά βέλτιστα για κάθε περίπτωση διαγράμματα βαθμίδων είναι αυτά που φαίνονται στα σχήματα 5.29 και 5.30.



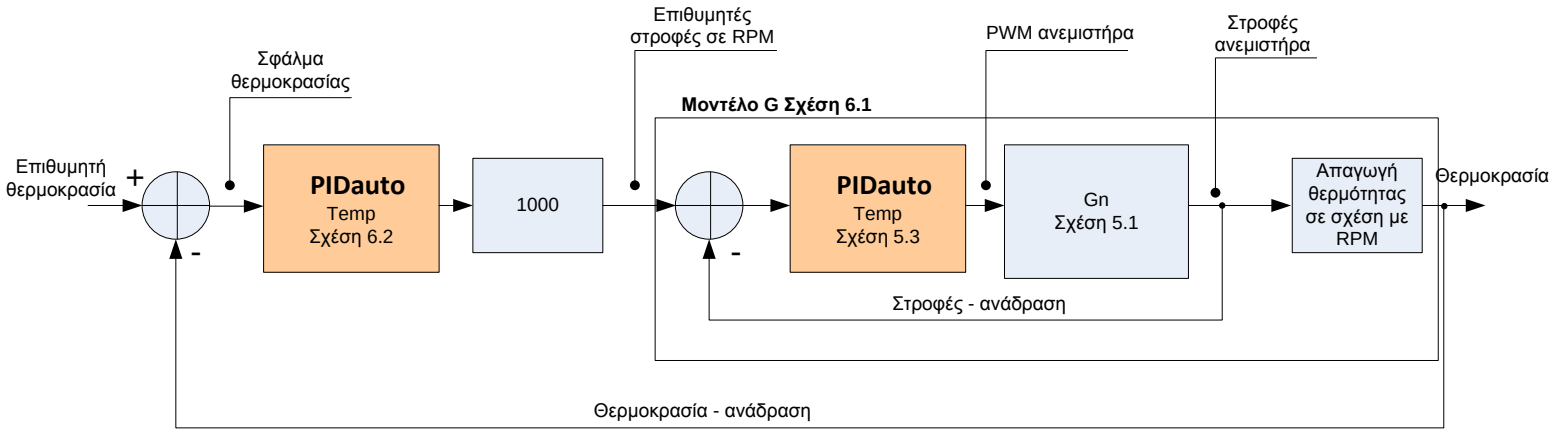
Σχήμα 5.29 – Διάγραμμα βαθμίδων βέλτιστης μοντελοποίησης συστήματος ελέγχου θερμοκρασίας.



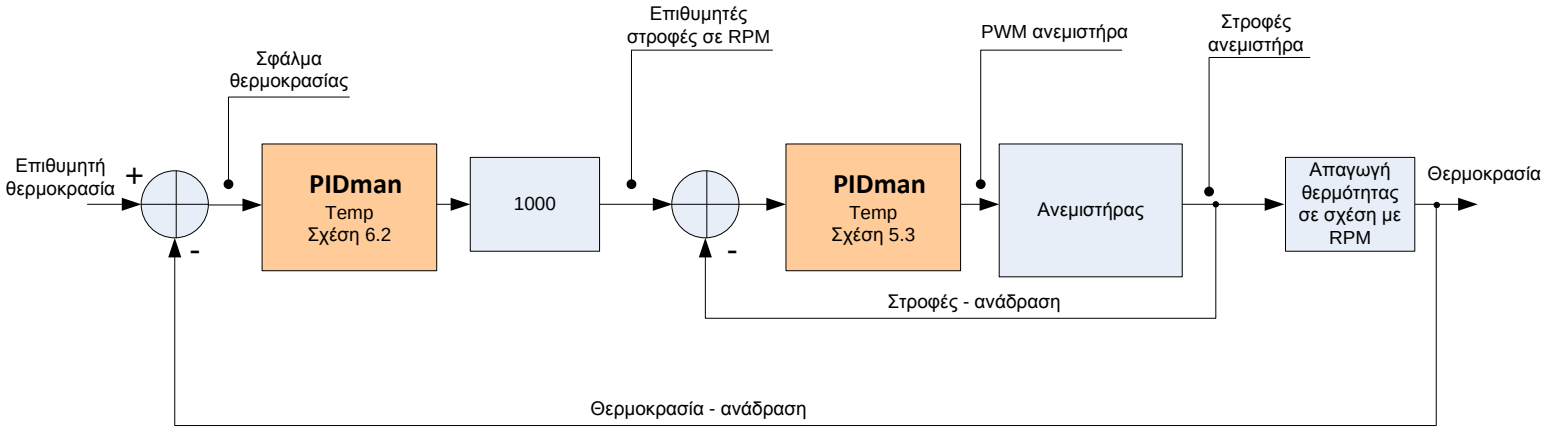
Σχήμα 5.30– Συνολικό διάγραμμα βαθμίδων βέλτιστου πραγματικού συστήματος ελέγχου θερμοκρασίας.

Αν εισάγουμε και τα εσωτερικά στοιχεία που αναλύονται στο προηγούμενο κεφάλαιο τότε τα τελικά συστήματα είναι αυτά που φαίνονται στα σχήματα 5.31 και 5.32 για την μοντελοποίηση και το πραγματικό σύστημα αντίστοιχα:

Θα πρέπει να σημειωθεί πως οι τιμές της επιθυμητής τιμής θερμοκρασίας μπορεί να πάρει μόνο συγκεκριμένες τιμές λόγο του κβαντισμού του Analog to Digital του αισθητήρα θερμοκρασίας.



Σχήμα 5.31– Συνολικό διάγραμμα βαθμίδων βέλτιστης μοντελοποίησης συστήματος ελέγχου θερμοκρασίας.



Σχήμα 5.32– Συνολικό διάγραμμα βαθμίδων βέλτιστου πραγματικού συστήματος ελέγχου θερμοκρασίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] http://www.arduino.cc/

[2] http://playground.arduino.cc/Code/PIDLibrary

[3] http://fritzing.org/

[4] Texas Instruments: LM35 Datasheet – Dallas / Texas, 2011

[5] Texas Instruments: ULN2803 – Dallas / Texas, 2012

[6] Lennart Ljung : "System Identification Toolbox, For Use with Matlab"

[7] http://nonscholaesedvitaediscimus.wordpress.com/2007/12/05/estimating-the-arx-model/

[8] Martin Dvořáček: Wind tunnel identification and control via OE model and lQ controller, Doctoral Degree Programme (2), FEEC BUT, 2010

[9] Richard C. Dorf, Robert H. Bishop: Σύγχρονα Συστήματα Αυτομάτου Ελέγχου, Τζίολα, 9η Εκδοση.

[10] Πετρίδης Βασίλειος: Συστήματα Αυτομάτου Ελέγχου Τόμος Α, Θεσσαλονίκη, Ζήτη 2001

[11] Δημήτριος Βαρσάμης: Σημειώσεις GUI για το εργαστήριο του μαθήματος Γραμμικός Προγραμματισμός & Βελτιστοποίηση. η 2001

ΠΑΡΑΡΤΗΜΑ Α

Α.1 Κώδικας Arduino

```
//Libraries included
#include <PID_v1.h>

//PINS !!!
int fanPulse = 2;
int lampPin = 3;
int fanPWMpin = 5;
int sensorIn = 0;

//INIT MODE
//mode 0: no Temperature PID, only RPM PDI
//mode 1: Temperature PID + RPM PID
int mode=1;

//Variables for RPM feedback
unsigned long pulseDurationLow;
unsigned long pulseDurationHigh;

//Motors's RPM
int RPM = 0;

//Temperature
int Temperature = 0;

//Limits of Fan's RPM
int RPMlimitHigh = 2000;
int RPMlimitLow = 100;

//PID of FAN's RPM parameters
double SetpointRPM = 1000;
double InputRPM, OutputRPM;

//Kp, Ki, Kd of FAN's RPM PID
double KpRPM = 2;
double KiRPM = 5;
double KdRPM = 1;

//set PID DIRECT ... UP PWM means UP RPM
PID RPMpid(&InputRPM, &OutputRPM, &SetpointRPM, KpRPM, KiRPM,
KdRPM, DIRECT);


//temperature PID parameters
double SetpointT = 50;
double InputT, OutputT;

//Kp, Ki, Kd of temperature's PID
double KC = 10;
double KpT = 2 ;
```

```
double KiT = 5 ;
double KdT = 1 ;

//set PID REVERSE ... UP RPM means DOWN temp
PID Tpid(&InputT, &OutputT, &SetpointT, KpT, KiT, KdT, REVERSE);

//Inititations
void setup(){

  //Begin serial in 115200
  Serial.begin(115200);

  //Pulse in at pin 2
  pinMode(fanPulse, INPUT);

  //Lamp PWM at pin 3
  pinMode(lampPin, OUTPUT);

  //FAN PWM at pin 5
  pinMode(fanPWMpin, OUTPUT);

  //enable pull-up resist
  digitalWrite(fanPulse,HIGH);

  //turn the PID of FAN's RPM on
  RPMpid.SetMode(AUTOMATIC);

  //turn the PID of temperature on
  Tpid.SetMode(AUTOMATIC);
}

//Returns the value of RPM reads pulse in
//Feedback Function !!!!!!!!! (of RPM control)
int readRPM() {

  //Duration of Pulse is (low pulse + High pulse) * 2
  pulseDurationLow = pulseIn(fanPulse, LOW);
  pulseDurationHigh = pulseIn(fanPulse, HIGH);
  double frequency =
1000000/(pulseDurationLow+pulseDurationHigh);
  return (int)frequency*60/2;

  //Serial.print("pulse duration:");
  //Serial.println((pulseDurationLow+pulseDurationHigh)/2);

  //Serial.print("time for full rev. (microsec.):");
  //Serial.println(pulseDurationLow+pulseDurationHigh);
  //Serial.print("freq. (Hz):");
  //Serial.println(frequency/2);
  //Serial.print("RPM:");
  //Serial.println(frequency*60/2);

}

//Serves the client demands
```

```
//serial communication is defined at setup function
void serialServer(){

  /* variables declaration and initialization                */
  int val;
  int val2;
  int temp;
  int k;
  double tempD;

  if (Serial.available() >0) {

    //whatever is available from the serial is read here
    val = Serial.read();

    //This part basically implements a state machine
    switch (val) {

      //case 1 means: READ RPM of FAN
      //sends the RPM of FAN
      case 49:
        Serial.println(RPM);
      break;

      //case 2 means: SET PWM of lamp
      //3 digits after number 2
      //2###
      case 50:
         val2 = Serial.read();
         temp=(val2-48)*100;
         val2 = Serial.read();
         temp+=(val2-48)*10;
         val2 = Serial.read();
         temp+=(val2-48);
         if(temp<256 && temp>=0){
           analogWrite(lampPin, temp);
         }
      break;

      //case 3 means: SET setpoint of RPM
      //4 digits after number 3
      //3####
      case 51:
         val2 = Serial.read();
         temp=(val2-48)*1000;
         val2 = Serial.read();
         temp+=(val2-48)*100;
         val2 = Serial.read();
         temp+=(val2-48)*10;
         val2 = Serial.read();
         temp+=(val2-48);
         if(temp<RPMlimitHigh && temp>=RPMlimitLow){
           SetpointRPM = temp;
         }
      break;
```

```
//case 4 means: SET K& of FAN's PID
//first digit after 4 means 1=p , 2=i , 3=d
//3 others for value
//4&###
case 52:
   k = Serial.read()-48;

   val2 = Serial.read();
   temp=(val2-48)*1000;
   val2 = Serial.read();
   temp+=(val2-48)*100;
   val2 = Serial.read();
   temp+=(val2-48)*10;
   val2 = Serial.read();
   temp+=(val2-48);

   tempD = ((double)temp)/100.0;

   if(k == 1){
     KpRPM = tempD;
     Serial.println(KpRPM);
   }else if(k == 2){
     KiRPM = tempD;
    Serial.println(KiRPM);
   }else if(k == 3){
     KdRPM = tempD;
     Serial.println(KdRPM);
   }

   RPMpid.SetTunings(KpRPM, KiRPM, KdRPM);
break;

//case 5 means: READ Sensor data 10-bit
//sends back Sensor data
case 53:
  Serial.println(analogRead(0));
break;

//case 6 means: Mode
//mode 0: no Temperature PID, only RPM PDI
//mode 1: Temperature PID + RPM PID
//else Nothing
//6& (& = 0 or 1)
case 54:
  temp=Serial.read()-48;
  if(temp==0 || temp==1){
    mode=temp;
  }
break;

//case 7 means: SET setpoint of temperature's PID
//4 digits after number 7
//7####
case 55:
```

```
   val2 = Serial.read();
   temp=(val2-48)*1000;
   val2 = Serial.read();
   temp+=(val2-48)*100;
   val2 = Serial.read();
   temp+=(val2-48)*10;
   val2 = Serial.read();
   temp+=(val2-48);

   SetpointT = temp;
   Serial.println(SetpointT);
break;

//case 8 means: SET K& of temperature's PID
//first digit after 4 means 1=p , 2=i , 3=d
//3 others for value
//8&###
case 56:
   k = Serial.read()-48;

   val2 = Serial.read();
   temp=(val2-48)*1000;
   val2 = Serial.read();
   temp+=(val2-48)*100;
   val2 = Serial.read();
   temp+=(val2-48)*10;
   val2 = Serial.read();
   temp+=(val2-48);

   tempD = ((double)temp)/100.0;

   if(k == 1){
     KpT = tempD;
     Serial.println(KpT);
   }else if(k == 2){
     KiT = tempD;
    Serial.println(KiT);
   }else if(k == 3){
     KdT = tempD;
     Serial.println(KdT);
   }

   Tpid.SetTunings(KpT, KiT, KdT);
break;

//case 9 means: READ PWM duty cycle
//sends PWM duty cycle
case 57:
  Serial.println(OutputRPM);
break;

//case 0 means: READ Setpoint RPM
//Setpoint of PID RPM
case 48:
  Serial.println(SetpointRPM);
```

```
      break;


    }
  }


}

//Main Loop !!
void loop(){

  Temperature=analogRead(0);
  serialServer();
  RPM=readRPM();

  //-----RPM PID CODE-----
  InputRPM = RPM;
  RPMpid.Compute();
  analogWrite(fanPWMpin,OutputRPM);
  //----------------------

  //////IF MODE 1 IS ENABLED////
  if(mode == 1){
    //-----temperature PID CODE-----
    InputT = Temperature;
    Tpid.Compute();

    //KC is a constant gain ...
    //cause Temperatures have much
    //difference as a zise from RPMs
    SetpointRPM=OutputT * KC;
    //----------------------
  }

}
```

ΠΑΡΑΡΤΗΜΑ Β

Β.1 Matlab class για επικοινωνία (Client)

```
classdef PIDarduino < handle

    % This class defines an PID arduino class


    properties (SetAccess=private,GetAccess=private)
        ser;
        Vref=5;  %Volts
        Nquant=10;%bits of analog to digital
        thermSlope=0.01; %Volts/Celciou
        factor;

    end

    methods

        % constructor, connects PID arduino to desired com port
        function a=PIDarduino(comPort)
            a.ser = serial(comPort,'BaudRate',115200);
            a.factor=(a.Vref/(2^a.Nquant))/a.thermSlope;
            fopen(a.ser);
            %wait
            pause(0.0014);

        end

        % distructor, deletes the object
        function delete(a)

            fclose(a.ser);

            % if it's an object delete it
            if isobject(a.ser),
                delete(a.ser);
            end


        end

        % gets the RPM value
        function val = getRPM(a)

            %demand "1" is read RPM value
            fwrite(a.ser,[49],'uchar');

            %wait
            pause(0.0014);

            %read data from serial buffer
```

```
            val = fscanf(a.ser,'%d');

            %wait
            pause(0.0014);

        end

        % set PWM of Lamp
        function val = setLamp(a , val)

            val=floor(val*255/100);

            if(val>=0 && val<=256)
                %demand "2" is set PWM for Lamp
                fwrite(a.ser,[50
48+splitDigits(val,3)],'uchar');

                %wait
                pause(0.0014);

            else
                error('Value not between 0 and 100 ...');
            end

        end

        % set setPoint of RPM (when in that mode)
        function val = setSpointRPM(a , val)



            if(val>=0 && val<=2000)
                %demand "3" is set RPM setPoint
                fwrite(a.ser,[51
48+splitDigits(val,4)],'uchar');

                %wait
                pause(0.0014);

            else
                error('Value not between 0 and 2000 ...');
            end

        end

        % get setPoint of RPM
        function val = getSpointRPM(a)



             %demand "0" is read RPM value
            fwrite(a.ser,[48],'uchar');

            %wait
            pause(0.0014);
```

```
            %read data from serial buffer
            val = fscanf(a.ser,'%d');

            %wait
            pause(0.0014);

        end

        % change mode of PID arduino
        function val = changeMode(a , mode)


            if(mode == 'T')
                %mode Temperature
                 %demand "6" is mode
                fwrite(a.ser,[48+6 48+1],'uchar');
            elseif(mode == 'R')
                 %mode RPM
                 %demand "6" is mode
                fwrite(a.ser,[48+6 48],'uchar');
            else
                error('Not a valid mode ... choose T or R');
            end
            %wait
            pause(0.0014);



        end



        % set Ki of temperatures PID
        function val = setKTemp(a , i , val)



            fwrite(a.ser,[48+8 48+i
48+splitDigits(floor(val*100),4)],'uchar');

            %wait
            pause(0.0014);

            %read data from serial buffer
            val = fscanf(a.ser);

            %wait
            pause(0.0014);

        end

        % set Ki of RPMs PID
        function val = setKRpm(a , i , val)
```

```
            fwrite(a.ser,[48+4 48+i
48+splitDigits(floor(val*100),4)],'uchar');

            %wait
            pause(0.0014);

            %read data from serial buffer
            val = fscanf(a.ser);

            %wait
            pause(0.0014);

        end

        % gets Temperature
        function val = getTemp(a)


            fwrite(a.ser,[48+5],'uchar');

            %wait
            pause(0.0014);

            %read data from serial buffer
            temp = fscanf(a.ser,'%d');
            val=temp*a.factor;

            %wait
            pause(0.0014);

        end

        % set setpoint of Temperature (if in that mode)
        function val = setSpointTemp(a , temper)

            temp=floor(temper/a.factor);
            fwrite(a.ser,[48+7 48+splitDigits(temp,4)],'uchar');

            %wait
            pause(0.0014);

            %read data from serial buffer
            val = fscanf(a.ser,'%d');

            %wait
            pause(0.0014);


        end

          % set setpoint of Temperature (if in that mode)
        function val = getPWM (a)


            fwrite(a.ser,[48+9],'uchar');
```

```
            %wait
            pause(0.0014);

            %read data from serial buffer
            val = fscanf(a.ser,'%d');

            val=val*100/255;

            %wait
            pause(0.0014);


        end

    end


end % class def
```

ΠΑΡΑΡΤΗΜΑ Γ

Γ.1 Κώδικας του Matlab GUI

```
function varargout = nikiforakisPID(varargin)
% NIKIFORAKISPID M-file for nikiforakisPID.fig
% NIKIFORAKISPID, by itself, creates a new NIKIFORAKISPID or %
% raises the existing singleton*.
%
% H = NIKIFORAKISPID returns the handle to a new NIKIFORAKISPID
% or the handle to the existing singleton*.
%
% NIKIFORAKISPID('CALLBACK',hObject,eventData,handles,...) calls
% the local function named CALLBACK in NIKIFORAKISPID.M with the
% given input arguments.
%
% NIKIFORAKISPID('Property','Value',...) creates a new
% NIKIFORAKISPID or raises the existing singleton*.  Starting
% from the left, property value pairs are applied to the GUI
% before nikiforakisPID_OpeningFcn gets called.  An unrecognized
% property name or invalid value makes property application
% stop.  All inputs are passed to nikiforakisPID_OpeningFcn via
% varargin.
%
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
% only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% nikiforakisPID

% Last Modified by GUIDE v2.5 25-Nov-2012 12:59:07

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @nikiforakisPID_OpeningFcn,
...
                   'gui_OutputFcn',  @nikiforakisPID_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before nikiforakisPID is made visible.
function nikiforakisPID_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to nikiforakisPID (see
VARARGIN)


    % Choose default command line output for nikiforakisPID
    handles.output = hObject;


    % Update handles structure
    guidata(hObject, handles);


%     Vref=5;  %Volts
%     Nquant=10;%bits of analog to digital
%     thermSlope=0.01; %Volts/Celciou
%     factor=(Vref/(2^Nquant))/thermSlope;
%     handles.metricdata.factor=factor;

    set(handles.text1,'Enable','off') ;
    set(handles.text2,'Enable','off') ;
    set(handles.text6,'Enable','off')  ;
    set(handles.text7,'Enable','off') ;
    set(handles.text8,'Enable','off')  ;
    set(handles.edit1,'Enable','off')  ;
    set(handles.edit3,'Enable','off');
    set(handles.edit4,'Enable','off')  ;
    set(handles.edit5,'Enable','off') ;
    set(handles.pushbutton1,'Enable','off')  ;
    set(handles.pushbutton4,'Enable','off')  ;
    set(handles.edit8,'Enable','off')  ;
    set(handles.text12,'Enable','off')  ;
    set(handles.edit9,'Enable','off')  ;
    set(handles.text13,'Enable','off')  ;

    set(handles.text25,'Enable','off') ;
    set(handles.text26,'Enable','off') ;
    set(handles.text27,'Enable','off') ;
    set(handles.text28,'Enable','off') ;
    set(handles.text29,'Enable','off') ;

    set(handles.edit18,'Enable','off')  ;
```

```
    set(handles.edit19,'Enable','off');
    set(handles.edit20,'Enable','off')  ;
    set(handles.edit21,'Enable','off') ;

    set(handles.radiobutton1,'Enable','off')  ;
    set(handles.radiobutton2,'Enable','off') ;

    set(handles.text30,'Enable','off') ;
    set(handles.slider2,'Enable','off') ;

    slider_step(1) = 0.5/100;
    slider_step(2) = 1/100;

set(handles.slider2,'sliderstep',slider_step,'max',100,'min',0,'
Value',0);

    handles.metricdata.connection=0;
    handles.metricdata.timerOn=0;
    handles.metricdata.ma=1;
    handles.metricdata.toggle=0;

    handles.metricdata.figCounter=0;
    handles.metricdata.saveNow=0;

    guidata(hObject, handles);



% UIWAIT makes nikiforakisPID wait for user response (see
UIRESUME)
% uiwait(handles.NikGui);




% --- Outputs from this function are returned to the command
line.
function varargout = nikiforakisPID_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
%         refreshdata(handles.axes2);
%         axes(handles.axes2);
%         refreshdata(handles.axes2);
%         %[h,figure] = gcbo;
%         h2=figure;
%         h1 = get(gcf,'CurrentAxes');
%         copyobj(gca,h2);
%
% %           axes(handles.axes2);
%           %get(gcf,'CurrentAxes');
%
%           %  h=gca;
%            saveas(h2,['Figure_'
num2str(handles.metricdata.figCounter)], 'bmp')
%            %hgexport(handles.axes2,['Figure_'
num2str(handles.metricdata.figCounter) '.bmp'])
%
handles.metricdata.figCounter=handles.metricdata.figCounter+1;
%            close(h2)
if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end
           save(['experiment'
num2str(handles.metricdata.figCounter)]);

handles.metricdata.figCounter=handles.metricdata.figCounter+1;
          handles.metricdata.saveNow=0;

        handles.metricdata.saveNow=1;
           guidata(hObject, handles);

    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end
    guidata(hObject, handles);


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

%     factor=handles.metricdata.factor;
```

```
    setPoint = str2num(get(hObject,'String'));
    handles.metricdata.setPoint = setPoint;

    a=handles.metricdata.a;
    a.setSpointTemp(setPoint);
    guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of
edit1 as a double


% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
    setPoint = str2num(get(hObject,'String'));
    handles.metricdata.setPoint = setPoint;



    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    con=handles.metricdata.connection;
    close(figure(24))
    if(con==1)
        a=handles.metricdata.a;
        a.delete;
    end
    disp('GoodBye..')
```

```
    delete(handles.NikGui)


function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    serialPort = get(hObject,'String');
    handles.metricdata.serialPort = serialPort;
    guidata(hObject,handles)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of
edit6 as a double


% --- Executes during object creation, after setting all
properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

    serialPort = get(hObject,'String');
    handles.metricdata.serialPort = serialPort;
    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    kp = str2num(get(hObject,'String'));
    handles.metricdata.kp = kp;
```

```
    a=handles.metricdata.a;
    a.setKTemp(1,kp);
%     pause(1);
guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of
edit3 as a double


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
    kp = str2num(get(hObject,'String'));
    handles.metricdata.kp = kp;


    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    ki = str2num(get(hObject,'String'));
    handles.metricdata.ki = ki;
```

```
    a=handles.metricdata.a;
    a.setKTemp(2,ki);

    pause(1);
    guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of
edit4 as a double


% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
    ki = str2num(get(hObject,'String'));
    handles.metricdata.ki = ki;

    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    kd = str2num(get(hObject,'String'));
    handles.metricdata.kd = kd;

    a=handles.metricdata.a;
    a.setKTemp(3,kd);
```

```
    pause(1);
    guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of
edit5 as a double


% --- Executes during object creation, after setting all
properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
    kd = str2num(get(hObject,'String'));
    handles.metricdata.kd = kd;

    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see
GUIDATA)se

    serialPort = handles.metricdata.serialPort;

    a=PIDarduino(serialPort);

    handles.metricdata.a = a;

    guidata(hObject,handles)

    setPoint= handles.metricdata.setPoint;
    kp= handles.metricdata.kp;
```

```
    ki= handles.metricdata.ki;
    kd= handles.metricdata.kd;
%     factor=handles.metricdata.factor;
    handles.guifig = gcf;


    if(isvalid(a))
        set(handles.text1,'Enable','on') ;
        set(handles.text2,'Enable','on') ;
        set(handles.text6,'Enable','on')  ;
        set(handles.text7,'Enable','on') ;
        set(handles.text8,'Enable','on')  ;
        set(handles.edit1,'Enable','on')  ;
        set(handles.edit3,'Enable','on');
        set(handles.edit4,'Enable','on')  ;
        set(handles.edit5,'Enable','on') ;
        set(handles.pushbutton1,'Enable','on')  ;
        set(handles.pushbutton4,'Enable','on')  ;
        set(handles.edit8,'Enable','on')  ;
        set(handles.text12,'Enable','on')  ;
        set(handles.edit9,'Enable','on')  ;
        set(handles.text13,'Enable','on')  ;

        set(handles.edit6,'Enable','off') ;
        set(handles.pushbutton3,'Enable','off')  ;
        set(handles.text9,'Enable','off') ;


        set(handles.text25,'Enable','on') ;
        set(handles.text26,'Enable','on') ;
        set(handles.text27,'Enable','on') ;
        set(handles.text28,'Enable','off') ;
        set(handles.text29,'Enable','off') ;

        set(handles.edit18,'Enable','on')  ;
        set(handles.edit19,'Enable','on');
        set(handles.edit20,'Enable','on')  ;
        set(handles.edit21,'Enable','off') ;

        set(handles.radiobutton1,'Enable','on')  ;
        set(handles.radiobutton2,'Enable','on') ;

        set(handles.text30,'Enable','on') ;
        set(handles.slider2,'Enable','on') ;


        a.setKTemp(1,kp);
%         a.analogRead(0);
        a.setKTemp(2,ki);
%         a.analogRead(0);
        a.setKTemp(3,kd);
%         a.analogRead(0);
        a.setSpointTemp(setPoint);
%         a.analogRead(0);
```

```
        handles.metricdata.connection=1;



        handles.metricdata.connection=0;
        handles.metricdata.index=0;
        handles.metricdata.temperature=[];
        handles.metricdata.setpointVec=[];
        handles.metricdata.RPM = [];
        handles.metricdata.setPointRPM = [];
        handles.metricdata.PWM = [];
        guidata(hObject,handles)

        handles.tmr = timer('TimerFcn',
{@TmrFcn,handles.guifig},'ExecutionMode','FixedRate','Period',0.
5);
        handles.tmrPlot = timer('TimerFcn',
{@TmrFcnPlot,handles.guifig},'ExecutionMode','FixedRate','Period
',1.5);
        handles.metricdata.timerOn=1;
        start(handles.tmr);
        start(handles.tmrPlot);



    end


    handles.metricdata.a = a;

    guidata(hObject,handles)



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end
    window = str2num(get(hObject,'String'));
    handles.metricdata.window = window;
    handles.metricdata.temperature = [];
    handles.metricdata.setpointVec = [];
    handles.metricdata.RPM = [];
    handles.metricdata.setPointRPM = [];
    handles.metricdata.PWM = [];
    handles.metricdata.index=0;
```

```
    guidata(hObject,handles)

    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end
    guidata(hObject, handles);


% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of
edit8 as a double


% --- Executes during object creation, after setting all
properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
    window = str2num(get(hObject,'String'));
    handles.metricdata.window = window;

    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end
    set(handles.text12,'Enable','on') ;
    set(handles.edit8,'Enable','on') ;
    set(handles.pushbutton1,'Enable','on') ;

    handles.metricdata.toggle=xor(handles.metricdata.toggle,1);
    if(handles.metricdata.toggle==0)
        start(handles.tmr);
        start(handles.tmrPlot);
```

```
        handles.metricdata.timerOn=1;
    else
        stop(handles.tmr);
        stop(handles.tmrPlot);
        handles.metricdata.timerOn=0;
    end

    guidata(hObject,handles)



% --- Executes on key press with focus on pushbutton4 and none
of its controls.
function pushbutton4_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was
pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control,
shift) pressed
% handles    structure with handles and user data (see GUIDATA)

function TmrFcn(src,event,handles)


    handles = guidata(handles);

    window=str2num(get(handles.edit8,'String'));
    index=handles.metricdata.index;
    temperature=handles.metricdata.temperature;
    setpoint=handles.metricdata.setpointVec;
    RPM=handles.metricdata.RPM;
    setPointRPM=handles.metricdata.setPointRPM;
    PWM=handles.metricdata.PWM;
    a= handles.metricdata.a;



    if(index<window)
           index=index+1;
    else
           temperature(1:window-1)=temperature(2:window);
           setpoint(1:window-1)=setpoint(2:window);
           RPM(1:window-1)=RPM(2:window);
           setPointRPM(1:window-1)=setPointRPM(2:window);
           PWM(1:window-1)=PWM(2:window);
    end



%     factor=handles.metricdata.factor;
```

```
    temp=a.getTemp() ;
    pause(0.02);
    RPMTemp=a.getRPM();
    pause(0.02);
    stRPMTemp=a.getSpointRPM();
    pause(0.02);
    PWMTemp=a.getPWM();
    pause(0.02);


    temperature(index) = temp;
    setpoint(index) = str2num(get(handles.edit1,'String'));
    RPM(index)=RPMTemp;
    PWM(index)=PWMTemp;
    setPointRPM(index)=stRPMTemp;




    handles.metricdata.index=index;
    handles.metricdata.temperature=temperature;
    handles.metricdata.setpointVec=setpoint;
    handles.metricdata.RPM=RPM;
     handles.metricdata.PWM=PWM;
      handles.metricdata.setPointRPM=setPointRPM;

    guidata(handles.guifig, handles);


 function TmrFcnPlot(src,event,handles)


      handles = guidata(handles);


        set(handles.text12,'Enable','off') ;
        set(handles.edit8,'Enable','off') ;
        set(handles.pushbutton1,'Enable','off') ;

     temperature=handles.metricdata.temperature;
      window=str2num(get(handles.edit8,'String'));
%      disp('Heeeeeeeeeeeeeeeeeeellllooooooooooo')
     setpoint=handles.metricdata.setpointVec;
     RPM=handles.metricdata.RPM;
     PWM=handles.metricdata.PWM;
     setPointRPM=handles.metricdata.setPointRPM;

     maCoeffs=str2num(get(handles.edit9,'String'));

    temperatureFilt=smooth(temperature,maCoeffs);
    setpointFilt=smooth(setpoint,maCoeffs);
    RPMFilt=smooth(RPM,maCoeffs);
```

```
    PWMFilt=smooth(PWM,maCoeffs);
    setpointRPMFilt=smooth(setPointRPM,maCoeffs);

    hold(handles.axes2,'off') ;


plot(handles.axes2,temperatureFilt(maCoeffs:min(end,window)),'Li
neWidth',2);

    hold(handles.axes2,'on') ;


plot(handles.axes2,setpointFilt(maCoeffs:min(end,window)),'r','L
ineWidth',2);

    legend(handles.axes2,'temperature','setpoint')
    title(handles.axes2,'PID Temperature - Marios Nikiforakis');
    ylabel(handles.axes2,'temperature');
    xlabel(handles.axes2,'time - n');
     set(handles.axes2,'YLim',[15 25]);


     hold(handles.axes3,'off') ;

plot(handles.axes3,RPMFilt(maCoeffs:min(end,window)),'b','LineWi
dth',2);
     hold(handles.axes3,'on') ;

plot(handles.axes3,setpointRPMFilt(maCoeffs:min(end,window)),'r'
,'LineWidth',2);

     legend(handles.axes3,'RPM','RPM setpoint')
    title(handles.axes3,'PID RPM');
    ylabel(handles.axes3,'RPM');
    xlabel(handles.axes3,'time - n');
    set(handles.axes3,'YLim',[0 2000]);



        hold(handles.axes4,'off');

plot(handles.axes4,PWMFilt(maCoeffs:min(end,window)),'k','LineWi
dth',2);
        title(handles.axes4,'Proccess Variable: PWM');
         ylabel(handles.axes4,'%');
        xlabel(handles.axes4,'time - n');
        set(handles.axes4,'YLim',[0 105]);
```

```
    if(handles.metricdata.saveNow==1)

%             save(['experiment'
num2str(handles.metricdata.figCounter)]);
%
handles.metricdata.figCounter=handles.metricdata.figCounter+1;
%             handles.metricdata.saveNow=0;
    end

handles.metricdata.temperatureFilt=temperatureFilt(maCoeffs:min(
end,window));
   handles.metricdata.setpointFilt=
setpointFilt(maCoeffs:min(end,window));

   handles.metricdata.RPMFilt=RPMFilt(maCoeffs:min(end,window));
   handles.metricdata.setpointRPMFilt=
setpointRPMFilt(maCoeffs:min(end,window));


handles.metricdata.PWMFilt=PWMFilt(maCoeffs:min(end,window));

     guidata(handles.guifig, handles);



% --- Executes during object creation, after setting all
properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: place code in OpeningFcn to populate axes2



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles.metricdata.ma=str2num(get(hObject,'String'));
% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of
edit9 as a double
    guidata(hObject, handles);


% --- Executes during object creation, after setting all
properties.
```

```
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
        handles.metricdata.ma=str2num(get(hObject,'String'));
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
    guidata(hObject, handles);



function edit21_Callback(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

%     factor=handles.metricdata.factor;

    setPointRPM = str2num(get(hObject,'String'));
    handles.metricdata.setPointRPM = setPointRPM;

    a=handles.metricdata.a;
    a.setSpointRPM(setPointRPM);
    guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit21 as
text
%        str2double(get(hObject,'String')) returns contents of
edit21 as a double


% --- Executes during object creation, after setting all
properties.
function edit21_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
```

```
% handles    empty - handles not created until after all
CreateFcns called
    setPointRPM = str2num(get(hObject,'String'));
    handles.metricdata.setPointRPM = setPointRPM;



    guidata(hObject,handles)
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    kdRPM = str2num(get(hObject,'String'));
    handles.metricdata.kdRPM = kdRPM;

    a=handles.metricdata.a;
    a.setKRpm(3,kdRPM);
%     pause(1);
guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit20 as
text
%        str2double(get(hObject,'String')) returns contents of
edit20 as a double


% --- Executes during object creation, after setting all
properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
```

```
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
 if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    kiRPM = str2num(get(hObject,'String'));
    handles.metricdata.kiRPM = kiRPM;

    a=handles.metricdata.a;
    a.setKRpm(2,kiRPM);
%     pause(1);
guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit19 as
text
%        str2double(get(hObject,'String')) returns contents of
edit19 as a double


% --- Executes during object creation, after setting all
properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    if(handles.metricdata.timerOn==1)
        stop(handles.tmr);
        stop(handles.tmrPlot);
    end

    kpRPM = str2num(get(hObject,'String'));
    handles.metricdata.kpRPM = kpRPM;

    a=handles.metricdata.a;
    a.setKRpm(1,kpRPM);
%     pause(1);
guidata(hObject, handles);
    if(handles.metricdata.timerOn==1)
        start(handles.tmr);
        start(handles.tmrPlot);
    end

    guidata(hObject,handles)
% Hints: get(hObject,'String') returns contents of edit18 as
text
%        str2double(get(hObject,'String')) returns contents of
edit18 as a double


% --- Executes during object creation, after setting all
properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

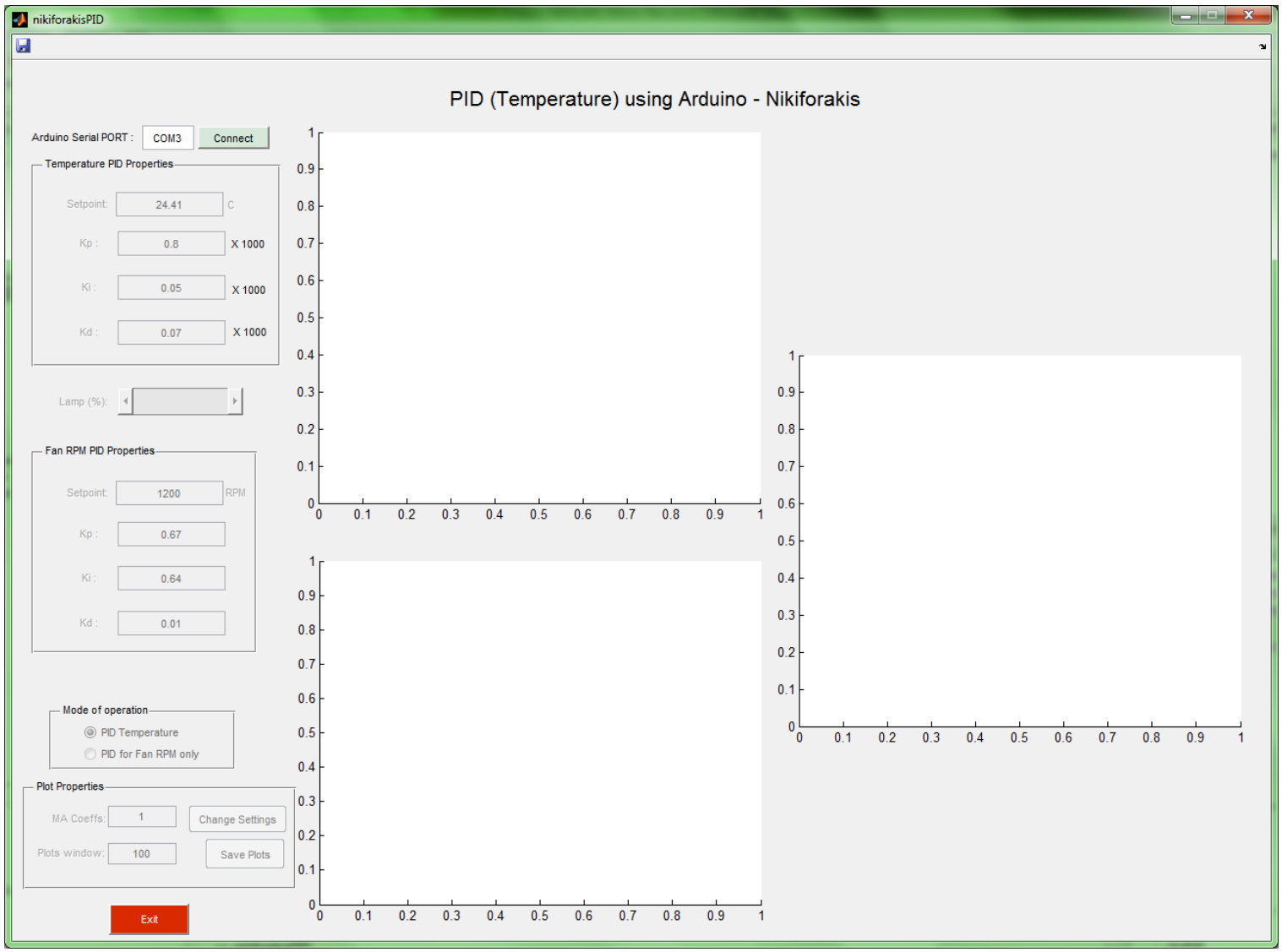
```
% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    set(handles.text25,'Enable','on') ;
    set(handles.text26,'Enable','on') ;
    set(handles.text27,'Enable','on') ;
    set(handles.text28,'Enable','off') ;
    set(handles.text29,'Enable','off') ;

    set(handles.edit18,'Enable','on')  ;
    set(handles.edit19,'Enable','on');
    set(handles.edit20,'Enable','on')  ;
    set(handles.edit21,'Enable','off') ;

     set(handles.text1,'Enable','on') ;
    set(handles.text2,'Enable','on') ;
    set(handles.text6,'Enable','on')  ;
    set(handles.text7,'Enable','on') ;
    set(handles.text8,'Enable','on')  ;
    set(handles.edit1,'Enable','on')  ;
    set(handles.edit3,'Enable','on');
    set(handles.edit4,'Enable','on')  ;
    set(handles.edit5,'Enable','on') ;
    a=handles.metricdata.a;
    a.changeMode('T');
% Hint: get(hObject,'Value') returns toggle state of
radiobutton1


% --- Executes during object creation, after setting all
properties.
function radiobutton1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.text25,'Enable','on') ;
    set(handles.text26,'Enable','on') ;
    set(handles.text27,'Enable','on') ;
```

```
    set(handles.text28,'Enable','on') ;
    set(handles.text29,'Enable','on') ;

    set(handles.edit18,'Enable','on')  ;
    set(handles.edit19,'Enable','on');
    set(handles.edit20,'Enable','on')  ;
    set(handles.edit21,'Enable','on') ;

     set(handles.text1,'Enable','off') ;
    set(handles.text2,'Enable','off') ;
    set(handles.text6,'Enable','off')  ;
    set(handles.text7,'Enable','off') ;
    set(handles.text8,'Enable','off')  ;
    set(handles.edit1,'Enable','off')  ;
    set(handles.edit3,'Enable','off');
    set(handles.edit4,'Enable','off')  ;
    set(handles.edit5,'Enable','off') ;
    a=handles.metricdata.a;
    a.changeMode('R');

    setPointRPM = str2num(get(handles.edit21,'String'));
    handles.metricdata.setPointRPM = setPointRPM;

    a.setSpointRPM(setPointRPM);

    guidata(handles.guifig, handles);
% Hint: get(hObject,'Value') returns toggle state of
radiobutton2


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

    slider_value = get(handles.slider2,'Value');
    a= handles.metricdata.a;
    a.setLamp(slider_value);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
range of slider


% --- Executes during object creation, after setting all
properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called
```

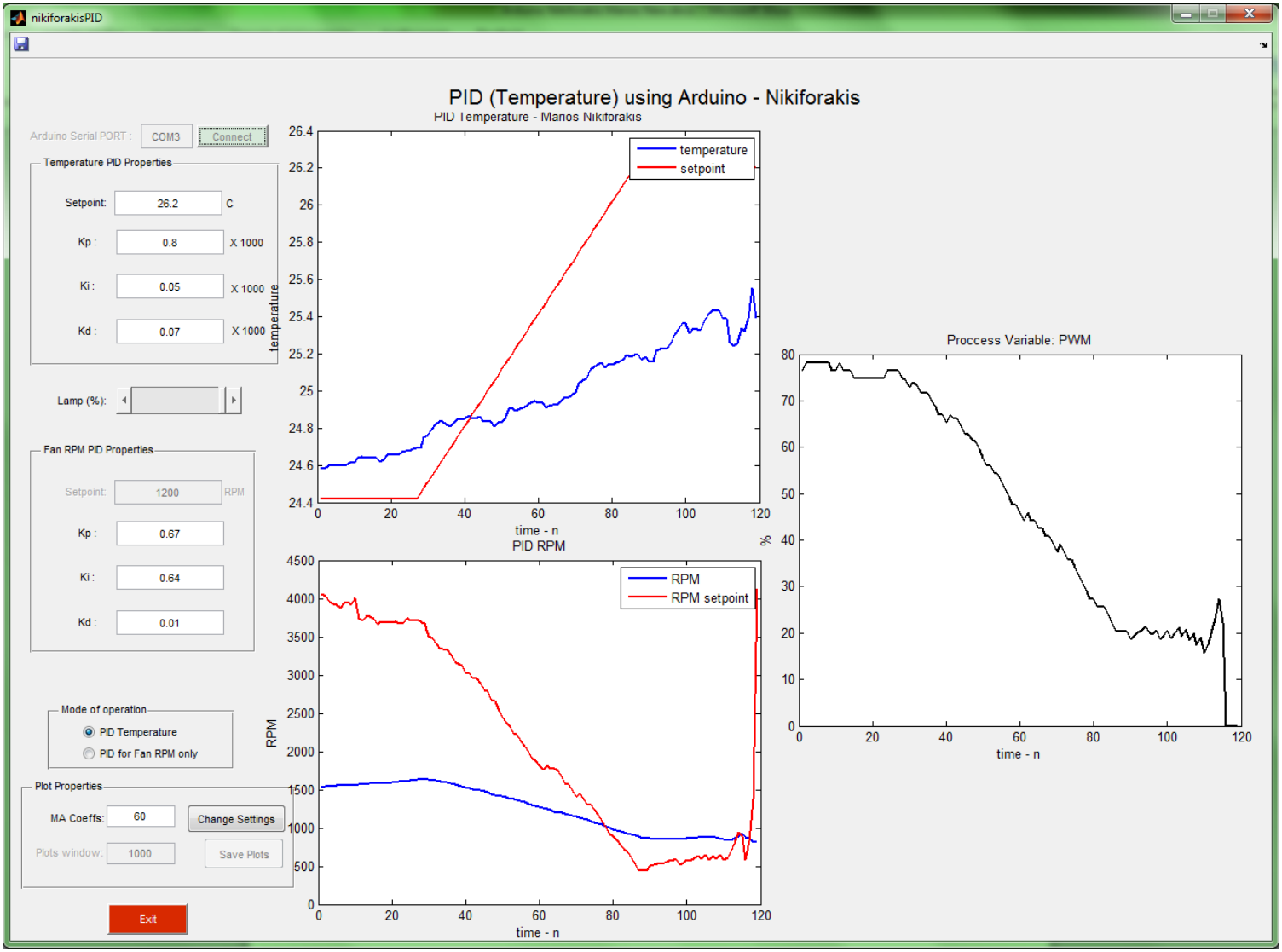
```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1
```

ΠΑΡΑΡΤΗΜΑ Δ

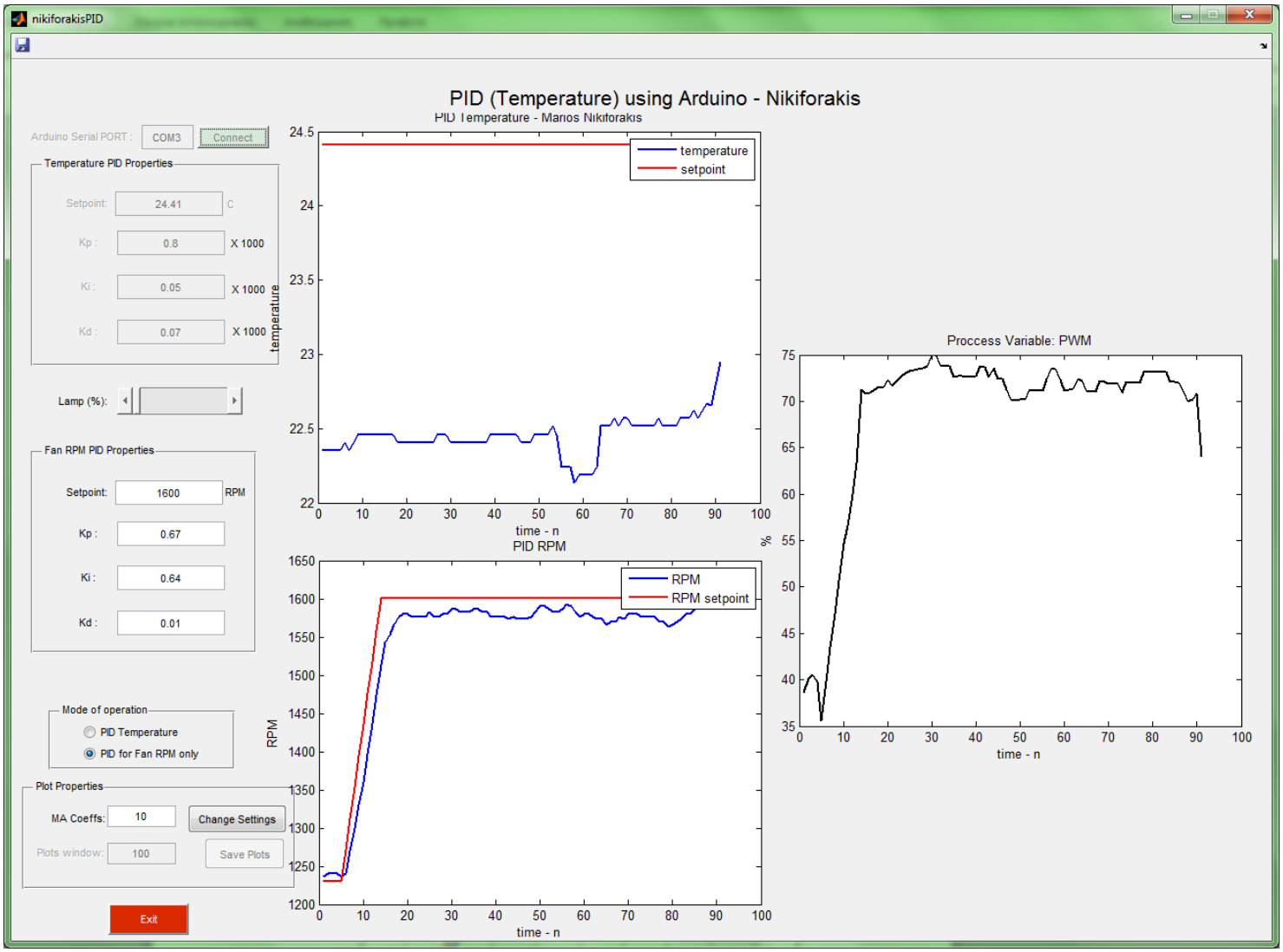
Δ.1 Εικόνες από το Περιβάλλον του GUI



Σχήμα Δ.1 – Γραφικό περιβάλλον πριν την σύνδεση με το Arduino



Σχήμα Δ.2 – Γραφικό περιβάλλον σε Mode Ελέγχου θερμοκρασίας (και εσωτερικά στροφών) mode=1



Σχήμα Δ.3 – Γραφικό περιβάλλον σε Mode Ελέγχου μόνο στροφών. mode=0

ΠΑΡΑΡΤΗΜΑ Ε

Ε.1 Κώδικας για την τύπωση των αποτελεσμάτων στο Matlab

```
function [ ] = plotPID( dataNum )
%PLOTPID Summary of this function goes here
%   Detailed explanation goes here

    close all;
    load(['experiment' num2str(dataNum) '.mat']);

    figure(1)
    subplot(3,1,1)
    L=length(handles.metricdata.temperatureFilt)-1;

plot(0:0.5:L*0.5,handles.metricdata.temperatureFilt,'LineWidth',2);

    hold on ;


plot(0:0.5:L*0.5,handles.metricdata.setpointFilt,'r','LineWidth',2);

    legend('temperature','setpoint')
    title('PID Temperature - Marios Nikiforakis');
    ylabel('temperature [C]');
    xlabel('time [sec]');
    grid on;

    subplot(3,1,2)
    L=length(handles.metricdata.RPMFilt)-1;
    plot(0:0.5:L*0.5,handles.metricdata.RPMFilt,'LineWidth',2);

    hold on ;


plot(0:0.5:L*0.5,handles.metricdata.setpointRPMFilt,'r','LineWidth',2)
;

    hold on ;



    legend('RPM','setpoint')
    title('PID of Fan`s RPM - Marios Nikiforakis');
    ylabel('RPM');
    xlabel('time [sec]');
    grid on;
    subplot(3,1,3)

    plot(0:0.5:L*0.5,handles.metricdata.PWMFilt,'g','LineWidth',2);
    title('Motor PWM - Marios Nikiforakis');
    ylabel('Duty Cycle [%]');
    xlabel('time [sec]');
    grid on;
end
```